Diss. ETH No. 19683
TIK-Schriftenreihe Nr. 125

Enabling Collaborative Network Security with Privacy-Preserving Data Aggregation

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by

MARTIN BURKHART

Master of Science ETH in Computer Science
born February 6, 1978
citizen of Bischofszell, TG

accepted on the recommendation of
Prof. Dr. Bernhard Plattner, examiner
Dr. Xenofontas Dimitropoulos, co-examiner
Dr. Douglas Dykeman, co-examiner

2011

Abstract

Today, there is a fundamental imbalance in cybersecurity. While attackers act more and more globally and coordinated, e.g., by using botnets, their counterparts trying to manage and defend networks are limited to examine local information only. Collaboration across network boundaries would substantially strengthen network defense by enabling collaborative intrusion and anomaly detection. Also, general network management tasks, such as multi-domain traffic engineering and collection of performance statistics, could substantially profit from collaborative approaches.

Unfortunately, privacy concerns largely prevent collaboration in multi-domain networking. Data protection legislation makes data sharing illegal in certain cases, especially if PII (personally identifying information) is involved. Even if it were legal, sharing sensitive network internals might actually reduce security if the data fall into the wrong hands. Furthermore, if data are supposed to be aggregated with those of a competitor, sensitive business secrets are at risk. To address these privacy concerns, a large number of data anonymization techniques and tools have been developed. The main goal of these techniques is to sanitize a data set before it leaves an administrative domain. Sensitive information is obscured or completely stripped off the data set. Sanitized properly, organizations can safely share their anonymized data sets and aggregate information. However, these anonymization techniques are generally not lossless. Therefore, organizations face a delicate privacy-utility tradeoff. While stronger sanitization improves data privacy, it also severely impairs data utility.

In the first part of this thesis, we analyze the effect of state-of-the-art data anonymization techniques on both data utility and privacy. We find that for some use cases only requiring highly aggregated data, it is possible to find an acceptable tradeoff. However, for anonymization techniques which do not destroy a significant portion of the original information, we show that attackers can easily de-anonymize data sets by injecting crafted traffic patterns into the network. The recovery of these patterns in anonymized traffic makes it easy to map anonymized to real data objects. We conclude that network trace anonymization does not properly protect the privacy of users, hosts, and networks.

In the second part of this thesis we explore cryptographic alternatives to anonymization. In particular, we apply secure multiparty computation (MPC) to the problem of aggregating network data from multiple domains. Unlike anonymization, MPC gives information-theoretic guarantees for input data privacy. However, although MPC has been studied substantially for almost 30 years, building solutions that are practical in terms of computation and communication cost is still a major challenge, especially if input data are voluminous as in our scenarios. Therefore, we develop new MPC operations for processing high volume data in near real-time. The prevalent paradigm for designing MPC protocols is to minimize the number of synchronization rounds, i.e., to build *constant-round* protocols. However, the resulting protocols tend to be inefficient for large numbers of parallel operations. By challenging the constant-round paradigm, we manage to significantly reduce the CPU time and bandwidth consumption of parallel MPC operations. We then implement our optimized operations together with a complete set of basic MPC primitives in the SEPIA library. For parallel invocations, SEPIA's operations are between 35 and several hundred times faster than those of comparable MPC frameworks.

Using the SEPIA library, we then design and implement a number of privacy-preserving protocols for aggregating network statistics, such as time series, histograms, entropy values, and distinct item counts. In addition, we devise generic protocols for distributed event correlation and top-k reports. We extensively evaluate the performance of these protocols and show that they run in near real-time. Finally, we apply these protocols to real traffic data from 17 customers of SWITCH (the Swiss national research and education network). We show how these protocols enable the collaborative monitoring of network state as well as the detection and analysis of distributed anomalies, without leaking sensitive local information.

Kurzfassung

Im Bereich Internetsicherheit herrscht ein grundlegendes Ungleichgewicht. Während Angreifer vermehrt global und koordiniert agieren (z. B. durch die Verwendung von Botnetzen), sind die Mittel ihrer Gegenspieler, welche versuchen, Netzwerke zu schützen, auf lokale Informationen beschränkt. Eine Zusammenarbeit über Netzwerkgrenzen hinweg würde die Sicherheit im Internet deutlich verbessern, da Anomalien und Angriffe gemeinsam erkannt werden könnten. Auch allgemeine Aufgaben des Netzwerkmanagements, wie z. B. die Überwachung von Datenflüssen und die Messung der Netzwerk-Performance, würden von einer Zusammenarbeit profitieren.

Oftmals verhindern jedoch Bedenken bezüglich Datenschutz eine Zusammenarbeit über Netzwerkgrenzen hinweg. Datenschutzgesetze verbieten den Austausch gewisser Daten, insbesondere dann, wenn damit Personen identifiziert werden könnten. Aber selbst wenn der Datenaustausch legal wäre, könnte der Austausch von Netzwerkinternas die Sicherheit eines einzelnen Netzes gefährden. Dies wäre vor allem dann der Fall, wenn Daten in falsche Hände gerieten. Je nach Situation könnten Konkurrenten sogar Informationen über wertvolle Geschäftsgeheimnisse erlangen. Um Probleme mit sensitiven Daten zu umgehen, wurden diverse Anonymisierungstechniken entwickelt. Das Ziel der Anonymisierung ist es, heikle Details aus Netzwerkdaten zu entfernen, bevor die Daten ein Netzwerk verlassen. Gewisse Details werden dabei unkenntlich gemacht oder komplett gelöscht. Auf diese Weise bereinigte Daten können ausgetauscht und aggregiert werden. Der grosse Nachteil dieser Techniken ist, dass dabei oft auch der Nutzen der Daten für den eigentlichen Verwendungszweck beeinträchtigt wird. Darum müssen Vorteile für die Sicherheit und Nachteile bezüglich des Nutzens genauestens gegeneinander abgewogen werden.

Im ersten Teil dieser Arbeit analysieren wir sowohl die Sicherheit von gebräuchlichen Anonymisierungsmethoden wie auch ihre Auswirkungen auf den Nutzen von Verkehrsranddaten. Für einige Anwendungsfälle, welche lediglich stark aggregierte Daten benötigen, ist es tatsächlich möglich, einen guten Kompromiss zu finden. Wir zeigen aber auch, dass “sanfte” Anonymisierungstechniken, welche Details lediglich verschleiern, durch Angreifer einfach ausgehebelt werden können. Die Angreifer können beispielsweise gezielt Muster in den Netzwerkverkehr einschleusen, die in den anonymisierten Daten wieder identifiziert werden können. Damit lassen sich anonymisierte zu echten Objekten zuordnen, womit die Anonymisierung gebrochen ist. Wir schliessen daraus, dass Anonymisierung von Netzwerkdaten die Anonymität von Benutzern, Servern und Netzwerken nicht ausreichend schützt.

Im zweiten Teil dieser Arbeit erforschen wir kryptographische Alternativen zur Anonymisierung. Namentlich wenden wir *Secure Multiparty Computation* (MPC) an, um Daten netzwerkübergreifend zu aggregieren. Im Gegensatz zur Anonymisierung liefert MPC informationstheoretische Garantien für die Vertraulichkeit der Daten. Obwohl MPC bereits seit beinahe 30 Jahren erforscht wird, ist es immer noch eine grosse Herausforderung, damit Lösungen zu entwickeln, welche bezüglich Rechenzeit und Kommunikationsaufwand praktikabel sind. Dies ist vor allem dann ein Problem, wenn grosse Datenmengen anfallen, wie dies typischerweise in Netzwerken der Fall ist. Deshalb entwickeln wir MPC Operationen, welche die zeitnahe Verarbeitung von grossen Datenmengen erlauben. Gemäss vorherrschendem Paradigma werden MPC Protokolle so konstruiert, dass sie möglichst wenige Synchronisationsrunden benötigen. Das heisst, es werden sogenannte *constant-round* Protokolle entwickelt. Leider sind die resultierenden Protokolle oft ineffizient, wenn sie in grosser Zahl parallel ausgeführt werden. Indem wir das constant-round Paradigma verlassen, wird es uns möglich, Rechen- und Kommunikationsbedarf von parallelen MPC Operationen erheblich zu reduzieren. Wir implementieren diese optimierten Operationen zusammen mit einem vollständigen Satz von grundlegenden MPC Primitiven in der SEPIA Bibliothek. Die Operationen von SEPIA sind für parallele Abarbeitung zwischen 35 und mehreren hundert Mal schneller als diejenigen von vergleichbaren MPC Frameworks.

Auf SEPIA aufbauend entwickeln wir dann mehrere datenschutzfreundliche Protokolle für die Aggregierung von Netzwerkstatistiken. Unsere Protokolle erlauben die Aggregierung von Zeitreihen, Histogrammen und Entropien, sowie das Zählen von verteilten Objekten. Zusätzlich entwickeln wir

Protokolle für verteilte Event-Korrelation und Top-k Listen. Wir evaluieren die Performance dieser Protokolle ausführlich und zeigen, dass sie in Echtzeit ausführbar sind.

Zu guter Letzt testen wir unsere Protokolle mit echten Netzwerkdaten von 17 Kunden von SWITCH (dem Forschungsnetz und ISP der Schweizer Hochschulen). Wir demonstrieren, wie unsere Protokolle eine kollaborative und datenschutzfreundliche Überwachung der Netze sowie eine Zusammenarbeit bei der Detektion und Analyse von verteilten Anomalien ermöglichen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This is why I loved technology: if
you used it right, it could give
you power and privacy.

Cory Doctorow

In the fable “The Blind Men and the Elephant” [125] by the American poet John Godfrey Saxe, six blind men from Indostan heard of a thing called “an elephant” but did not know what it was. To satisfy their minds, they went to observe a real elephant. Each of them approached the elephant from a different side and came to his own conclusion about what an elephant is. The one that touched the side found “It’s very like a wall!”, while the one examining the tusk shouted “It’s very like a spear!”. As illustrated in Figure 1.1, the knee was judged to be like a tree, the trunk like a snake, the ear like a fan, and the tail like a rope. When they finally came together to discuss their observations they had a long dispute about what an elephant was. However, as Saxe put it: *“Though each was partly in the right, all were in the wrong!”*

Is the Internet an Elephant?

The situation in today’s Internet research bears quite some similarity with the blind men’s fable. The Internet has grown to be a veritable elephant over the past 20 years, driven mainly by global commercialization in the 1990s and 2000s. According to the Internet Systems Consortium (ISC) [71], there were

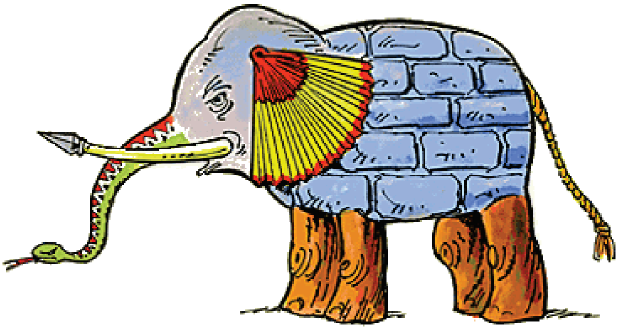


Figure 1.1: *The elephant as imagined by the blind men. (Image ©Word Info.)*

only 56,000 hosts connected to the Internet in 1988. In 1992 it passed the 1 million hosts mark, in 1996 the 10 million, and in 2001 the 100 million mark. In January 2011, there were already more than 800 million hosts connected. Today, the Internet is rapidly expanding to include mobile devices, such as smart phones. According to a report by Initiative [70], the mobile phone will overtake the computer as the most common web access device worldwide by 2013, with an estimated 1.82 billion internet-enabled phones in use.

Though being entirely man-made, the distributed nature, huge size, and strong dynamics of the Internet have made it impossible to describe its state in simple terms and from a single point of view. It has become a complex phenomenon people have opinions about. Consequently, methodologies used today in Internet measurement research are often empirical, capturing large amounts of data that are later analyzed in-depth and have to be interpreted.

Similar to the reports of the blind men, measurement studies are limited in scope and accuracy. Each study examines the Internet at a specific location, e.g., a university network, at a specific point in time, using specific tools. Typically, results from these studies are generalized to some degree, i.e., they are believed to reflect at least parts of the Internet. However, there are many parameters constraining generalization. First of all, the Internet is constantly evolving. It is difficult enough to obtain and process high quality traffic data. But to get data spanning months or even years, allowing to analyze temporal evolution and trends, is close to impossible. Moreover, measurements in one network cover just a tiny fraction of the global Internet. To compensate this, researchers from CAIDA[1] proposed to establish periodic "Day in the Life of the Internet" events [78] with the goal to measure the Internet core simultaneously from all over the world. Such a setup would allow correlation of different measurements at the same point in time. However, depending on what we measure and where we are, the Internet might actually *look* different. For instance, most studies are carried out in *academic* setups, which makes it difficult to argue about *residential* networks [94]. Also, statistical methods used in anomaly detection or traffic classification are prone to learning site-specific patterns (e.g., [110]). Due to a lack of reference data sets, it often remains unclear how well these methods generalize. Even seemingly easy questions, such as "How big is the Internet?", are hard to answer. Odlyzko shows that the Internet growth rate, although substantial, was severely overestimated (by about a factor of 10) in the late 1990s, leading to an inflation of the dot-com and telecom bubbles [104].

Unfortunately, also the dark side of the Internet has grown dramatically over the past years. The cybercrime scene has professionalized and governments around the world are preparing for cyber-warfare [35]. Recent studies [77] show that coordinated wide-scale attacks are prevalent: 20% of the studied malicious addresses and 40% of the IDS alerts are attributed to coordinated wide-scale attacks. According to the 2009 CSI Computer Crime and Security Survey [46], 23% of responding organizations found botnet zombies, 29% experienced DoS attacks, 14% dealt with webpage defacement, and 14% report system penetration by outsiders. Moreover, there is an imbalance in the cyber arms race. While cybercriminals act globally and are well coordinated, e.g., by using botnets, operators protecting their networks often have to resort to local information only. Yet, many network security and monitoring problems would profit substantially if a group of organizations aggregated their local network data. For example, IDS alert correlation [89, 154, 156], requires the joint analysis of local alerts. Similarly, aggregation of local data is useful for alert signature extraction [109], collaborative anomaly detection [117], multi-domain traffic engineering [93], and detecting traffic discrimination [138]. Even the difficult problem of detecting fast-fluxing P2P botnets becomes tractable with cross-AS collaboration [101].

All these examples clearly illustrate the need for large-scale distributed Internet measurements. Only by combining many individual pieces, we will get the big picture of the Internet and the threats therein.

[1]The Cooperative Association for Internet Data Analysis (CAIDA) promotes cooperation in the engineering and maintenance of a robust and scalable Internet (`http://www.caida.org`).

Privacy in Network Traffic Data

Now one might ask: If data sharing brings all these benefits, why is it not done in practice? Parts of the problem are certainly a lack of standards and coordination. That is, data captured in different networks might not be directly comparable due to different tools, data formats, or measurement techniques. Another issue is the large amount of data involved. The storage and processing of traffic data requires substantial resources, especially if packet data are involved. Therefore it is not trivial to ship data around or gather it in a central repository. However, these obstacles can be overcome with community initiative, coordination, and engineering [41, 69, 97], as has been done in other data-driven disciplines such as astronomy or particle physics [148]. By far the more difficult problem is how to address *privacy concerns* with network data.

Traffic data contain very sensitive information about users, servers, and networks. With packet data, the entire network communication of a user is captured. But even if payload is stripped away, as with packet headers or NetFlow data [40], stored IP addresses still allow the identification of users and hosts. The associated connection information allows the creation of precise communication profiles, e.g., containing information about who is communicating with whom and when, or which websites a person visits.

From a legal perspective, network data are “personal data”. For instance, European law [51,52] defines personal data as data identifying a person either directly or indirectly (i.e., through the use of additional information in possession of third parties). To this category belong, e.g., IP addresses and user profiles. The law restricts the processing allowed on personal data and mandates anonymization for subsequent storage or before further processing. Ohm et al. discuss many subtleties regarding legal issues in network research [106], pertaining mainly to U.S. law. They find that many research papers fall short of clear legal compliance due to a disconnect between legislation and current academic practice. For example, the application of data reduction or anonymization does not necessarily legalize analyses. Furthermore, Burstein et al. [32] point out that the flow of data to be analyzed poses additional problems. In the U.S., researchers (mostly working for governmental institutions) are, in principle, not allowed to analyze data from entities regulated by the Stored Communications Act (SCA), such as commercial ISPs. As a result, there is much uncertainty in the networking community and operators often choose the safe way, i.e., they completely refrain from data sharing. Data sharing among *international* partners brings up the additional complication of heterogeneity in international data protection legislation.

Even if ambiguity in legislation is fixed in the future, organizations will not easily engage in data sharing. After all, there are internal network data at stake. Security policies might deny sharing because of a high risk of information disclosure. Even though collaboration might be useful, organizations have to carefully balance benefits with risks of potential damage. Even anonymized data may contain topological information, hint at particular services deployed, or reveal policies in place. In a competitive setting, overall statistics might reveal information about a participant's customer base. In summary, the situation is intricate. Even if the men inspecting the Internet elephant are not blind, they refuse to exchange their observations from fear of privacy breaches.

This is exactly the starting point of the present thesis. *Our goal is to devise methods that enable cross-organizational collaboration even on sensitive network data.* Two achieve this, we consider two different paradigms:

Anonymization: With anonymization, the process of collaboration is to first anonymize local data. Then, anonymized data are exchanged, either bilaterally or by using some sort of central (or distributed) repository. Data analyses are then run on the entirety of data instead of local data only.

Secure Multiparty Computation (MPC): With MPC, sensitive data remain stored locally, e.g., in a local database. Using secret sharing techniques, random pieces of local data (shares) are distributed to a set of computation nodes. Together, they perform a distributed cryptographic protocol on the shares. In the end, only the final analysis result is made public and announced to input data providers.

In Part I of this thesis, we thoroughly assess the impact of state-of-the-art anonymization techniques for IP addresses on data utility and privacy. In Part II, we propose to use MPC for many problems in network security and monitoring. While this is much more costly in terms of computation and communication, it provides cryptographic privacy guarantees and allows the use of ungarbled data.

The SWITCH Traffic Data Repository

The availability of real-world traffic data was essential for a sound evaluation of our work. Therefore, we briefly explain our data set before going into the details of Part I and Part II.

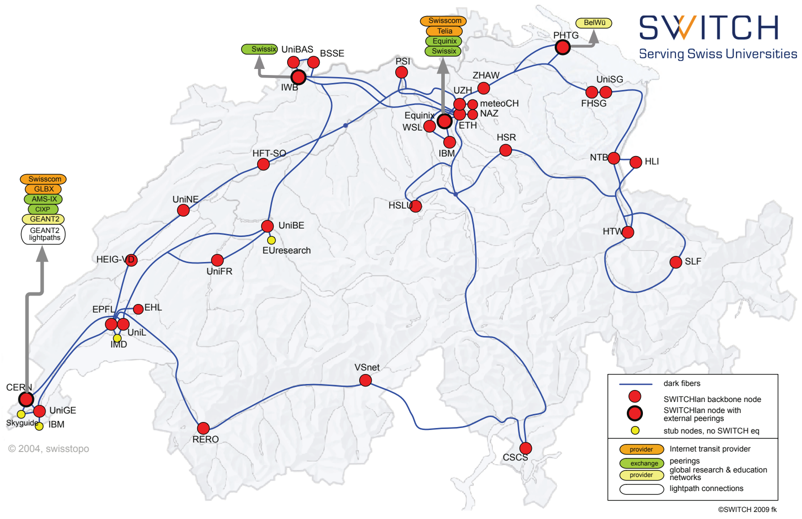


Figure 1.2: *The SWITCH backbone topology as of May 2009.*

The data we used have been captured from SWITCH (AS559), the Swiss research and education network [137]. SWITCH currently has 47 customer networks, including all Swiss universities, various research labs (e.g., IBM, PSI, CERN), and several governmental institutions. An overview over the SWITCH network topology is depicted in Figure 1.2. The Communication Systems Group (CSG) has been collecting NetFlow [40] traces from the border routers of SWITCH since 2003. Today, about 120 million flows per hour are captured on average, reflecting 2-4 terabytes of traffic. The address range managed by SWITCH amounts to 2.3 million IP addresses. The use of data stored by the CSG is regulated by non-disclosure agreements. Furthermore, people processing the data must be supervised by CSG members, work at ETH premises, and projects have to be approved by SWITCH.

1.1 Part I: Network Data Anonymization

Many tools and techniques for anonymizing IP addresses have been developed (e.g., [99, 108, 131]). The basic principles are to blackmark, permute, or truncate IP addresses. Permutations can either be random or (partially) prefix-preserving [60], i.e., common prefixes of arbitrary length are preserved under the permutation function. The basic techniques are discussed in detail in chapter 2.

Anonymization is Unexplored

Alas, the creation of an anonymization policy for a specific data set typically involves a mix of expertise, heuristics, and gut feelings [108]. The security guarantees of particular methods are difficult to quantify. Furthermore, data owners mandating an anonymization policy have to supervise the use of anonymized data and negotiate with data users whether a certain type of analysis is still possible, and if so, to what degree [4]. On the one hand, the more information is removed from data, the better privacy is protected. On the other hand, removal of information makes data less useful for analyses. Tuning this privacy-utility tradeoff is very delicate, especially if quantitative measures are missing. The IETF addresses this problem in the specification of IPFIX, the future format for network flow data and successor of NetFlow [19, 113]. It is the goal of the IETF to require anonymization support on routers in order to be able to directly export anonymized data and avoid privacy breaches during transport, processing, and storage of data. However, due to insufficient understanding of the basic properties of existing techniques, the requirement for anonymization support is not qualified with “must” but with “may” [113, §6.7]. This is in line with Ohm et al., who demand a thorough assessment of traditional strategies for privacy protection. A clearer understanding of these techniques is a first step towards fixing the gap between legislation and research practice [106].

Assessing the Privacy-Utility Tradeoff

In the first part of this thesis, we take up this challenge and shed light on quantitative utility and privacy properties of state-of-the-art anonymization techniques for IP addresses. We evaluate the utility of anonymized data by performing statistical network anomaly detection on original and anonymized NetFlow data in chapter 3. Network anomaly detection is a prominent application of NetFlow data and has attracted a lot of research interest in the last years (e.g., [21, 83, 133, 139, 146]). Interestingly, some of the approaches were evaluated on anonymized data from the Abilene network, which were anonymized by truncating 11 bits from IP addresses [82,83,133]. Presumably, such a strong anonymization had some impact on detection results. The authors of [133] briefly discuss the problem, but conclude that they "could not imagine any scenario where anonymization could hide an anomaly". Contrarily, our results indicate that data utility for detecting scans and denial of service attacks degrades substantially under truncation, especially when distinct count metrics are applied. Only the detection of network-wide volume anomalies, such as outages or alpha flows, is not impacted by anonymization.

To evaluate the privacy guarantees in presence of a worst-case attacker, we perform active traffic injection attacks in chapter 4. The goal of these attacks is to inject known traffic patterns into networks and recover these patterns from anonymized data, allowing to de-anonymize IP addresses. Our results show that it is indeed easy to perform traffic injection attacks in practice, also in large networks and even though secondary flow fields were randomized and coarse-grained to blur patterns. Specifically by stretching injected patterns over time, the attacker can evade detection. The success of these attacks and the impossibility of defending against them leads us to call into question the role of anonymization as a complete solution to the problem of data protection (see chapter 7). It must be applied together with legal and social means to achieve the aims of better data sharing for research and operations.

In chapter 5 we also analyze the specific privacy-utility tradeoff of IP address truncation, which goes beyond permutation and actually deletes information from traces. Interestingly, there is an asymmetry between IP addresses assigned to internal and external address ranges. For internal addresses, fewer bits need to be truncated to provide acceptable privacy, simply because networks are more densely filled with active addresses. For instance, by truncating 8 bits, all addresses within the same /24 network become indistinguishable. The external address range is much sparser, requiring roughly 7 bits of truncation more for the same privacy level. Regarding data utility, we find that entropy metrics exhibit better robustness against truncation than count metrics. Only three combinations of entropy metrics and truncation strength achieve acceptable utility *and* privacy at the same time.

1.2 Part II: Privacy-Preserving Data Sharing using MPC

In the second part of this thesis, we propose to use the radically different approach of secure multiparty computation (MPC) for privacy-preserving network management. The motivation for this is to escape the vexatious privacy-utility tradeoff. *Nobody wants to compromise on privacy. Nobody wants to compromise on utility, either.* Part II is devoted to exploring how far we can go with this strong attitude of zero compromise.

Challenges with MPC

For almost thirty years, MPC techniques [153] have been studied for solving the problem of jointly running computations on data distributed among multiple parties, while provably preserving data privacy without relying on a trusted third party. In theory, any computable function on a distributed data set is also securely computable using MPC techniques [68]. However, designing solutions that are practical in terms of running time and communication overhead is far from trivial. For this reason, MPC techniques have mainly attracted theoretical interest in the last decades. Recently, optimized basic primitives, such as comparisons [47,103], make progressively possible the use of MPC in real-world applications. Remarkably, the first real-world application of MPC, a sugar-beet auction, was demonstrated in 2009 [17].

Adopting MPC techniques to network monitoring and security problems introduces the important challenge of having to deal with voluminous input data that require online processing. For example, anomaly detection techniques typically require online monitoring of how traffic is distributed over ports or IP addresses. Such input data impose stricter requirements on the performance of MPC protocols than, for example, bids in a distributed auction. In particular, network monitoring protocols must process potentially thousands of input values while meeting *near real-time* guarantees[2]. This is not presently possible with existing MPC frameworks.

The SEPIA Library

We design, implement, and evaluate SEPIA, a library for efficiently aggregating multi-domain network data using MPC. The foundation of SEPIA is a set of optimized MPC operations, implemented with performance of parallel execution in mind.

It is a common believe that the running time of MPC protocols is determined mainly by the number of synchronization rounds they require [9,65].

[2]We define *near real-time* as the requirement of fully processing an x-minute interval of traffic data in no longer than x minutes, where x is typically a small constant. For our evaluation, we use 5-minute windows, which is a frequently-used setting.

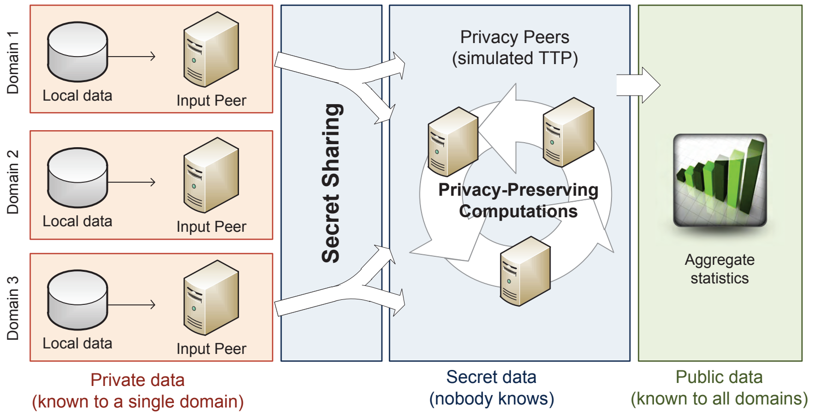


Figure 1.3: *Deployment scenario for SEPIA.*

Therefore, theorists have adopted the paradigm of designing *constant-round* protocols, for which the number of rounds does not scale with input size. While the number of rounds is certainly the bottleneck if only a few operations are performed, we show that the situation changes when many operations are performed at the same time. Then, the cost for starting a synchronization round is amortized over hundreds or thousands of operations. By not enforcing protocols to run in a constant number of rounds, we design MPC comparison operations that require up to 80 times less distributed multiplications and, amortized over many parallel invocations, run much faster than state-of-the-art constant-round alternatives (see chapter 9).

A typical setup for SEPIA is depicted in Fig. 1.3, where individual networks are represented by one *input peer* each. The input peers distribute shares of secret input data among a (usually smaller) set of *privacy peer*s using Shamir's secret sharing scheme [129]. The privacy peers perform the actual computation and can be hosted by a subset of the networks running input peers but also by external parties. Finally, the aggregate computation result is sent back to the networks. We adopt the semi-honest adversary model, hence privacy of local input data is guaranteed as long as the majority of privacy peers is honest. A detailed discussion of SEPIA's design and our security assumptions is presented in chapter 10.

Privacy-Preserving Protocols

To enable anomaly detection in a distributed setting, we design four MPC protocols on top of SEPIA's basic primitives in chapter 11. The *entropy protocol* allows input peers to aggregate local histograms and to compute the entropy of the aggregate histogram, which is commonly used for anomaly detection. Similarly, the *distinct count protocol* finds and reveals the number of distinct, non-zero aggregate histogram bins. The most general protocol is our *event correlation protocol*, which correlates arbitrary events across networks and only reveals the events that appear in a minimum number of input peers and have aggregate frequency above a configurable threshold. Finally, our *top-k protocol* PPTKS estimates the global top-*k* items over private input lists using a novel approach based on a sketch data structure that trades off a slightly lower estimation accuracy for lower MPC computational overhead. In addition, we implement the four protocols, along with a state-of-the-art *vector addition protocol* for aggregating additive time series or local histograms.

In chapter 12, we evaluate our protocols in realistic settings and with real traffic data from the SWITCH network. Our evaluation shows that the protocols are indeed practical and can be used in various scenarios. For instance, we demonstrate that aggregating port histograms with 65K bins, computing the port entropy, or generating top-k reports from 180K distinct IP addresses is possible in near real-time, even if input data are distributed among 25 participants.

Distributed Troubleshooting in Practice

We make a final step towards practice in chapter 13 by applying our protocols to traffic data from 17 SWITCH customer networks collected during the global Skype outage in August 2007. We show how the networks can use SEPIA with our protocols to collaboratively detect, investigate, and characterize such anomalies. In particular, organizations can easily determine the scope and learn details of distributed anomalies. They can assess how much they are affected compared to other organizations and sometimes even profit from early warnings. The protocols aid in identifying root causes, which is vital for taking countermeasures. Also, local anomalies can be identified as such, by learning that other organizations are not affected.

1.3 Contributions

This thesis makes several contributions to research in the field of privacy-preserving technologies. Our work was published in peer-reviewed conferences, workshops, and journals.

1. **Quantification of the privacy-utility tradeoff in anonymization:** We quantify the privacy-utility tradeoff for IP address anonymization techniques by studying the utility of anonymized traffic data for network anomaly detection [26] and measuring host anonymity [27].

2. **Evaluation of worst-case attacks on anonymization:** We demonstrate that active injection attacks are easy to perform even in large-scale networks. Potential countermeasures aimed at obscuring injected patterns are largely ineffective. Hence, injection attacks are a real threat to any anonymization scheme based on permutations [30].

3. **Application of secure multiparty computation to networking:** We propose to apply MPC techniques to problems in network security and monitoring. We show that the widely-used constant-round MPC paradigm is not suitable when many MPC operations are executed in parallel. Instead, protocols should focus on optimizing both the number of rounds and the number of required MPC multiplications. This approach leads to better overall performance. Following this insight, we optimize basic MPC primitives for the main challenges in networking: voluminous input data and near real-time processing [31].

4. **Development of privacy-preserving protocols:** We develop privacy-preserving protocols for aggregation of network statistics, correlation of distributed events, and distributed top-k queries. We thoroughly evaluate the protocols with respect to resource requirements, showing that they are efficient enough for near real-time use in typical scenarios [28, 29, 31].

5. **Implementation of the SEPIA library:** We implement our optimized MPC primitives along with our privacy-preserving protocols in the SEPIA library. The library and its source code are made available publicly [128].

6. **Demonstration of MPC-based collaboration benefits**. We apply our protocols to traces from real networks to finally bridge the gap between

MPC theory and networking practice. We show how the networks could have collaborated to detect and timely mitigate a real-world large-scale anomaly [28, 31]. This demonstrates that independent domains can have concrete benefits from MPC-based collaboration.

Part I

Network Data Anonymization

Chapter 2

Anonymization Techniques

The right to be let alone is indeed
the beginning of all freedom.

Justice William O. Douglas

There are many different techniques and tools for anonymization of network traffic data. In this chapter we introduce the most important techniques with a special focus on techniques for anonymizing IP addresses. For NetFlow data [40], which is the type of data we will be using subsequently, payload information is not available in traces. The most sensitive fields in NetFlow records are IP addresses, because they can be mapped to users, servers, and networks. In many companies, IP addresses can be easily mapped to desktop computers and their owners. Profiling these IP addresses leads to detailed user behavior profiles and is subject to data protection legislation. IP addresses may also represent servers or gateways of a company. Statistics about these important network infrastructure elements are likely to be protected by internal network security policies. Also, statistics about entire subnets are sensitive, especially if these subnets match individual customers of an ISP. By using the term *host privacy* we relate to all privacy issues on an IP address level, including the privacy of users, servers, and subnets.

Many simple tools like TCPdpriv [99], CryptoPAn [60], CANINE [88], and Tcpmkpub [108] come with predefined options for anonymizing certain fields in specific data formats, e.g., packet traces. Anonymization frameworks like FLAIM [131] and Anontool [63] provide a comprehensive collection of

IP Address	Truncation (16 bits)	Random Permutation	Prefix-Pres. Permutation	Partial Prefix-Pres. Perm. (16 bits)
129.132.91.35	129.132.0.0	112.4.23.73	22.5.99.76	73.9.8.1
129.132.91.177	129.132.0.0	62.12.96.67	22.5.99.41	73.9.181.17
129.132.8.37	129.132.0.0	205.72.5.18	22.5.181.92	73.9.1.230
152.88.3.90	152.88.0.0	2.14.12.133	110.27.20.1	18.7.18.133
152.96.99.2	152.96.0.0	19.0.111.20	110.9.0.12	24.125.43.6
82.130.102.115	82.130.0.0	12.171.92.3	145.21.5.19	145.213.2.77

Table 2.1: *Examples of IP address anonymization.*

anonymization techniques that can be flexibly applied to various fields and allow the definition of fine-grained anonymization policies. FLAIM even supports extension with plugins to handle new data formats.

2.1 IP Addresses

The most commonly employed IP address anonymization techniques are blackmarking, truncation, random permutation, prefix-preserving permutation, and partial prefix-preserving permutation. An illustrative example for each technique, except blackmarking which is trivial, is given in Table 2.1.

Blackmarking

Blackmarking is the simplest of all studied anonymization techniques. It replaces all IP addresses in a trace with the same value. Several traces from the Internet traffic archive (LBNL) are anonymized with blackmarking. Please refer to the UCRchive [141] for a comprehensive list of available traces.

Truncation

Truncation replaces a number of least significant bits of an IP address with 0. Thus, truncating 8 bits would replace an IP address with its corresponding class C network address. The traces from the Abilene network, which have been used to evaluate numerous anomaly detection studies (e.g., [82,83,133]), are anonymized with truncation of 11 bits.

Random permutation

Random permutation translates IP addresses using a random permutation that does not preserve the prefix structure. Since permutation creates a one-to-one mapping, the number of distinct IP addresses is the same. A special case of random permutation is the renumbering of IP addresses (e.g., TCPdpriv with level 0). Packet Traces from UCLA CSD, as well as several traces from the Internet traffic archive (LBNL) are sanitized using random permutation.

Partial prefix-preserving permutation

As proposed in [108], partial prefix-preserving permutation (PPP) permutes the host and network part of IP addresses independently. It preserves the prefix structure in a trace at one specific prefix length p, and at the level of IP addresses. PPP is a popular technique that is used for anonymizing traces from the Passive Measurement and Analysis project (PMA) and the Internet traffic archive (LBNL). PMA uses $p = 12$ and $p = 16$. Moreover, level 1 of TCPdpriv corresponds to $p = 16$.

Prefix-preserving permutation

Prefix-preserving permutation permutes IP addresses so that two addresses sharing a common real prefix of any length, also share an anonymized prefix of equal length (see e.g., [60]). This technique is applied to traces from CAIDA and CRAWDAD.

2.2 Secondary Fields

In this section, we discuss techniques applied to fields other than IP addresses, such as port numbers or timestamps. Of course, generic methods like black-marking or random permutation can be applied to many fields and also work in combination with other techniques. Some of the techniques described below are used in chapter 4 for obscuring injected fingerprints. However, we do not evaluate these techniques in the same way as we do for IP address anonymization techniques.

Hashing

Hashing of values is a special form of random permutation. It is also applicable to larger fields, e.g., the payload. If seeded hash functions are used and the seed is kept secret, the adversary is kept from brute-forcing values. At the same time, it allows the owner of the data to retain mapping information without storing large mapping tables. Although usually rare, hashing can introduce collisions.

Classification

Classification simply partitions the range of possible values into pre-defined classes. For instance, a popular way of anonymizing port numbers is to classify them into well-known (< 1024) and dynamic ports (≥ 1024).

Time Unit Annihilation

Timing fields can be broken down into year, month, day, minute, seconds, and milliseconds parts. The annihilation of time units corresponds to blackmarking of a specific part of timing, e.g., milliseconds and seconds can be annihilated to avoid the resolution of records below minute accuracy. Of course, this could be combined with classification, e.g., to bucketize all timestamps in 10 minute slots. Similarly, the year and month part could be annihilated to hide when a trace was collected.

Enumeration

Enumeration sorts all occurring records by value and replaces the values by the rank of their record. It is usually applied to timestamp fields to preserve the logical order of events but remove the absolute timing information. In that case, the "rank" has to be mapped back to a valid timestamp. This could for instance be done by choosing a random offset and for each rank increment the timestamp by one millisecond.

Random Shift

Sometimes it may be important to preserve the distances between values but hide their absolute values, for instance to assess how far apart two events are temporally. For this, a random shift is performed by adding a random offset to the values.

Chapter 3

Impact of Anonymization on Data Utility

Nothing can have value without being an object of utility.

Karl Marx

In this chapter, we investigate how five popular IP address anonymization techniques impact statistical network anomaly detection on flow data. To make our discussion more systematic, we first discuss the design space for detection metrics, and show how the different anonymization techniques diminish this design space.

3.1 Granularity Design Space

Up to now the metrics used for anomaly detection, and traffic analysis in general, have been designed in an ad-hoc manner, based on (i) the characteristics of the data set under study, and (ii) the type of traffic characteristic one is interested in. Prominent examples of such metrics are the well-known volume metrics, such as byte, packet, and flow counts, which are simply computed over all traffic in a given trace. Lakhina et al. [82] group the *anonymized* traffic from the Abilene network into Origin-Destination (OD) flows before analyzing it further with Principal Component Analysis. Whereas, in [147, 152]

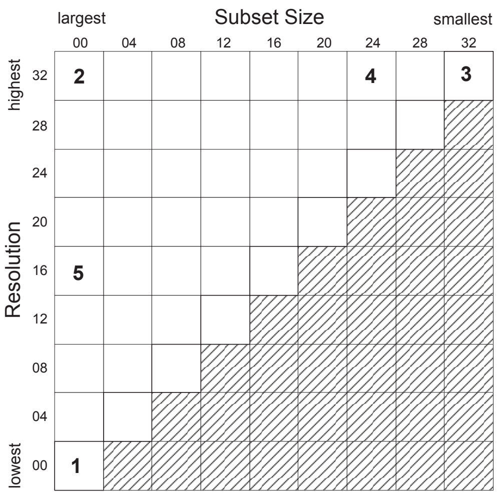


Figure 3.1: *Granularity design space for metrics used in statistical anomaly detection.*

host-based metrics such as IP address entropy or the number of active connections are used for host profiling in a clustering algorithm.

To investigate the impact of anonymization in a systematic manner, however, it is necessary to explore the whole granularity design space for traffic analysis metrics. The dimensions of this design space are similar to dimensions in ordinary demographic statistics. Consider, for example, we wanted to do a statistical survey about average income. The first thing to be defined is the population size of the survey. That is, do we survey the entire country, just a district, or even a single city? At the time when we produce statistics from the survey, we need to take another choice regarding the granularity of the statistics: Do we calculate the average income per person, per household, or per town? Only by picking both a population size and a granularity, we arrive at a meaningful number, representing, for instance, the average income *per household* in *the canton of Zurich*. Similarly, the granularity design space for traffic statistics has two dimensions:

Subset size: The size (or span) of the network that is to be analyzed. This corresponds to the population size in the above example. When analyz-

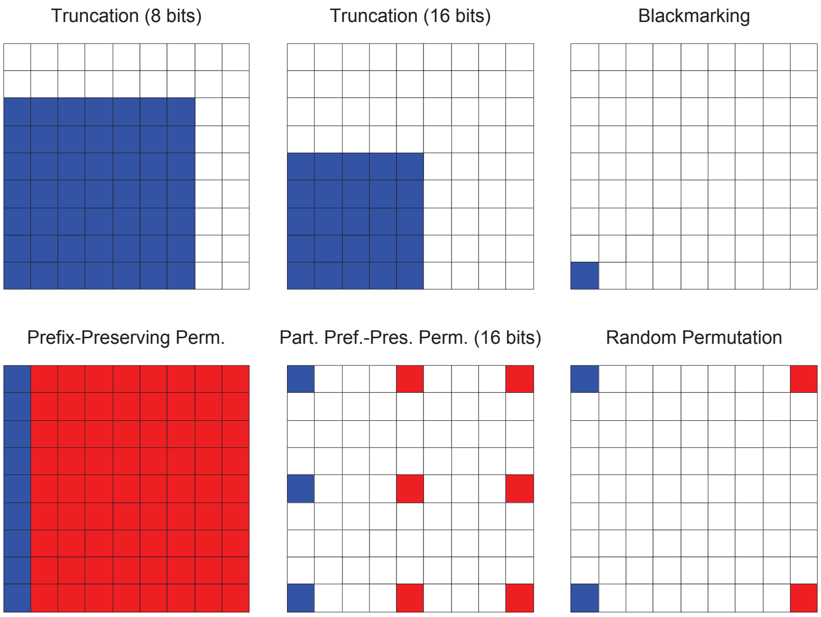


Figure 3.2: *Resolutions and subset sizes available with different anonymization techniques.*

ing backbone data for example, we can analyze all traffic, or we can focus on a specific subnet in the backbone.

Resolution: The address granularity at which the traffic is analyzed. We can select a very high resolution of IP addresses if one is interested in profiling hosts, or a low resolution of Autonomous Systems (or OD flows) if one is interested in more global events.

We give a matrix representation of the granularity design space in Figure 3.1. The x-axis ranges from the largest subset size of all traffic (00 bits), to the smallest possible subset size of a single IP address (32 bits). Likewise the y-axis ranges from the highest resolution of individual IP addresses (32 bits) to the lowest resolution of all traffic (00 bits). The design space matrix can be divided in three sections: the upper triangle, the diagonal, and the lower triangle. Traditional volume metrics fall in one of the cells on the diagonal, where the subset size equals the resolution (e.g., we select all traffic from a /16 subnet and compute the flow counts over all traffic coming from that /16 network). The upper triangle features metrics where the resolution level is smaller than the size of the selected traffic subset. Such metrics, e.g, the number of unique IP addresses in the inbound traffic to a selected /24 subnet, are also frequently used for traffic analysis. Finally, for metrics on the lower triangle, the resolution level is larger than the size of the selected traffic subset. Such metrics are rarely used today, and are thus of less interest to our study.

Note that the full design space is not always available. This is the case, for example, when working with data from stub networks (e.g., a campus network) where the maximum available subset size equals the IP address range assigned to the studied network. For a /16 campus network the subset size may range from 16 to 32, i.e., subset sizes larger than 16 bit are not available. Nevertheless, to keep the subsequent discussion general we assume the whole design space is available.

To further illustrate the granularity design space, we give example metrics for five selected cells of the matrix:

Cell 1 (00,00): Select all traffic and set the resolution to the minimum. An example metric is the well-known volume over all traffic.

Cell 2 (00,32): Select all traffic and set the resolution to the maximum. Examples are the volume per IP address or the number of unique IP addresses in all traffic.

Cell 3 (32,32): Select traffic to/from one IP address and set the resolution to the maximum. Metrics falling in this category are, e.g., the number of unique ports per IP address, or the number of unique IP addresses that the host under observation sends traffic to.

Cell 4 (24,32): Select traffic to/from one /24 network and set the resolution to the maximum. Examples for this case are the flow count per IP address, or the unique number of IP addresses that send traffic in the monitored /24 network.

Cell 5 (00,16): Select all traffic and set the resolution to /16 networks. An example metric is the volume per /16 networks in all traffic.

3.2 How Anonymization Diminishes the Design Space

In the following we show how the most commonly employed IP address anonymization techniques diminish the granularity design space. The available subset of the design space for different anonymization techniques is illustrated in Figure 3.2, where filled squares mark the possible combinations of subset size and resolution for each anonymization technique. Note that for permutation-based approaches all fields with a subset size smaller than 00 are marked with a different color. We did this to signify that subsets of smaller sizes may be distinguished, but not identified, since the mapping from real to anonymized IP addresses is usually not known. Hence, a subset of interest has to be identified by different means, e.g., by selecting subnets with particular traffic characteristics. For the subsequent analysis, however, we make no distinction between the two cases.

Blackmarking (BM) replaces all IP addresses in a trace with the same value. As a result, all information about individual IP addresses or subnets is lost and only metrics with the lowest resolution and the largest subset size, e.g., the volume over all traffic, can be computed. This corresponds to a single cell in the design space matrix, the lower left corner of the matrix.

Truncation (TR$\{t\}$) replaces the t least significant bits of an IP address with 0. With respect to the design space, this means only metrics with a resolution and subset size of [00, 32 - t] can be computed when truncation is used. The number of available granularities decreases with t, the number of truncated bits, as illustrated in Figure 3.2.

Random permutation (RP) translates IP addresses using a random permutation that does not preserve the prefix structure. Since RP creates a one-to-one mapping, the number of distinct IP addresses is the same. Hence, when RP is used for anonymizing a trace, metrics that can be computed on it may only feature the highest and lowest resolution values, as well as largest and smallest subset sizes. Note that these correspond to the four corners of the design space matrix.

Partial prefix-preserving permutation (PPP$\{p\}$) preserves the prefix structure in a trace at one specific prefix length p, and at the level of IP addresses. Consequently, this technique retains all granularities that have a resolution and subset size of either 00, p, or 32 (see Figure 3.2).

Prefix-preserving permutation (PP) is actually the best anonymization technique with respect to utility since it preserves the full design space. We will use it in our measurement study as a reference to a perfect anonymization scheme (with respect to utility).

3.3 Quantification of Data Utility

In this section, we describe our methodology for studying the impact of anonymization on statistical anomaly detection. We introduce the data set used in this study, and describe the methodology for classifying it. We further present the Kalman filter that is used as detection algorithm.

3.3.1 Measurement Data

The data used in this study were captured from the border routers of the SWITCH network. We analyzed a three-week period (from August 19th to September 10th 2007). This data set contains a variety of anomalies with diverse characteristics. In total, 43.2 billion flows covering a volume of 713 Terabytes of traffic were analyzed. In contrast to previous work, this study is based on un-sampled and non-anonymized flow data. Such data sets are difficult to obtain (at least over longer observation periods), but mandatory if bias and distortion in the results are to be avoided.

3.3.2 Ground Truth

The first step of any measurement study on anomaly detection is the establishment of ground truth for the available traces. Unfortunately, obtaining ground truth for an unclassified data set is still a large challenge and involves a lot of manual inspection. In the following, we describe our methodology for labeling the data set.

Visual inspection of metric time series: We computed the time series for five well-known metrics over an entire three weeks period at 15-minute intervals, resulting in 2016 data points per metric. As metrics, we selected byte, packet, and flow counts, unique IP address counts, and the Shannon entropy[1] of flows per IP address. Moreover, we distinguished incoming and outgoing traffic, as well as TCP and UDP traffic adopting what is common practice in the anomaly detection community. Finally, we visually inspected all these time series for unusual events.

[1] $H(X) = -\sum_{i=1}^{n} P(x_i) \cdot log_2\big(P(x_i)\big)$

	Volume	(D)DoS	Scan	Fluct.	Unknown	Total
TCP	75	32	542	24	19	689
UDP	64	14	4	239	28	339

Table 3.1: *Ground truth: Number of anomalous intervals per anomaly type and total for UDP/TCP.*

Analysis of raw NetFlow traces: For all intervals that could not be classified by time series inspection with high confidence, we did further analysis on the raw NetFlow traces. For this purpose, we used nfdump [102], a tool developed by SWITCH for forensic analysis to collect more information about suspicious events, e.g., which hosts and ports are affected.

Assigning ground truth to each interval: If at least *one* of the analyzed metric time series exposed an unusual event in some interval, we classified that interval as anomalous. Note here that most events were visible across multiple metric time series.

Identifying the anomaly type: Having classified all intervals as normal or anomalous, we went one step further and assigned the anomalous events to different types. Since a commonly agreed methodology for classifying known anomalies is not yet established, we define and distinguish the following types of events:

Volume: Volume anomalies are events that cause a sharp increase or decrease in the volume-based metrics, but do not affect the feature-based metrics. In our trace, we found two large loss events and several alpha flows (i.e., high-volume flows).

(D)DoS: Denial of Service attacks cause a concentration of the flows on one or few target IP addresses and hence a drop in the destination IP address entropy. If, on top, the attack is distributed, we will additionally see a spike in the source IP address counts and entropy metrics. If the attack is large in terms of flows or even packets, in addition, it will cause a spike in volume-based metrics.

Scan: Scans provoke a sharp increase in the destination IP address counts and entropy. If the attack sources are distributed, we also see a spike in the source IP counts.

Network Fluctuation: Events that cause an increase or decrease in the IP counts at lower resolutions but are not significant in the IP address counts at the highest resolution, fall into this class. Examples of such anomalies are ingress shifts and route flaps, but also massively distributed coordinated events that involve only a small number of IP addresses (e.g., botnet activity or stealth scans).

Unknown: Despite the classification effort that was made, some events remained unclassified. All unclassified events fall into this class.

Table 3.1 summarizes the identified events in our three-week long trace for UDP and TCP traffic. Note that we counted the number of anomalous intervals not the number of anomalies. Therewith the number of anomalous intervals can be quite large for anomalies that persist over several hours or even days. For example, a large part of the 542 TCP-scanning intervals belongs to a single long-lasting event. Likewise, most of the 239 intervals classified as *network fluctuation* belong to one single event that reappeared every 2 hours over several days, but lasted each time only for few intervals.

3.3.3 Anomaly Detection with the Kalman Filter

From the list of available statistical anomaly detection methods, we selected the Kalman filter since its excellent performance for anomaly detection has been shown in [134]. The Kalman filter is an efficient recursive filter that estimates the state of a dynamic system from a series of incomplete and noisy measurements. It models normal traffic as a "measurement-corrected" AR(1) process plus zero-mean Gaussian noise. The difference between this model and the actual measured time series, the so-called residual, is used for detection (see Figure 3.3 for an illustration). An alarm is raised by the detector if the residual exceeds some threshold.

We calculated a total of 60 metrics (see next Section) on the three-weeks data set, and applied the Kalman filter *separately* to each of those. This results in a $\{60 \text{ x } 2016\}$ matrix of residual values, one for each metric and interval. An anomaly is detected if at least one of the 60 residual values for an interval exceeds a threshold. We assess the performance of the Kalman filter with the help of ROC curves [61]. ROC curves plot the rate of false positives against the rate of true positives for a range of thresholds (see Figure 3.5 for an example). As thresholds, we use multiples of the a posteriori estimation for the standard deviation (s) of the considered metric. The thresholds range from $0.2s$ (top right corner) to $2.4s$ (bottom left corner). Typically, an interval exceeding the standard deviation of the noise process is considered unusual.

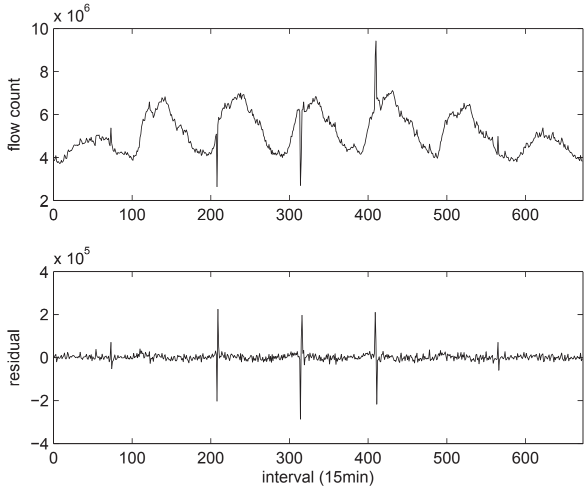


Figure 3.3: *Time series and corresponding residual signal from the Kalman filter.*

In general, the higher the true positive rate at a particular false positive rate, the better the performance of the detector. Hence, the curve of an optimal detector goes through the top left corner whereas a curve close to the diagonal represents random guessing.

For both UDP (left) and TCP (right) traffic, Figure 3.5 shows one ROC curve per anomaly class, and one curve for the overall detection capabilities. We restrict our analysis to a specific type of anomaly as follows: we exclude all intervals that have been manually classified as anomalous, but of a different type, and used these shortened time series as input for the detector. We see from this figure that the Kalman filter generally works well. For UDP, we obtain very high true positive rates at a small false negative rate for all classes of anomalies. Detection results for TCP traffic are slightly worse. This is however expected since TCP has a larger traffic share than UDP and is also more volatile compared to UDP traffic.

When examining the traces in detail, we found that false positives are often due to fast increases or decreases in the normal daily traffic cycle, which are misinterpreted by the Kalman filter as anomalous events. Another source for false positives are temporary increases in the volatility of a metric. This type of false positives can be avoided by recalibrating the Kalman filter each

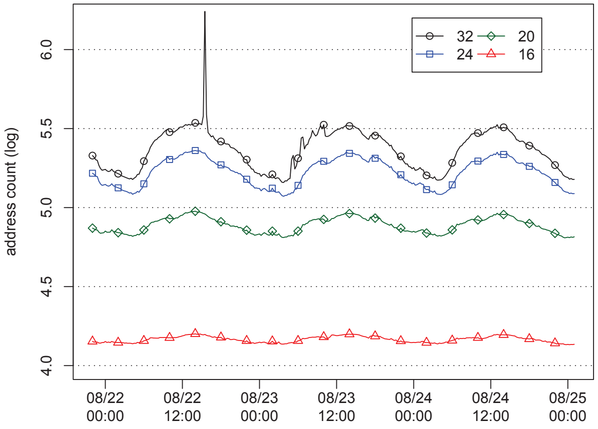


Figure 3.4: *Illustration of the loss of resolution effect. The unique address count at resolutions 32, 24, 20 and 16 is shown.*

time the volatility changes. We also observed that the Kalman filter tends to miss anomalies that increase or decrease more slowly, and take multiple intervals to grow to their full strength. For some of those, however, the Kalman filter detects the end of the anomaly when the traffic suddenly falls or rises to its previous level. There is one more thing to point out in Figure 3.5: The false positive rate at a specific threshold is practically the same for all classes of anomalies (i.e., all markers representing the same threshold are more or less vertically aligned). The reason therefore is that the false positive rate depends only on the normal traffic, which is the same for all cases, and not on the studied anomaly.

3.3.4 Computing the Utility of Anonymized Data

Basically, we use the same methodology for the non-anonymized case as for the anonymized traffic. The difference is, however, that we run the Kalman filter only on the subset of the 60 metrics that is available for the anonymization technique under study.

The 60 studied metrics are different variants of three volume-based metrics (vbm) (byte, packet, and flow counts) and two feature-based metrics (fbm) (the unique IP address count, and the Shannon entropy of flows per IP address). We distinguished TCP and UDP traffic as well as incoming and

	vbm{00}	fbm{16}	fbm{24}	fbm{32}
PP	✓	✓	✓	✓
PPP(16)	✓	✓		✓
RP	✓			✓
TR(08)	✓	✓	✓	
TR(16)	✓	✓		
TR(32)	✓			

Table 3.2: *Metrics available with different anonymization techniques (vbm = volume-based metrics, fbm = feature-based metrics).*

outgoing traffic. Moreover, we used a subset size of 0 for all metrics, i.e., we computed our metrics over all available traffic. Since we were interested in exploring how the restriction of available resolutions affects anomaly detection, we computed the metrics at four representative resolution levels of {00, 16, 24, 32} bits[2]. Here, we made a distinction between volume- and feature-based metrics. We computed volume-based metrics only at the lowest resolution of all traffic. This is because the computation of volume metrics at higher resolutions (e.g., the volume per IP) results in one time series per entity, and a clustering mechanism would be required to summarize them into one metric. The impact of anonymization on clustering algorithms, however, is not subject of this study. Feature-based metrics, on the other hand, were computed at a resolution of {16, 24, 32} bits. The lowest resolution was not used for feature-based metrics since it results always in a value of one (e.g., there is only one unique /0 prefix in the trace). Therewith, we obtain a total of 60 detection metrics.[3]

The resolutions available with each anonymization scheme are given in Table 3.2. Also refer to Figure 3.2 which illustrates the subset of the design space available with each technique. The volume-based metrics computed at a resolution of 00 bits (vbm{00}) are available for all anonymization techniques. Feature-based metrics computed at a resolution of 16 bits are available with all techniques that retain this resolution, i.e., PP, PPP(16), TR(08), and TR(16). Likewise, feature-based metrics at a resolution of 24 bits are available with all anonymization techniques that retain the resolution of 24 bits, i.e., PP, and TR(08). Finally, feature-based metrics computed at a resolu-

[2] A resolution of 8 bits is too low for our data set.

[3] $(3[vbm] + (2[fbm] \times 2[src/dst] \times 3[res])) \times 2[in/out] \times 2[udp/tcp] = 60$.

tion of 32 bits are available with all permutation-based techniques since these retain the notion of individual IP addresses.

To assess the utility, we compared the ROC curves obtained when using the restricted set of metrics available with each anonymization technique. Further, we reduced the complex ROC curves to a single utility value by computing the area under the curve (AUC) [20]. To obtain the AUC from the empirical ROC curves, we fitted a piecewise cubic Hermite interpolating polynomial to the data, and approximated the area under the curve numerically. In the next section, we describe and discuss the results obtained with the methodology described above.

3.4 Measurement Results

We commence this section with an illustrative example for the *loss of resolution* effect, i.e., we examine how the restriction of available resolutions through anonymization impacts the detectability of anomalies. In Figure 3.4, we plot the count of unique source addresses in all incoming TCP traffic at different resolutions of IP addresses (32), /24 networks (24), /20 networks (20), and /16 networks (16). In the curve for the highest resolution value, we see a large peak corresponding to 1.2 million additional IP addresses launching a denial of service attack. Interestingly, the peak completely disappears at a resolution of 24 and lower. Detailed analysis of the data for /24 networks reveals a small increase by about 5'000 additional networks in the same interval. But this change is not significant within the total of all /24 prefixes and thus not visible in the curve. From this observation, we conclude that the attack sources remain in a few /24 networks. As seen in the figure, this example anomaly disappears at lower resolutions, and thus, it will be hard or impossible to detect in data anonymized with truncation. In the following, we systematically assess the overall utility of the different anonymization techniques with the help of the Kalman filter detector applied to the whole three-week long data set.

3.4.1 ROC Curves for Anonymized Data

To assess the utility of the different anonymization schemes, we study the impact of anonymization separately for each type of anomaly. This is necessary since each type of anomaly is exposed at characteristic traffic granularities, and thus, is impacted differently by the restriction of the granularity

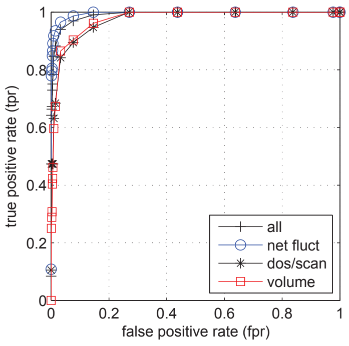


Figure 3.5: *ROC curves for different types of anomalies in UDP (left) and TCP traffic (right).*

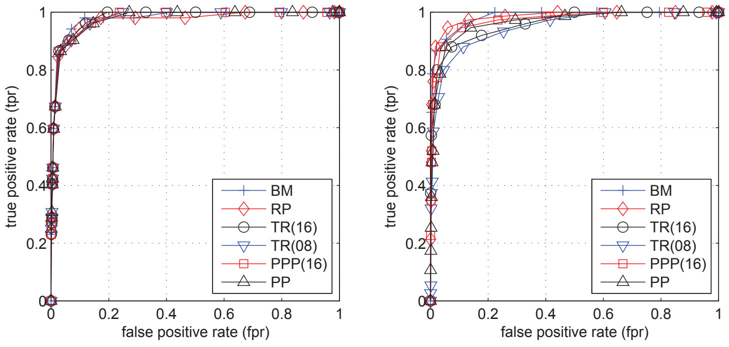


Figure 3.6: *Volume anomalies in anonymized traffic, UDP (left), TCP (right).*

design space through anonymization. To give an example: Alpha flows are mainly visible in byte and packet counts computed at the lowest resolution, while scans are primarily visible in feature-based metrics at higher resolutions. Hence, studying all anomalies together will thus not lead to any conclusive results. As a result, we distinguish the following anomalies for our evaluation:

Volume Anomalies, such as outage events and alpha flows, are mainly exposed by volume-based metrics. Since volume-based metrics at the lowest resolution are available with all anonymization schemes (see Table 3.2), we

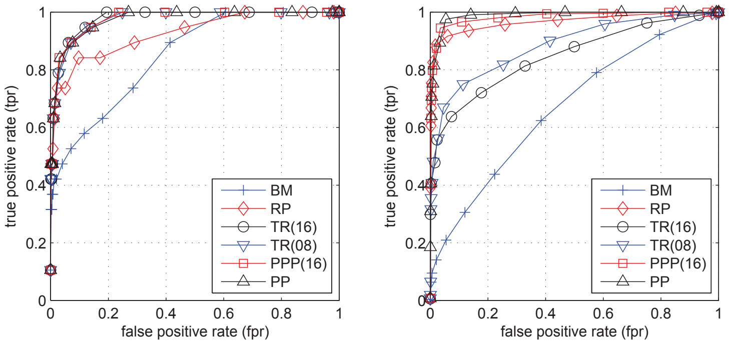


Figure 3.7: *Scanning and denial of service anomalies in anonymized traffic, UDP (left), TCP (right).*

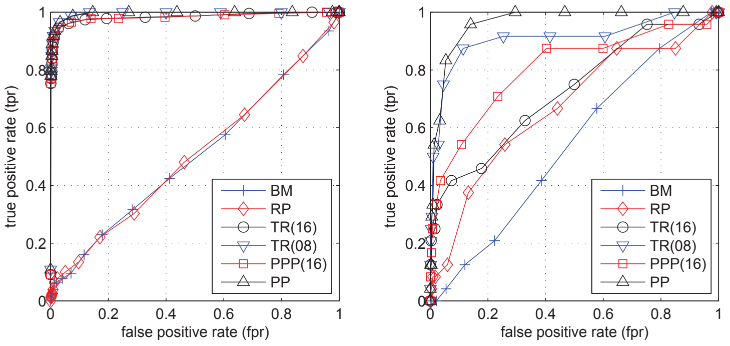


Figure 3.8: *Network fluctuations in anonymized traffic, UDP (left), TCP (right).*

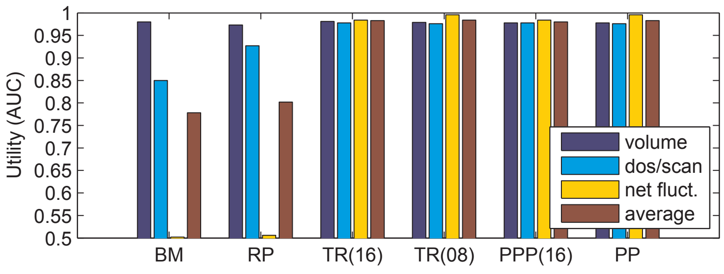
expect that anonymization does not have a large impact on the detection of volume anomalies. Indeed, the measurement results presented in Figure 3.6 clearly confirm this expectation. Anonymization does not alter the utility of data when one is solely interested in detection of volume anomalies. When examining the plot for TCP traffic more closely, we observe that blackmarking and random permutation perform slightly better than the other schemes. We conclude that using fewer metrics might even be beneficial as it results in fewer false positives.

Scanning and denial of service anomalies are both mainly visible in feature-based metrics. Measurement results for this class of anomalies are presented in Figure 3.7. The curves for UDP and TCP confirm that blackmarking performs worst, whereas prefix-preserving permutation (which does not restrict the granularity design space at all) has the best performance. To give an example: at a false positive rate of 0.02, the detection rate for blackmarking is reduced by 50% for UDP, and even more for TCP traffic. Surprisingly, the ranking for truncation and random permutation is not the same for UDP and TCP traffic. For TCP traffic, random permutation outperforms truncation, while for UDP traffic the opposite applies. We think this is due to structural differences between, normal and anomalous, UDP and TCP traffic. We verified with the data that TCP scans and DoS attacks are mainly visible at the resolution of individual IP addresses, which are preserved by random permutation but not truncation. On the contrary, UDP scans and DoS attacks are visible at high and low resolutions, but metrics at lower resolutions have fewer false positives. Consequently, truncation outperforms random permutation for detecting UDP scans and DoS attacks.

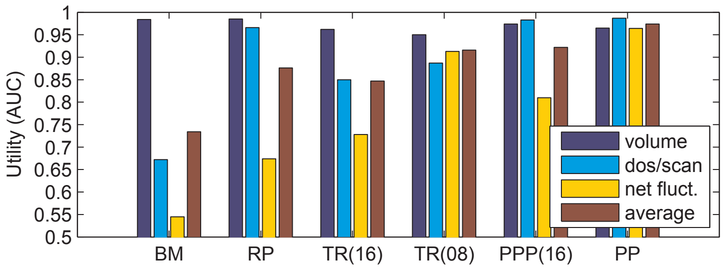
Network fluctuations are mainly visible in feature-based metrics at lower resolutions. The ROC curves for UDP and TCP traffic presented in Figure 3.8 show that detection of network fluctuations is almost impossible when either blackmarking or random permutation is used. Truncation of 8 bits, on the other hand, does not result in a severe performance degradation, neither for TCP nor UDP traffic. UDP and TCP differentiate with respect to the performance of 16-bit truncation and PPP. This might, however, be a particularity of our data set: Most of the 239 network fluctuation anomalies in the UDP traffic are mainly visible at a resolution of 16 bits, whereas TCP network fluctuations are rather visible at a resolution of 8 bits.

3.4.2 Utility of Anonymized Traces for Anomaly Detection

We will now summarize the detailed results from subsection 3.4.1 using the area under the curve (AUC) as a measure for the utility of an anonymized data set. An AUC value of 1 means that the detector achieves perfect accuracy, whereas a detector with AUC=0.5 is as bad as random guessing. For each anonymization technique we plot the AUC value (y-axis) in Figure 3.9(a) (UDP traffic) and in Figure 3.9(b) (TCP traffic). We show one curve for each of the three anomaly classes (volume, scan/DoS, and network fluctuations), as well as the average over all classes. Note that this average utility corresponds



(a) UDP traffic



(b) TCP traffic

Figure 3.9: *Utility for anomaly detection.*

to a data set that contains the same ratio of volume, scan/DoS, and network fluctuation anomalies.

Figure 3.9(a) and Figure 3.9(b) clearly show that the detectability of volume anomalies is not impacted by anonymization. Furthermore, it confirms our intuition that prefix-preserving anonymization has the best utility with respect to backbone anomaly detection in general, and that blackmarking has the worst utility for all classes except volume anomalies. The utility of random permutation, truncation, and partial prefix-preserving permutation largely depends on the type of anomaly in question. Partial prefix-preserving permutation performs almost as well as prefix-preserving permutation; it has a lower utility only for network fluctuation anomalies in TCP traffic. Random permutation has a high utility for the detection of large-scale scans and denial of service attacks, but a low utility for detecting network fluctuations in UDP as well as TCP traffic. Truncation performs very well for UDP traffic in our measurements, but has a lower utility for detecting scans, denial of service attacks, and network fluctuation anomalies in TCP traffic. Moreover, the utility for these anomalies decreases when more bits are truncated.

Naturally, the overall utility of an anonymization technique applied to a given data set depends on the mix of anomalies within the trace. For our data set, the overall utility for detecting TCP anomalies is clearly dominated by the 542 scanning intervals. Likewise, the overall utility for UDP traffic is to a large extent dominated by the 239 intervals with network fluctuations. Hence, for this anomaly mix, PP and PPP offer the highest utility, truncation lies in the middle, and random permutation and blackmarking result in the lowest utility. We summarize the results for our data set as follows:

- Blackmarking decreases the utility for detecting anomalies in UDP and TCP traffic dramatically.
- Random permutation performs only slightly better than blackmarking for UDP. However, For TCP, it enables much better detection of scan/-DoS anomalies.
- Truncation of 8 or 16 bit decreases the utility for detecting anomalies in TCP traffic by roughly 10 percent, while performing well for UDP traffic.
- (Partial) prefix-preserving permutation has no significant negative impact on the utility for detecting anomalies in UDP and TCP traffic.

To derive more general conclusions about the utility of different anonymization schemes it would be helpful to study the anomaly mix in available flow traces. If this mix converges, at least for traces recorded in the same time period, a general conclusion about the utility can be derived directly from our results. Moreover, we expect that the results for volume, scan/DoS, and network fluctuation anomalies hold also for other traces. Verifying this assumption requires that we apply our methodology to further un-anonymized data sets from different networks. Unfortunately, such traces are not currently available in abundance.

3.5 Implicit Traffic Aggregation

In section 3.4, we investigated how anomaly detection results are falsified when data are not available in appropriate resolutions. Another aspect of anonymization that is worth studying is the restriction along the subset size dimension, causing an implicit aggregation of traffic.

Let us illustrate this with an example. Consider the case where traffic from a single host (i.e., subset size 32) is to be investigated in presence of 4 bits truncation. The best one can do under these circumstances is investigation of the /28 network that contains the host, since the individual host can no longer be distinguished from other hosts in the same network. As a consequence, the analyzed traffic is a mixture of traffic from the target host and traffic from other hosts in neighboring subnets. In accordance with loss of resolution, we refer to this effect as the *loss of focus effect*.

The impact of the loss of focus effect is, of course, highly dependent on the distribution of traffic in the studied network. *It is extremely difficult to predict the implications for a particular case.* In the worst case, even truncation of a single bit can be fatal. That is, traffic characteristics of the target host could get lost completely in another host's traffic. In the best case, where the truncated subnet is dedicated to a single host, no traffic is aggregated. In that case, the loss of focus effect is simply reduced to a loss of resolution effect.

We can, however, estimate the average severity of truncation-induced traffic aggregation by analyzing the count of additional, non-belonging, flows for individual hosts. In particular, we count the flows of all webservers belonging to a single /16 network in our un-anonymized traces. Then, we apply truncation to the traces and count the number of flows to the subnets containing the webservers.

When truncating a single bit, the measured traffic is not increased for more than 50% of the observed webservers (170 in total). Only for around 10% of the webservers, the measurement shows traffic increase of 100% or more. However, if more bits are truncated the situation gets worse. The ratio of unaffected servers drops to 20% for 2 bits, 5% for 4 bits, and even 0% for 8 bits. Similarly, the ratio of server measurements experiencing at least a doubling of traffic goes up to 25% for 2 bits, 55% for 4 bits and 89% for 8 bits.

We conclude that accurate detection of small-scale anomalies[4] is very difficult, if not impossible, when the desired subset size is not supported by the restricted granularity design space. The probability that host characteristics are lost in aggregated traffic is simply too high. Only anomalies of sufficient scale have a chance to be visible in aggregated traffic at larger subset sizes.

[4]With small-scale anomalies we denote anomalies affecting only a single host or a small subnet.

In addition, false positives are introduced by the aggregation with traffic from other hosts.

3.6 Summary

In this chapter, we have answered the question of how anonymization techniques impact statistical anomaly detection. We introduced the detection granularity design space as an important tool to illustrate this impact. We have shown how the design space, spanned by subset size and resolution, is reduced by the most common IP address anonymization techniques. Finally, we analyzed the utility of anonymized traces for the detection of large-scale anomalies, as well as the impact on traffic characteristics of individual hosts with the aid of backbone traffic traces gathered over a three-weeks period.

In general, our results indicate that the restriction of the granularity design space through anonymization hinders anomaly detection. With respect to the individual techniques, we have found that prefix-preserving permutation offers the best utility, and blackmarking performs the worst. Moreover, we have shown that the performance of random permutation, partial-prefix preserving permutation, and truncation strongly depends on the type of anomaly that is studied, as well as the underlying transport protocol. Our results indicate that the detection of volume anomalies, such as outages or alpha flows, is not impacted by anonymization at all. The utility for detecting scans and denial of service attacks degrades when truncation is applied. Detection of network fluctuations, on the other hand, is impacted principally by blackmarking and random permutation.

Thus, if one is interested in a particular type of anomaly, anonymization could be tuned in a way such that the results are less impaired. In addition, we have shown that the anonymization-induced loss of focus, i.e., when the desired subset size is not available, in most cases completely distorts the traffic characteristics of individual hosts.

Chapter 4

Identifying Hosts in Anonymized Data

You have zero privacy anyway.
Get over it.

Scott McNealy, 1999

After having studied the remaining *utility* of anonymized data in the last chapter, we now move forward and analyze the *host privacy* in anonymized data. As defined in chapter 2, the term host privacy refers to all privacy issues associated with IP addresses, including the generation of user profiles and the leakage of server and network statistics. We start out with a classification of attackers and then perform experiments with live traffic data. Our results show that privacy protection of permutation-based techniques is weaker than anticipated. We discuss our results and implications for the real-world application of anonymization in detail.

4.1 Real-World Attacker Models

We consider two general types of attacks against network data anonymization, inspection and injection.

Inspection attacks attempt to break anonymization using only information from the trace and information available from observation of the network and from other relevant sources (e.g., DNS and WHOIS databases) *after* the trace was collected. Note that inspection attacks are not completely passive; they may employ active network measurement techniques, e.g., scanning networks suspected to be the measured network in a given trace in order to locate services.

Inspection attacks are usually not *privileged*, that is, the attacker does not have any information not available to anyone else outside the operator of the measured network and data publisher. They are, however, possible against already-published data. Inspection attacks can be thought of as a superclass of all the attack classes enumerated in [79] except Data Injection.

Injection attacks exploit additional foreknowledge of traffic which will appear in a subsequently published trace, generally by generating traffic in the measured network with known characteristics. The attacker here is privileged, having information about the patterns injected into the trace that no other external observer does. This is analogous to a known-plaintext attack against a cryptosystem: by causing known traffic to be captured in the trace for subsequent anonymization and publication, the attacker has knowledge of some raw data within the trace. Note that while the operator of the measured network does have information about the injected patterns by virtue of having access to the raw data, they may not realize what portion of the traffic contains the injected patterns, or even indeed that patterns have been injected. Injection attacks do, of course, require knowledge about the network location and time period during which a trace will be collected and published *before the fact*, and the ability to cause network traffic to be generated on the measured network. Injection attacks are analogous to the Data Injection attack class in [79].

One-time publication of traces without prior announcement are more likely to be attacked by inspection alone, since successful injection would require either some measure of luck or a pervasive traffic injection infrastructure. But if traces are published on a regular basis from a given network, or publication is announced beforehand as in a coordinated collection activity such as [78], then potential injection attackers know the schedule of collection, and the publisher must assume that injection attacks are underway.

Note that long-term injection attacks do not pose significant storage requirements for the attacker. Deterministic or pseudorandom traffic sequences can be described completely by the generation algorithm and its parameters (e.g., a seed for a pseudorandom number generation). When stored with the start time of each injection activity, this enables the attacker to easily synchronize the trace and generation timeline. This information can be leveraged to reconstruct timing information removed from the traces, such as the date of the capture, or inter-flow timing as in timestamp enumeration.

4.2 Traffic Injection Experiments

Traffic injection attacks establish covert channels and are therefore inherently difficult to detect. Almost any field in traffic data can be used to carry some bits of hidden information along with real data. An example of how timing fields of IP packets can be used as covert channels is given in [33].

The signal injected by an attacker can vary in terms of strength and duration. Signal strength refers to the number of bits used to encode the signal and how different the injected signal is from normal traffic. Of course, even a weak signal can be detected, given enough time. In this section, we empirically explore the success rate of these attacks by injecting artificial packet sequences into a network. We call the footprint these packets leave in flow traces *patterns*. The specific questions we try to answer are: Can sequences of packets injected in the attacker's home network, routed across the Internet to a target network, captured in the destination's backbone network, exported in NetFlow format together with all the backbone's traffic, and then after anonymization still be recognized by the attacker? If yes, what is the minimum pattern complexity (number of packets, data volume, specific ports) required? Could anonymization of secondary flow fields lower the pattern recognition rate or even mitigate these injection attacks? Are certain host classes (e.g., webservers) more easily identified by injection attacks than others?

Figure 4.1 gives an overview of our experimental setup. Using a traffic generation and injection framework developed for this study [124], we injected fingerprinted packets into the SWITCH network. We then collected these packets with an existing collection infrastructure, which produces unsampled and non-anonymized NetFlow records from the entire SWITCH border. We sent packets to 30 target hosts during working hours, as up to 140 million flows per hour were collected. Both captured and injected flows were anonymized according to a variety of policies using software integrated into this collection infrastructure, which we developed for this study. Then, given this anonymized traffic, we attempted to de-anonymize the target IP addresses by recovering the fingerprinted flows from the set of all SWITCH flows. As

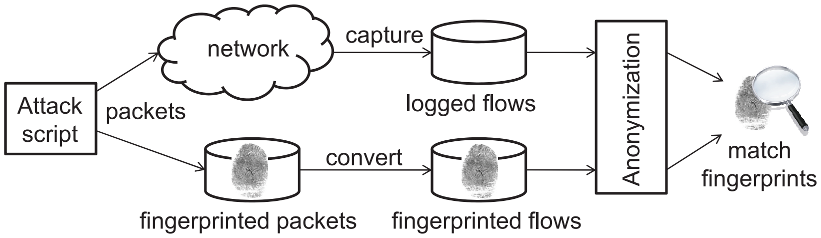


Figure 4.1: *Setup of the injection attacks.*

we controlled the set of injected traffic and target addresses, we have ground truth to compare to. We emphasize that we only attacked our own anonymization schemes and there was no anonymization applied by the data owner. In general, breaking the data owner's anonymization scheme is not considered acceptable use of data [4].

4.2.1 Pattern Complexity

In this section we look at the pattern complexity necessary to identify fingerprints in anonymized traffic. A simple pattern that can be used to identify hosts is network scanning. For instance, Pang et al. [108] explicitly remove scans from traces to mitigate injection attacks. Others look for existing scans in traces but refrain from performing own scans in order not to be detected by administrators [80]. The patterns we devise aim at being stealthier than scans and thus harder to detect. Our patterns P_1 to P_5 listed in Table 4.1 contain an increasing amount of packets, randomness, and variation. We speculate that more complex patterns, such as P_5, will deviate more from "normal" traffic patterns and will thus be easier to identify later on. The complete design space for these patterns is large and will be discussed in more detail in section 4.3.

We injected each of these patterns to 30 target hosts, selected randomly from webservers, public student workstations and darkspace addresses. Each of the 150 pattern-host pairs was injected in a distinct 5-minute window.

Because the attacker relies on secondary flow fields like ports, timing or packet count/size to identify his patterns, we then blurred these patterns by applying anonymization policies A_1 to A_6 (see Table 4.2) to the captured and fingerprinted traces. From A_1 to A_6, anonymization increases in strength as data utility decreases.

	Packets	Src port	Dst port	Interval [ms]	Packet sizes
P_1	1	Fixed	80	-	160
P_2	5	R(65k)	R(65k)	200	256
P_3	10	Fixed	80	200	480 [+32]
P_4	10	R(65k)	R(65k)	200	832 [+32]
P_5	50	R(65k)	R(65k)	150+R(300)	1208 [+R(8)]

Table 4.1: *Injected Patterns. Values in square brackets denote the field evolution between packets. R(x): random number between* 1 *and x. Interval: time between two packets.*

	IP address	Ports	Timestamp [s]	Packet count	Byte count
A_1	Perm.	-	-	-	-
A_2	Perm.	-	-	Offset(5)	Offset(50)
A_3	Perm.	Bucket(8)	Offset(30)	-	-
A_4	Perm.	Bucket(2)	Offset(60)	-	-
A_5	Perm.	Bucket(8)	Offset(30)	Offset(5)	Offset(50)
A_6	Perm.	Bucket(2)	Offset(120)	Offset(10)	Offset(200)

Table 4.2: *Anonymization Policies. Bucket(x): bucketized in x buckets, Offset(x): Added a uniform random offset between $-x$ and $+x$ (minimum resulting value being 1).*

For each pair of injected and captured flows we computed a difference score considering the flow fields used in the patterns. This compensates for potential imprecision as a result of retransmissions or measurement artifacts. For each injected flow we selected the captured flow with the smallest difference score. We considered a pattern correctly recovered, if the majority of flows belonging to the pattern were attributed to the correct target host.

The average number of recovered pattern-host pairs per anonymization policy is shown in Figure 4.2. We recovered all patterns, even those where P_1 consists of a single packet, from policies A_1 and A_2. Bucketing of ports and timestamp randomization in A_3 and A_4 reduce the success of pattern recovery, especially for the simpler patterns P_1 and P_2. However, patterns P_3 – P_5 are still very powerful even when ports are sorted into two buckets and timestamps are distorted by up to one minute. P_3 and P_5 even resist A_5 that anonymizes every field. Policy A_6, which severely distorts the data, did prevent the recovery of all tested patterns. However, it is obvious that one could

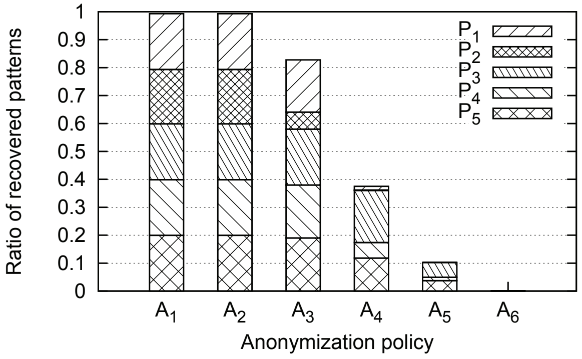


Figure 4.2: *Average ratio of recovered patterns for each anonymization policy. There are 150 patterns in total, 30 for each P_i.*

define more complex patterns with many packets and strong value variations to defeat even A_6.

The success of a pattern is greatly determined by its distinctness from the background traffic. Consider P_1 and P_3, both resulting in a single flow. The unique combination of 10 packets with 6,240 Bytes makes P_3 detectable even with A_5, while P_1 is lost with A_4. However, if no a priori knowledge about network traffic statistics is available, it is best to inject patterns with random values that result in multiple flows.

We found no difference in identifiability among host classes. This is due to the fact that a pattern has to stand out in all flows, not just in flows involving the target. Consequently, all hosts are vulnerable to injection attacks, whereas with inspection attacks only hosts with unique traffic characteristics (e.g., large webservers) are identifiable.

4.2.2 Flow Aggregation

In addition to IP address permutation, we also experimented with anonymizing flows by address masking, or truncating the least significant x bits of the IP addresses. The smallest identifiable entities in the aggregated traces were therefore subnets of size $/(32-x)$. Although this effectively prevents the de-anonymization of individual hosts, the injected patterns were equally recoverable, since truncation leaves secondary flow fields untouched. However, if flows are instead *aggregated*, by truncating addresses and/or bucketizing other flow key fields, then collapsing each flow value field by performing the natural aggregation functions for that value field (e.g., summation for counters, set union for flags), the number of distinct flow signatures is significantly reduced. Borrowing from *k*-anonymity [136], all aggregated flows representing fewer than *k* flows are then deleted or aggregated in one huge flow to preserve overall traffic statistics. This significantly complicates the recognition of injected patterns, because the patterns are merged with at least *k* other flows.

We simulated the effects of flow aggregation with various anonymization policies. For instance, with bilateral port classification, deletion of seconds in timestamps, truncation of 16 (8) bits from external (internal) addresses, only 12.5% of the flows remained unique, (i.e., were not aggregated with at least one other flow). As expected, we could no longer recognize injected patterns in the aggregated flows.

However, in addition to the fact that flow signature reduction requires severe anonymization policies applicable only to certain specific analysis purposes, an attacker can adapt to aggregation if he is capable of arbitrarily choosing source IP addresses (e.g., via a large scale botnet). In particular, to evade aggregation of injected patterns, the attacker simply needs to find a unique source subnet from which to inject traffic. If no other traffic originates from this subnet, he can easily inject patterns resulting in more than *k* flows. These patterns would then be recognizable in their aggregated form.

4.2.3 Pattern duration

Instead of developing more complex patterns, an attacker could also use very simple patterns but inject them over longer time periods. Here we assume an attacker that is able to inject *only one bit of information per time window*, due either to strong anonymization or a desire for stealth. In particular, our worst-case attacker decides in each time window to either send traffic to the target host (1) or not (0). If IP addresses are permuted 1-to-1, this allows the construction of a sequence of 0s and 1s for each src/dst IP address pair. For instance, 1011... describes a pair that appears in the first, third and fourth window but not in the second. Given enough time windows, there will eventually be only one pair matching the injected on/off pattern, namely the one with the attacker as source and the target as destination, allowing de-anonymization of the target host. This method is essentially independent of the anonymization

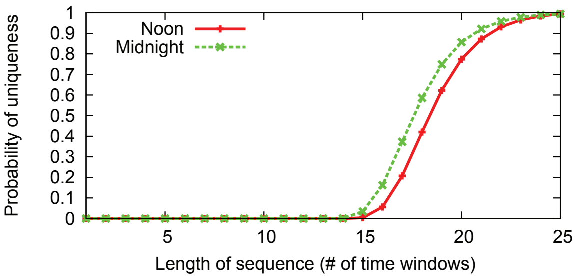


Figure 4.3: *Probability of a random traffic presence sequence of a given length to be unique in the trace.*

technique. In fact, secondary flow fields could all be overwritten with random values.

Figure 4.3 shows an analysis of the probability that a random on/off pattern of a certain length creates a unique sequence in a network trace collected at noon and at midnight of a CET (UTC+1) workday. At noon, 80% of the 20-bit sequences and 99% of the 25-bit sequences uniquely identify the fingerprinted src-dst pair. Even though the traffic volume is much higher at noon than at midnight, and therefore the number of possible candidates at noon (4.1 M per 5 min) is significantly higher than at night (2.5 M per 5 min), the average number of windows required to be unique is only increased by one. This is due to the fact that already an increase of one time window doubles the number of existing on/off patterns and therefore reduces the chance of a collision by a factor of 2. This exponential increase of the search space by a linear increase of the time allows scaling the attack to networks of arbitrary size. Depending on the anonymization of timestamps, this allows quick identification of targets. For instance, if A_5 is applied, a window size of 1 min. is sufficient to separate patterns and allows identification within 25 min. Of course, several of these sequences can be injected in parallel.

4.3 Injection Attack Space

In [130] the authors show that defending against injection attacks (with the goal of identifying alert submitters in collaborative intrusion detection) inevitably involves the suppression of rare or low-density attacks. This is basically what we did with A_3 to A_6. We suppressed small signals by generalizing

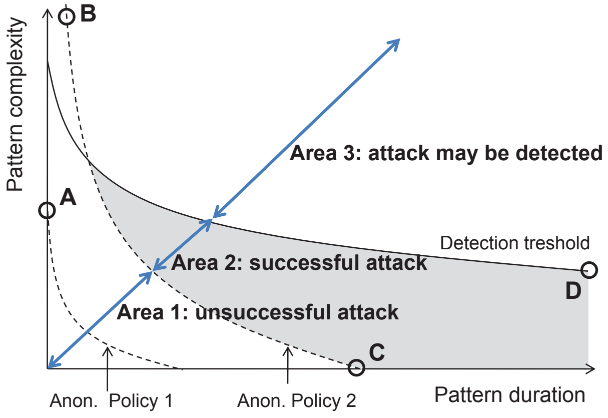


Figure 4.4: *Injection attack space.*

and randomizing the traces, hence success rate dropped for simple (i.e., low-volume) patterns. Packet sampling has a similar effect. After packet sampling, only events of a certain size are detectable, be it injected patterns or genuine traffic properties. Sampling does not effectively counter injection attacks, but it increases the required volume/time for attacks to work.

Technical defenses against injection attacks are limited by the decision problem of whether a given packet is legitimate or belongs to injected patterns. It gets even worse: A packet could be *both legitimate and injected*. For instance, an attacker could inject patterns using ordinary HTTP GET requests to de-anonymize a web server.

To discuss attacker and defender tradeoffs, we outline a conceptual injection attack space in Figure 4.4. The x-axis denotes the duration of the attacks and the y-axes the pattern complexity, i.e., how much is the injected pattern different from ordinary traffic. Towards the top right corner of the plot, the complexity and duration of injected traffic constantly increases until a significant part of the collected trace consists of injected traffic. The plot shows three areas:

Area 1: The injected pattern is not recovered by the attacker, either because the pattern does not stand out from normal traffic or because the deployed anonymization policy blurred the pattern. This is, for instance, the case for pattern P_2 with policy A_4 in Figure 4.2.

Area 2: The injected pattern is successfully recovered by the attacker, for instance P_4 with policy A_3. In addition, the network operator is unaware of this attack. This denotes a successful injection attack.

Area 3: The injected pattern is recovered by the attacker, but also the operator becomes aware of the attack. The operator might run intrusion detection systems capable of detecting abnormal traffic (e.g., packet sequences not consistent with application-layer protocol state). Also, if the injected patterns pass a certain volume, the attack becomes visible in network wide statistics. For instance, [80] refrain from active scanning in order not to be detected. Instead, they use existing scans in the trace.

Of course, the goal of the attacker is to operate in area 2 where he recovers the injected patterns but is not discovered himself. As this is a conceptual diagram, an empirical determination of the exact dimensions of this area would require an exhaustive study of varying pattern complexities and anonymization techniques. This is a suggested area for future work. However, we can identify two approaches to minimize the size of this area by restricting the number space from which patterns can be generated. Bogon filters [50] limit the size of the source IP address space, and have other security benefits as well. Aggressive ingress filtering based upon known services limits the size of the destination IP address space as well as the destination port space, but may interfere with applications using port negotiation or other NAT traversal techniques. Measurement of the impact they have on injection attacks is an area for future work.

Attack Space Asymmetry

Although the specific shape of the injection attack space is hard to determine and depends on the target network, some vertices can be fixed as shown in Figure 4.4, leading to an asymmetry between pattern complexity and duration.

First, with weak or no anonymization, it is possible to easily recover injected patterns, even for very short durations (point **A**). For instance, all patterns where recovered for A_2. Secondly, the stronger anonymization is, the harder it is to recover a pattern (see subsection 4.2.1). Thus, *anonymization can be used to raise the bar for attackers*. Anonymization can even be made strong enough to prevent pattern recognition by complexity alone. This corresponds to point **B**. Because the line of policy 2 never intersects the y-axis, it is not possible for the attacker to get into area 2 for small pattern durations. In our experiments, this corresponds to A_6, which makes the detection of all injected patterns impossible. Of course, rigorous anonymization policies have the downside of also reducing data utility significantly. Finally, it is always possible to de-anonymize hosts given enough time. This is even achieved by patterns with minimum complexity of 1 bit per time window, as shown in subsection 4.2.3. This corresponds to point **C**.

Points **B** and **C** together suggest that *it is significantly harder to defend against longer-duration attacks*. The attacker only needs constant storage to stealthily inject a sequence of random patterns over time. At the same time, the defender is looking for a needle in a (continuously growing) haystack. Therefore, the attack detection threshold line never touches the x-axis (point **D**). The conclusion is that *attackers can always reach area 2*, specifically by stretching patterns over time.

4.4 Summary

In this chapter, we have summarized the landscape of attacks against data anonymization, an important tool in enabling data sharing for research and cooperative network operations purposes. We have found that more research has been done on *inspection* attacks, in which only information from the trace and information available from observation of the network and from other relevant sources are used in breaking anonymization, than on *injection* attacks, which use privileged knowledge of traffic patterns within a trace as well.

From this realization, we performed experiments demonstrating the ease of recovery of injected traffic patterns and the difficulty to defend against injection attacks. These experiments led us to define an injection attack space which has an important asymmetry: it is relatively easy for an attacker to increase the chance of a successful pattern recovery by simply lengthening the pattern, but no easier for a defender to detect a longer pseudorandom injected pattern.

Chapter 5

The Privacy-Utility Tradeoff

Water which is too pure has no fish.

Ts'ai Ken T'an

In the last chapter, we demonstrated how an attacker capable of injecting traffic into a network can reverse any IP address anonymization technique based on permutation, including hashing, enumeration, random permutation, or (partial) prefix-preserving permutation. In fact, permutations do not actually remove information from the data, they just transform it. Thus, in worst case, it is always possible to reverse the permutation and gain access to the full information. From a risk analysis perspective, this is not acceptable.

Still looking for a convincing privacy-utility tradeoff, we go one step further in this chapter and apply anonymization that actually deletes information. In particular, we apply IP address truncation, which makes it impossible to identify individual hosts in the trace. We quantify both utility and privacy of the truncation technique and generate an *R-U map*, which plots the risk versus the utility in a single plot. This allows the identification of possible operating points that achieve *high utility* and *low risk of disclosure* at the same time.

5.1 Asymmetry of Internal and External Prefixes

The impact of anonymization highly depends on the underlying traffic characteristics. For the SWITCH network, we have observed that the distribution of SWITCH-internal[1] and external hosts is quite different. Pang et al. [108] make the same distinction and apply a weaker anonymization policy for external than for internal addresses. In their case, the perceived risk is lower for external addresses. In this Section we highlight the distributional differences in our network and explain why we think they give rise to a significant asymmetry between the internal and external domain.

As mentioned before, truncation causes a coarse-graining of network entities. Individual hosts are no longer distinguishable because truncation of x bits aggregates all IP addresses in the corresponding /(32-x) subnet. As a consequence, metrics based on IP addresses loose a significant amount of detail. Rather than counting IP addresses or computing the entropy of hosts, metrics are computed on prefixes of length 32-x. Figure 5.1 illustrates this effect with the example of unique address count. As more and more bits are truncated, the number of distinguishable prefixes is reduced. Prefixes are differentiated with respect to direction (source or destination) and address domain (internal or external). We differentiate the direction because this separates targets of scans that are mainly visible in internal destination addresses. The distinction between internal and external is done using the assigned prefix table of the AS. For instance, if a host within the SWITCH network sends a packet to `google.com`, the host will appear in the internal source addresses and `google.com` in external destinations. When `google.com` replies to the request it is added to external sources and the host to internal destinations. Note that a distinction between internal and external address space only makes sense in a stub AS (e.g. SWITCH) or an organization network. For transit ASes (e.g. Abilene), all addresses should be considered external.

Surprisingly, two groups of prefixes with different behavior can be clearly identified in Figure 5.1: those that represent external addresses and those of SWITCH-internal addresses. For increasing x, the number of external prefixes remains almost constant roughly up to point **(a)** around prefix length /24, and then changes to an exponentially decreasing regime for $x > 8$ bits. In contrast, the number of internal prefixes falls exponentially with x from the beginning

[1] Internal addresses are those belonging to the public IP address range announced by SWITCH (AS 559).

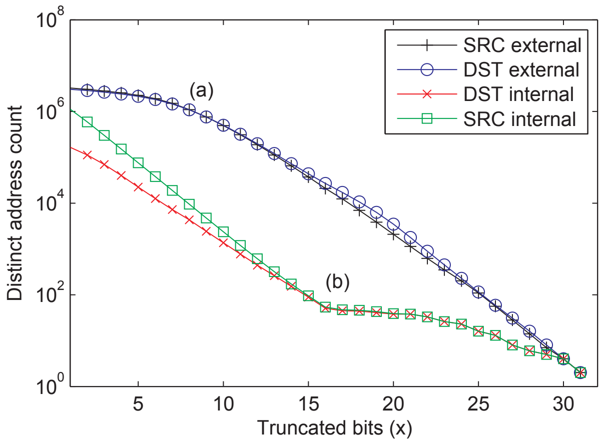


Figure 5.1: *Prefix structure analysis of internal and external addresses for one hour of regular traffic (x: truncated bits).*

and levels off at point **(b)** around 16 bits. The plateau to the left of **(a)** indicates that only few hosts in each external /24 subnet exchange traffic with the observed network. Since a corresponding plateau is missing for internal prefixes, we conclude that we see traffic from almost all hosts of internal /24 subnets. Naturally, the total number of internal /24 subnets is limited by the prefix table of SWITCH. The plateau after turning point **(b)** is due to the fact that the prefix table is dominated by 30 to 40 complete /16 subnets. Thus, the number of internal prefixes cannot grow exponentially from for $x = 32$ to $x = 16$ with prefix length as for external prefixes.

This observation leads to the question whether the truncation-induced reduction of risk and utility is asymmetric with respect to internal and external addresses. That is, we hypothesize that anomalies detectable in external address distributions are more resistant to truncation than those of internal distributions because the characteristics of external /32 prefixes are less impaired by truncation of the first 8 bits (see Figure 5.1). On the risk side, internal addresses aggregate faster with increasing x than external addresses, which makes it harder to identify individual internal hosts.

Note that the difference between internal source and destination prefixes is due to scanning activities that activate virtually all internal addresses (about 2 million) in the destination set. That is, even nonexistent hosts appear in the destination set, simply because a packet was sent to them. Thus, by multiplying the set of potential addresses, scanning could, at first glance, complicate the task of an attacker trying to identify a specific host in the data set. However, we believe it is quite simple to detect scanning activity and focus on the real hosts. Therefore, we only consider internal source addresses, representing the active hosts of the network.

Along with the general assessment of risk and utility in presence of truncation, we try to confirm or refute these asymmetry assumptions in the remainder of this chapter.

5.2 Utility Reduction

Using the methodology established in section 3.3, we now evaluate in more detail how truncation of IP address bits reduces the utility of data for detecting scans and (D)Dos events.

Truncation of x bits (e.g., 8 bits) replaces each IP address in a flow trace with its respective $/(32-x)$ (e.g., /24) address prefix. Hence, truncation mainly impacts metrics based on IP addresses, such as the unique address count, or the distribution of flows per address as captured by the Shannon entropy. Specifically, the number of unique IP address counts is replaced by the number of unique $/(32-x)$ networks in the trace; the distribution of flows per IP address is replaced by the distribution of flows per $/(32-x)$ network.

In Figure 5.2, we plot the AUC value for two metrics, address counts, and address entropy, for different truncation rates. Moreover, we distinguish metrics computed on internal and external addresses.

5.2.1 Counts vs. Entropy

When comparing the detection results for count and entropy metrics in truncated data, we see a clear difference between the two. While the detection rate on non-anonymized data is high, likewise for counts and entropy, the picture changes completely when truncation is used. The detector performance for counts decreases significantly with higher truncation rates. Even for a truncation rate as low as 4 bits, the detection (or true positive) rate for a given false positive rate of 0.05 is reduced by more than 50%, which makes detection practically impossible. For entropy metrics, on the other hand, the performance decreases much slower. Both, internal and external address entropy,

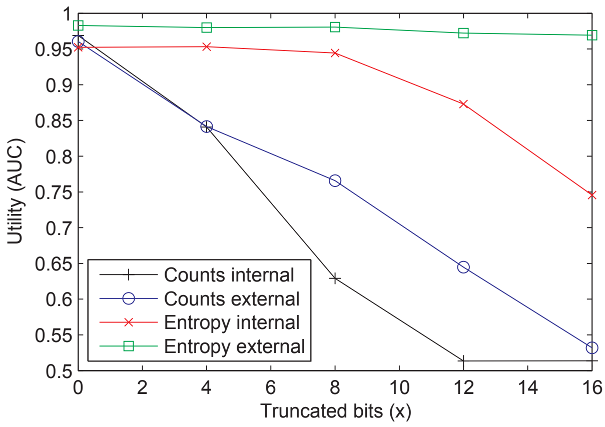


Figure 5.2: *Utility in terms of AUC for internal/external address counts and entropies.*

perform well even when the data are anonymized with truncation of as much as 8 bits.

How can we explain this remarkable difference? We have two possible explanations for the bad performance of counts: First, it is possible that the attacks are not well distributed with respect to network prefixes, e.g. /16 networks. A second reason might be that the background noise is higher for prefix counts than for IP address counts. Further analysis is required to check on these assumptions. The superior performance of entropy metrics on truncated data suggests that the distribution of flows per network prefix is similar to the distribution per IP address. The effect of truncation on entropy metrics is that the flows to all IP addresses within the same prefix are merged into a single value. Hence, the distribution within the prefix range gets lost, but the distribution of flows over all prefix ranges is retained.

We conclude from this observation that the increased complexity for computing entropy metrics, compared to simple address counts, is worth the effort, when working with anonymized data. Entropy has already been shown to work reasonably well on the Abilene traces that are anonymized with truncation of 11 bits [82, 118].

5.2.2 Internal vs. External Prefixes

We have introduced the prefix asymmetry observed in stub networks, like the SWITCH network we are studying, in section 5.1. We have shown that almost all available internal prefixes appear in normal traffic (exponential decay of unique prefixes with decreasing prefix length), while only a small part of all external prefixes is present in normal traffic (almost the same number of IP addresses and /24 networks). When comparing the plots for internal and external addresses for count and entropy metrics, we find that count metrics (upper row) do not show a significant difference for internal and external addresses, while for entropy metrics (lower row) there is a clear difference between the two. Entropy computed on external addresses is not affected at all, even when as much as 20 bits are truncated from the IP addresses. For internal addresses performance decreases significantly when more than 12 bits are truncated.

We believe that for count metrics the impact of truncation dominates any impact that the difference in internal and external addresses could have. For entropy metrics, on the other hand, the distribution is preserved much better for external addresses with increasing truncation rates. This can be explained with the fact that the used external address space is much sparser than the internal one, and thus less IP addresses are merged into a single value through truncation, and the distribution is less affected.

Therefore, when using entropy as metric, the truncation rate can be up to 20 bits for external addresses without affecting the utility of the data for the detection of (D)DoS attacks and network scans, while for internal addresses truncation of 8 bits is acceptable. These specific results are, however, only valid for our data set, and need to be validated with other sources in the future. In particular, tolerable truncation of internal addresses is highly dependent on the network size.

5.3 Measuring Risk of Host Identification

For a sound quantification of the risk of a privacy breach, one must specify the goal and the capabilities of the attacker as well as a metric that is suitable to comprise success of the attack.

Here, the goal of the attacker is host identification. This means, for a given anonymized IP address the attacker tries to recover the original address. We assume that the attacker knows the set of IP addresses in the original trace.

That is, the attacker tries to map the IP addresses from the anonymized traces to the list of known original addresses. Such a list could, at least partially, be compiled by scanning of active hosts and using public information about well-known sites. These assumptions are not quite as strong but similar to those of Coull et al. [44] in the sense that they represent a worst case scenario. In their model, the attacker has exact information about the objects (e.g. hosts, servers or users) and knows the distributions in original data. Similar to Coull et al., the focus of our analysis lies *on the statistical ability to distinguish objects* rather than the practical details of performing an attack.

As we have shown in chapter 4, permutations are reversible using inspection and injection attacks. Hence, to provide stronger host privacy, we truncate bits of IP addresses and study the introduced uncertainty of the attacker when identifying hosts. With truncation of x bits, addresses represent /(32-x) networks. All addresses belonging to the same /(32-x) network are indistinguishable in the truncated trace. If all addresses are active (i.e. present in the traces), this gives the attacker a success probability of $1/2^x$ for correctly guessing a given address.[2]

However, it is usually not the case that all hosts are active in a network. We assume that our attacker knows all active addresses and is therefore able to restrict guesses to active hosts. Following k-anonymity [136], we compute the average size of anonymity sets in which hosts are placed. A host's anonymity set (with truncation of x bits) is given by all active addresses in the same (32-x) subnet. Accordingly, we define our metric $risk(x)$ to be the average probability of guessing a host's address correctly from all addresses in its anonymity set.

Assume for example, that 10^6 addresses are active in the trace. We then apply truncation of $x = 12$ bits, leaving 10^5 unique /24 prefixes in the trace. Then $risk(12) = 1/10$, since 10 indistinguishable addresses remain per prefix, on average. In our evaluation, we derive these numbers empirically from the data set for both internal and external addresses.

Risk Asymmetry

As we have already discussed in section 5.1, internal and external prefixes exhibit different behavior when truncation is applied (see Figure 5.1). Our

[2] It is important to note that external sources of information could help to further distinguish between hosts. For instance, if it is known that there is only one webserver in a subnet, all port 80 traffic can be attributed to this host. Our analysis assumes that no external knowledge is given and therefore provides a lower bound estimation of risk.

risk definition reflects the existing asymmetry between internal and external address space.

For SWITCH-internal addresses, only 10.5% of the 2.2 million IP addresses assigned to SWITCH are active during the hour shown in Figure 5.1. For external addresses, only about 0.08% are active (3.4 million visible versus $2^{32} = 4.3 \cdot 10^9$ potential addresses). This means that internal addresses are much more densely packed within their subnets than external addresses. For an external address, the normal case is that it is unique within its /22 subnet. For internal addresses, there are on average more than 100 active addresses per /22 subnet. Thus, if 10 bits are truncated, internal hosts immediately belong to an anonymity set of size 100, whereas external addresses are still uniquely identifiable.

To compensate this asymmetry, one must truncate roughly 7 bits more from external addresses than from internal addresses. Although the 7 bits are specific to the SWITCH network, a similar effect always occurs when traffic is measured at the borders of a network and a distinction between internal and external address space is made. The fraction of active internal hosts tends to be higher because all traffic from internal hosts passes through the borders while only an insignificant portion of the whole Internet traffic is seen.

It is interesting to ask whether it actually makes sense to protect external addresses in the same way as internal addresses. Whereas our results indicate that external addresses should be anonymized stronger, Pang et al. [108] explicitly apply the weaker prefix-preserving permutation to external and the stronger partial prefix-preserving permutation to internal addresses. They argue that external addresses are more difficult to attack due to their non-locality. However, with truncation, it is exactly this non-locality that makes external addresses stand out within their networks and makes them uniquely identifiable.

It is comprehensible that data publishers are more concerned with privacy of their own network and customers. Anyway, data protection laws do not distinguish between internal or external addresses. In particular, in Switzerland and in the European Union any data that can be used either directly or indirectly, i.e., through the use of additional information, to identify an individual is considered "personal data" and must therefore be protected. In Switzerland, where our data have been collected, "personal data" include data that can be used to identify a legal person, i.e., a company.

In the future, we expect more and more network traces to be published, which is the goal of anonymization research in the first place. Of course, our

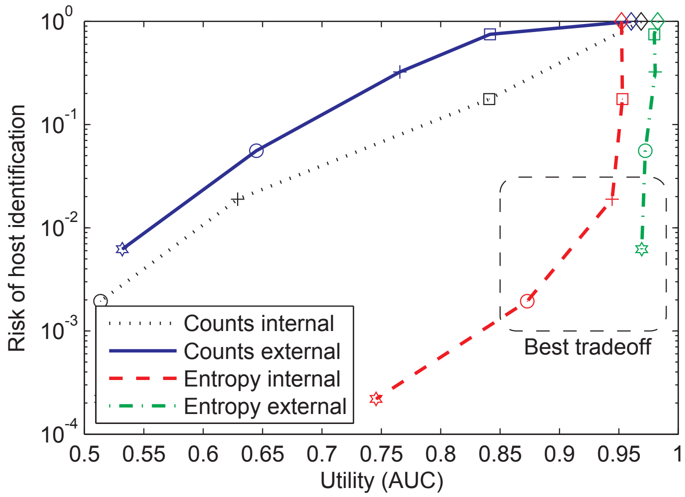


Figure 5.3: *R-U map that illustrates the risk-utility tradeoff for IP address truncation. Markers indicate the number of truncated bits (*$\Diamond : 0$*,* $\Box : 4$*,* $+ : 8$*,* $\circ : 12$*,* $\star : 16$*).*

internal addresses are external addresses in all other data sets. Consequently, if external addresses were anonymized weakly, they could be correlated over different data sets. That would leave people incapable of protecting their own addresses.

5.4 Putting Pieces Together: The Risk-Utility Map

In this Section we summarize the results from section 5.2 and section 5.3 using the R-U map [55]. The R-U map is a parametric plot with a running parameter, having x-coordinates (Utility) and y-coordinates (Risk) determined by a function of the parameter. In our case, the parameter is the number of truncated bits x, taking the values 0-16 in steps of 4. Depending on x, the risk is quantified by $risk(x)$ as defined above, and the utility is quantified by the AUC values shown in Figure 5.2.

Metric	x	Utility	Risk
internal entropy	8	0.94	0.018
internal entropy	12	0.87	0.002
external entropy	16	0.97	0.006

Table 5.1: *Anonymized metrics with the best risk-utility tradeoff (x: truncated bits).*

In Figure 5.3, we plot the utility vs. risk for internal and external count and entropy metrics for different truncation rates (identified by markers). The utility on the x-axis ranges from 0.5 to 1, where 0.5 corresponds to random guessing and 1 to the maximum detection quality. The risk ranges from 1 to 10^{-4}, where 1 represents the maximum risk. Consequently, the sweet spot on this map is the lower right corner with a high utility and low risk.

To our surprise, there are three risk-utility combinations within this area of interest, two for internal entropy and one for external entropy. Table 5.1 summarizes these metrics with the best risk-utility tradeoff.

As expected, both internal and external counts are far off the sweet spot. For counts, as soon as risk drops, utility goes down as well.

5.5 Summary

In this chapter, we have presented a quantitative assessment of the risk-utility tradeoff involved in IP address truncation. We evaluated the utility of anonymized data by performing anomaly detection on original and anonymized data. The risk of host identification was empirically calculated based on host anonymity sets. The results were summarized in a comprehensive Risk-Utility map.

We found that entropy metrics are far more resistant to truncation than unique count metrics. Moreover, there is a fundamental asymmetry between internal and external address distributions for stub networks. Both risk and utility decay faster for internal addresses with increasing number of truncated bits. Consequently, the least useful metric is internal address count as even truncation of only 4 bits renders it useless. On the other hand, the utility of external address entropy is virtually untouched even for truncation of 20 bits. With respect to the risk-utility tradeoff we identified three metrics (calculated on truncated data) with an excellent tradeoff having a remaining utility between 87% and 97% and host identification risk of only 0.2% to 1.8%.

We conclude that truncation offers a better protection against host identification risk than permutation-based IP address anonymization techniques. At the same time, a major part of utility for anomaly detection can be retained using entropy metrics.

Chapter 6

Related Work on Anonymization

If I have seen a little further it is by standing on the shoulders of Giants.

Isaac Newton

The tools and basic techniques for network data anonymization have already been discussed in chapter 2. In this section we discuss measures for utility and privacy and give an overview over attacks against anonymization.

Quantifying Utility

There are a number of proposed global utility measures. For instance, Woo et al. [150] apply empirical CDFs, cluster analysis, and propensity scores to measure data utility. Karr et al. [76] use the Kullback-Leibler divergence to measure the similarity of value distributions before and after anonymization. While these global measures are applicable in many contexts, their values often have no direct interpretation in a specific scenario. Global measures are broad, yet blunt.

More specific to actual use cases, [84] and [155] recently compared the performance of the snort intrusion detection system on anonymized and original packet traces. Brickell and Shmatikov [25] applied machine learning tasks to microdata and found that even modest privacy gains require almost complete destruction of data mining utility. They also report that in most cases, trivial sanitization outperforms complex privacy models such as k-anonymity [136] and l-diversity [92].

Attacks on Anonymization

Privacy has been modeled with many generic statistical definitions based on generalization and suppression of records, such as k-anonymity [136], l-diversity [92], and t-closeness [86]. In k-anonymity, each group of elements with similar non-sensitive quasi-identifiers (called equivalence class) must contain at least k elements, such that from learning values of quasi-identifiers, it is impossible to infer a specific sensitive attribute. The notion of l-diversity addressed issues with this definition and assured that each equivalence class has at least l well-presented values for each sensitive attribute. t-closeness goes one step further and requires that the distribution of sensitive attributes within each equivalence class resembles the overall distribution of these attributes.

While these are valid privacy definitions, they are merely syntactical (i.e., they do not consider data semantics) and mainly studied for microdata, e.g., medical records. The complex semantics of network traffic makes a direct application of these methods to network data difficult, at best [43]. Moreover, it is impractical to apply these types of global constraints to voluminous streaming data. Therefore, privacy properties of network data anonymization techniques have mainly been studied empirically by performing attacks against specific anonymization schemes. In Table 6.1 we briefly summarize de-anonymization studies and note whether the analysis considers injection attacks or not, or have access to some non-public information before the attack. Most of the more recent work has focused on inspection attacks only. Of the studies that consider injection attacks, [108] does not empirically quantify involved risks and [23, 24, 60] assume it is *somehow possible* to mount injection attacks but do not explore the success rate of actually recovering injected patterns in real traffic traces and potential countermeasures. In a slightly different context, [15] uses injection attacks to identify anonymous Internet sensors.

Reference	Inj.	Priv.	Summary
Fan [60]	(✓)	(✓)	Evaluates the robustness of prefix-preserving permutation wrt. the de-anonymization of a number of IP addresses in general. Their analysis is oblivious wrt. how addresses are de-anonymized and the success rate of doing it. Discusses active attacks as a problem.
Ribeiro [115]	-	-	Fingerprints hosts in the live network by scanning 9 TCP ports. Leverages a tree editing algorithm to optimally attack prefix-preserving schemes.
Pang [108]	(✓)	-	Does not empirically quantify de-anonymization risks, but removes scans from traces to mitigate injection attacks. Prefixes are only partially preserved. Acknowledges that defending against general injection attacks, not just scans, is a tough problem.
Brekne [24]	✓	✓	Assumes a very powerful adversary that knows the traffic distribution of the attacked network and performs successful injection attacks to de-anonymize prefix-preserving schemes.
Brekne [23]	✓	-	Shows theoretically that injection attacks can be used to break any *static* pseudonymization scheme. Suggests dynamic (not 1-to-1) mapping of addresses.
Coull [45]	-	-	Fingerprints heavy-hitter hosts by building behavioral profiles and infers network topology from MAC addresses (if available).
Coull [44]	-	✓	Identifies objects at high risk of being de-anonymized with fingerprinting attacks. Assumes an attacker that has complete knowledge of object distributions in original traces.
Koukis [80]	-	-	De-anonymizes web browsing traffic by fingerprinting webservers with characteristic HTTP object sizes. Instead of injecting scans, they propose to detect existing scans in traces.

Table 6.1: *Summary of de-anonymization studies and their attacker models.* Injection*: paper considers injection attacks*. Privileged*: attacker has information not available to the public* before *the attack.*

Chapter 7

The Role of Anonymization Reconsidered

Just because something doesn't
do what you planned it to do
doesn't mean it's useless.

Thomas A. Edison

Throughout Part I of this thesis, we have shown that anonymization involves a delicate privacy-utility tradeoff. Tuning this tradeoff to an acceptable privacy level leaves only little room for applications. A notable exception is the use of entropy metrics for network anomaly detection. The utility of these metrics can still be very high even if a significant amount of detail was removed from traces. Note however, that these results pertain to the detection of network-wide anomalies. Similar approaches on a per-host level are rendered impossible by the applied truncation of address bits.

The success of inspection attacks on anonymization, and, ultimately, the impossibility of defending against injection attacks leads us to call into question the role of anonymization as a complete solution to the problem of data protection. First, we note that there is no general solution to the privacy-utility tradeoff posed by anonymization; it must be considered within the context of the analysis to be done on the data, which information needs to be protected, and which information is required to be present for a given analysis task. A singular, purely technical solution to the problem is not forthcoming; technical aspects of a solution may draw from several techniques.

Analyses may be designed to take advantage of the fact that certain analytically useful techniques (e.g., flow aggregation) have anonymizing effects. Anonymization techniques may be applied in concert with these analysis design techniques to improve data protection. Query-based systems [100] may prove beneficial for network research, in which questions and answers are exchanged while data repositories are protected and immobile, but require further development in order to provide comprehensive ways to safely query a protected database, as well as an analysis of new attack models to which such systems may be vulnerable.

Anonymization must also be considered within its legal context. When anonymization is done for privacy reasons (which is the case, e.g., in data sharing), the data to be protected are specified by data protection law. Laws change depending on the jurisdiction, as does the definition of what “personal data” are, and how data must be protected. Discussions held at a panel on Legal Requirements and Issues in Network Traffic Data Protection [18] among U.S., European, and Japanese lawyers found that the regulations are heterogeneous across jurisdictions, and more work is needed to find appropriate inter-jurisdictional solutions for data sharing.

Specifically, European law [51] defines personal data as *any data identifying a person either directly or indirectly* (i.e., through the use of additional information in possession of third parties). To this category belong, e.g., IP addresses and user profiles. The law restricts the processing allowed on these data and mandates anonymization for subsequent storage or before further processing (e.g., research). Note that anonymized data as defined by the law may identify persons *neither directly nor indirectly*. Therefore, current anonymization techniques applied alone do not provide “anonymization” in the legal sense. Consequently, lawyers start arguing that legislation has to abandon the concept of PII (personally identifying information) and move on to more holistic definitions, considering a series of factors in context-specific solutions [105].

While U.S. law is less restrictive than European with respect to data protection, there is the same disconnect between anonymization in practice and anonymization in the legal sense [32]. The real impediment in this case is in the publication of collected data, not the protection of identifying information it contains. The Stored Communications Act (18 U.S.C. §2702(a)(3)) is the primary legal obstacle to data sharing; [32] advocates primarily architectural changes in data collection and export to address it.

Legal, social, and technical means must be used together to achieve the aims of better data sharing for research and operations. Design of network data analyses must take into consideration the potential privacy impact and legal implications of the data used at each stage of the analysis. We reiterate many of the points made in [4]: threat models are situation dependent, and a clear Acceptable Use Policy is essential, as is deeper interaction between data providers and researchers. In any case, legal solutions are required to enforce those policies, apart from guaranteeing compliance to relevant laws. Here, anonymization protects against accidental disclosure, assists in compliance, and provides evidence of good faith.

Part II

Privacy-Preserving Data Sharing using MPC

Chapter 8

Introduction to Secure Multiparty Computation (MPC)

And now for something
completely different.

Monty Python's Flying Circus

In the second part of this thesis we pursue a completely different approach. Instead of further tuning the ill-fated privacy-utility tradeoff for anonymization, we set out for a very ambitious goal:

1. No compromise on privacy!

2. No compromise on utility!

What sounds impossible at first sight, has in fact been studied in cryptography under the name of *secure multiparty computation* (MPC[1]) for almost three decades. However, as we know, there is no such thing as a free lunch.

[1]In the literature, *secure multiparty computation* is sometimes alternatively abbreviated with SMC.

The price to pay for this miracle is a substantial overhead in computation and communication costs.

MPC techniques allow the privacy-preserving joint processing of data distributed among multiple parties without resorting to a trusted third party. In theory, any computable function on a distributed data set is also securely computable using MPC techniques [68]. Unfortunately, designing solutions that are practical in terms of running time and communication overhead is far from trivial. All the more, this is the case when MPC is adopted for network monitoring and security problems, since network data are typically voluminous and require online processing. In particular, network monitoring protocols must process potentially thousands of input values while meeting *near real-time* guarantees[2]. This is not presently possible with existing MPC frameworks.

One reason why MPC operations are inefficient for networking purposes is that the prevalent design paradigm aims at reducing the number of synchronization rounds between participants at all costs. The overhead of synchronization rounds is believed to be the main source of delay for MPC protocols, while the contribution of local computation time is considered neglectable. In chapter 9, we show that with many MPC operations running in parallel, as required by our scenarios, the time for starting synchronization rounds is amortized over all running operations and is not dominating the running time anymore. On the contrary, running time is dominated by local computations and the number of intermediate values to be exchanged over the network.

Therefore, we first optimize basic MPC operations for networking applications in chapter 9. Using these operations, we then design a variety of protocols for distributed network monitoring and collaborative troubleshooting in chapter 11. All primitives and protocols are implemented in the SEPIA library, which is publicly available [128] and described in more detail in chapter 10. We thoroughly evaluate our protocols both with regard to performance (chapter 12) and usefulness (chapter 13). Because of our optimizations, the protocols turn out to be very practical indeed. Depending on the protocol, they allow 25 to 140 participants to aggregate local traffic information in near real-time. The privacy guarantees provided by our protocols pave the way for new collaborative approaches in network monitoring and security that were simply not possible before.

[2]We define *near real-time* as the requirement of fully processing an x-minute interval of traffic data in no longer than x minutes, where x is typically a small constant. For our evaluation, we use 5-minute windows, which is a frequently-used setting.

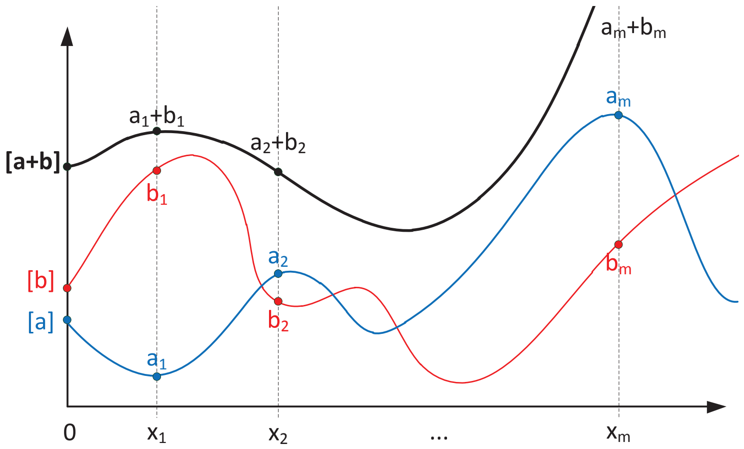


Figure 8.1: *Illustration of Shamir's secret sharing scheme. The secret values a and b are encoded on random polynomials. Evaluation points of these polynomials (the shares) are distributed to the players.*

But to start with, the remainder of this chapter gives an introduction to MPC based on Shamir's secret sharing scheme, which is used in all operations and protocols devised later on.

8.1 Shamir's Secret Sharing Scheme

Our protocols are based on Shamir's secret sharing scheme [129], which is illustrated in Figure 8.1. In order to *share* a secret value s among a set of m players, the dealer generates a random polynomial f of degree (at most) $t < m$ over a prime field $\mathbb{Z}_p$ with $p > s$, such that $f(0) = s$. Each player $i = 1 \dots m$ then receives an evaluation point $s_i = f(x_i)$. s_i is called the *share* of player i. The x_i are public and typically $x_i = i$. The secret s can be reconstructed from any $t+1$ shares using Lagrange interpolation but is completely undefined for t or less shares. To actually *reconstruct* a secret, each player sends his shares to all other players. Each player then locally interpolates the secret.

In particular, for any subset of interpolation points with indices $I \subseteq \{1,\dots,m\}$ and $t < |I| \leq m$, $f(x)$ is given by a linear combination

$$f(x) := \sum_{i \in I} s_i l_i(x) \tag{8.1}$$

where $l_i(x)$ are the so-called Lagrange basis polynomials:

$$l_i(x) := \prod_{j \in I \setminus \{i\}} \frac{x - x_j}{x_i - x_j}.$$

For a given point (x_i, s_i), the corresponding basis polynomial $l_i(x)$ evaluates to 1 exactly for x_i and to 0 for all x_j with $j \neq i$. By scaling the $l_i(x)$ with s_i and adding them up, Equation 8.1 constructs a polynomial that goes through all the interpolation points.

For simplicity of presentation, we use $[s]$ to denote the vector of shares $(s_1,\dots,s_m)$ and call it a *sharing* of s. In addition, we use $[s]_i$ to refer to s_i. Unless stated otherwise, we choose p with 62 bits such that arithmetic operations on shares can be performed by 64-bit CPU instructions directly, not requiring software algorithms to handle big integers.

Arithmetic Operations on Secrets

Given two sharings $[a]$ and $[b]$, we can perform private addition and multiplication of the two values a and b. Because Shamir's scheme is linear, addition of two sharings, denoted by $[a]+[b]$, can be computed by having each player locally add his shares of the two values: $[a+b]_i = [a]_i + [b]_i$ (see Figure 8.1 for an illustration). Similarly, local shares are subtracted to get a share of the difference. To add a public constant c to a sharing $[a]$, denoted by $[a]+c$, each player just adds c to his share, i.e., $[a+c]_i = [a]_i + c$. Similarly, for multiplying $[a]$ by a public constant c, denoted by $c[a]$, each player multiplies its share by c.

Multiplication of two sharings requires an extra round of communication to correct the degree and guarantee randomness of the new polynomial [13, 66]. In particular, to compute $[a][b] = [ab]$, each player first computes $d_i = [a]_i[b]_i$ locally. He then shares d_i to get $[d_i]$. Thus, a distributed multiplication requires a synchronization round with m^2 total messages, as each player i sends to each player j the share $[d_i]_j$. Together, the players then perform a distributed *private* Lagrange interpolation to compute $[ab] = \sum_i \lambda_i [d_i]$ where

λ_i are the Lagrange coefficients. Substituting s_i by $[d_i]$, and setting $x = 0$ in Equation 8.1, the coefficients can be computed as follows:

$$\lambda_i = l_i(0) = \prod_{\substack{1 \le j \le m \\ j \ne i}} \frac{x_j}{x_j - x_i}.$$

The λ_i are public and depend only on the x_i. Therefore they can be computed in advance and reused for multiple secrets. Note that the degree of the intermediate polynomial is doubled, since two polynomials of degree t are multiplied. Therefore, to successfully interpolate $[ab]$, we need at least $2t+1$ shares [66]. If we have m players, this means we have to choose t such that $m \ge 2t+1$. Hence, we always set $t = \lfloor (m-1)/2 \rfloor$ to satisfy this condition and choose t as large as possible.

To specify protocols composed of basic operations, we use a shorthand notation. For instance, we write

$$foo([a], b) := ([a]+b)([a]+b)$$

where foo is the protocol name, followed by input parameters. Valid input parameters are sharings (e.g., $[a]$) and public constants (b). On the right side, the function to be computed is given, a binomial in that case. The output of foo is again a sharing and can be used in subsequent computations. All operations in $\mathbb{Z}_p$ are performed modulo p, therefore p must be large enough to avoid modular reductions of intermediate results, e.g., if we compute $[ab] = [a][b]$, then a, b, and ab must be smaller than p.

8.2 Adversary Models

In MPC, there are two common adversary models:

Semi-honest: In the semi-honest adversary model (a.k.a. honest-but-curious), honest players follow the protocol and do not combine their information. Semi-honest players do follow the protocol but try to infer as much as possible from the values (shares) they learn, also by combining their information. It is possible to securely run protocols if the majority of players is honest [13, 37].

Malicious: As in the semi-honest model, honest players do follow the protocol. But malicious players may cheat in arbitrary ways, e.g., by sending inconsistent values, not sending values at all, or delaying replies arbitrarily. Achieving security is much more complex in the malicious model than in the semi-honest model. It is possible to securely run protocols if at least $2/3$ of all players are honest [13, 37].

In our work, we use the semi-honest adversary model[3]. However, the operations and protocols we devise are independent of the adversary model. If implementations of the basic addition and multiplication operations are secure against malicious adversaries, then so are the protocols built on top. Recently, efficient solutions for dealing with malicious adversaries based on two-dimensional Shamir shares have been proposed [11, 48].

8.3 Network Communication

A set of independent multiplications, e.g., $[ab]$ and $[cd]$, can be performed in parallel in a single round. That is, intermediate results of all multiplications are exchanged in a single synchronization step. *A round simply is a synchronization point where players have to exchange intermediate results* in order to continue computation. While the specification of the protocols is synchronous, we do not assume the network to be synchronous during runtime. In particular, the Internet is better modeled as asynchronous, not guaranteeing the delivery of a message before a certain time. Because we assume the semi-honest model, we only have to protect against network-induced delays of individual messages, potentially leading to a reordering of message arrival. We assume the availability of secure (i.e., authentic and confidential) channels between the players.

In practice, we implement communication channels using SSL sockets over TCP/IP. TCP applies acknowledgments, timeouts, and sequence numbers to preserve message ordering and to retransmit lost messages, providing FIFO channel semantics. We implement message synchronization in parallel threads to minimize waiting time. Each player proceeds to the next round immediately after sending and receiving all intermediate values.

[3]See section 10.2 for a discussion of why we believe it is realistic to assume semi-honest adversaries in our scenarios.

8.4 Security Properties

All the protocols we devise are compositions of the above introduced addition and multiplication primitives, which were proven correct and *information-theoretically* secure by Ben-Or et al. [13]. In particular, they showed that in the semi-honest model, no set of t or less corrupt players gets any additional information other than the final function value. Also, these primitives are *universally composable*, that is, the security properties remain intact under stand-alone and concurrent composition [34]. Because the scheme is information-theoretically secure, i.e., it is secure against computationally unbounded adversaries, the confidentiality of secrets does not depend on the field size p. For instance, regarding confidentiality, sharing a secret s in a field of size $p > s$ is equivalent to sharing each individual bit of s in a field of size $p = 2$.

Since we use SSL for implementing secure channels, the *overall system* relies on a public key infrastructure (PKI) and is only computationally secure.

Chapter 9

Making MPC Practical

In theory, there is no difference between theory and practice. But, in practice, there is.

Jan L. A. van de Snepscheut

Unlike addition and multiplication, comparison of two shared secrets is a very expensive operation. In this chapter, we first review state-of-the-art approaches for doing comparisons in MPC and illustrate why these approaches are not very efficient for our purposes. We then optimize the design and implementation of comparison operations and benchmark their performance. The effect of our optimizations is striking, allowing, for instance, a more then 800 times faster execution of *equal* operations than with comparable frameworks.

9.1 Challenging the Constant-Round Paradigm

The complexity of an MPC protocol is typically assessed counting the number of distributed multiplications and synchronization rounds, because addition and multiplication with public values only require local computation. The overhead of synchronization rounds is believed to be the main source of delay for MPC protocols, while the contribution of local computation time is often considered negligible. For instance, Bar-Ilan et al. state:

"In practical systems, the time spent sending and awaiting messages is large compared to local processing time, and is often the dominating factor in the speed of distributed protocols. It is of great importance to reduce the need for unnecessary interaction among processors." [9, page 2]

Similarly, the authors of FairplayMP explain why they chose an algorithm that evaluates arbitrary functionality in only 8 synchronization rounds:

"We speculate that a major bottleneck of secure computation is the number of communication rounds. [...] The overhead of starting a communication round is caused by the overhead of the communication infrastructure, and also from the fact that in each round all parties need to wait for the slowest party to conclude its work before they can begin the next round." [12, page 2]

Consequently, design of MPC protocols is currently guided by the *constant-round* paradigm [9, 10, 65, 72]. Constant-round means that the number of synchronization rounds in a protocol does not depend on input parameters.

Focusing on comparison operations, Damgård *et al.* introduced the bit-decomposition protocol [47] that achieves comparison by decomposing shared secrets into a shared bit-wise representation. On shares of individual bits, comparison is straight-forward. With $l = \log_2(p)$, the protocols in [47] achieve a comparison with $205l + 188l \log_2 l$ multiplications in 44 rounds and equality check with $98l + 94l \log_2 l$ multiplications in 39 rounds. Subsequently, Nishide and Ohta [103] have improved these protocols by not decomposing the secrets but using bitwise shared random numbers. They do comparison with $279l + 5$ multiplications in 15 rounds and equality check with $81l$ multiplications in 8 rounds. While these are constant-round protocols, they involve lots of multiplications. For instance, a single equality check of two shared IPv4 addresses ($l = 32$) with the protocols in [103] requires 2592 distributed multiplications, each triggering m^2 messages to be transmitted over the network. This is definitely not acceptable if we want to compare lists of shared IP addresses against each other.

Can we do better?

Our key observation for improving efficiency is the following: *For scenarios with many parallel comparison operations, it is possible to build much more*

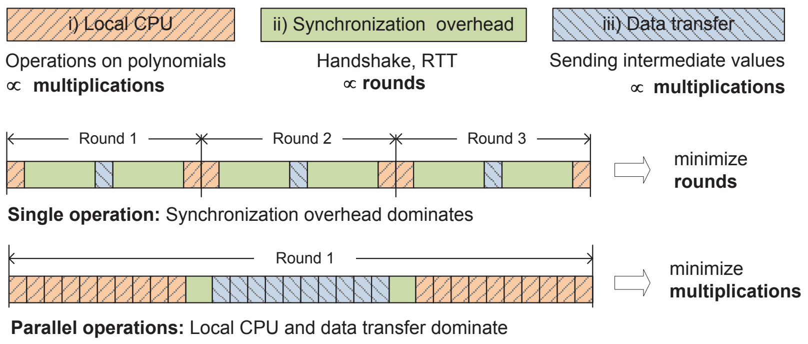

i) Local CPU
Operations on polynomials
∝ multiplications
ii) Synchronization overhead
Handshake, RTT
∝ rounds
iii) Data transfer
Sending intermediate values
∝ multiplications
Round 1
Round 2
Round 3
minimize rounds
Single operation: Synchronization overhead dominates
Round 1
minimize multiplications
Parallel operations: Local CPU and data transfer dominate

Figure 9.1: *Source of delay in composite MPC protocols.*

practical protocols by not enforcing the constant-round property. We design protocols that run in $O(l)$ rounds and therefore are not constant-round, although, once the field size p is defined, the number of rounds is also fixed, i.e., not varying at runtime.

As illustrated in Figure 9.1, the overall local running time of a protocol is determined by

i) the local CPU time spent on computations,

ii) the delay experienced during synchronization, and

iii) the time to transfer intermediate values over the network.

Designing constant-round protocols aims at reducing the impact of ii) by keeping the number of rounds fixed and usually small. To achieve this, large multiplicative constants for the number of multiplications are often accepted (e.g., $279l$ as with [103]). Yet, both i) and iii) directly depend on the number of multiplications. Note that in the literature, the term "synchronization overhead" often does not distinguish between ii) and iii). As long as the overall workload is small and ii) is dominant, this does not make a big difference. But when the number of parallel operations to perform and synchronize becomes significant, i) and iii) become dominant and the reduction of rounds, which only affects ii), is of minor importance. With many parallel operations, the players also have to wait for the slowest participant in each round, here called Joe. But Joe is often the slowest because he either has less CPU cycles available or he is connected to the Internet with less bandwidth than others. Hence, increasing the number of multiplications for trading off the number of rounds in a protocol will only make poor Joe lag behind even more.

In summary, protocols with few rounds (usually constant-round) are certainly faster for applications with few parallel operations. However, with many parallel operations, as required by our scenarios, the impact of network delay is amortized and the number of multiplications (the actual workload) becomes the dominating factor. Our evaluation results confirm this and show that CPU time and network bandwidth are the main constraining factors, calling for a reduction of multiplications.

9.2 Optimized Operations

In the following, we design primitives for equality check and less-than comparison that require significantly less multiplications than existing alternatives.

Equality Check

In the field $\mathbb{Z}_p$ with p prime, Fermat's little theorem states

$$c^{p-1} = \begin{cases} 0 & \text{if } c = 0 \\ 1 & \text{if } c \neq 0 \end{cases} \tag{9.1}$$

Using (9.1) we define a protocol for equality check as follows:

$$equal([a],[b]) := 1 - ([a]-[b])^{p-1}$$

The output of *equal* is [1] in case of equality and [0] otherwise and can hence be used in subsequent computations. Using square-and-multiply for the exponentiation, we implement *equal* with $l+k-2$ multiplications in l rounds, where k denotes the number of bits set to 1 in the binary representation of $p-1$. When checking for different secret sizes below 64 bits, we found that it is easy to find appropriate prime numbers with $k \leq 3$ within around 3 additional bits of the required maximum secret size. For example, when representation of 32-bit secrets is needed, one can use the following prime number with $l = 33$ bits and $k = 3$:

$$p = 6,442,713,089 = 110000000000001000000000000000001_2$$

In the above example for comparing IPv4 addresses, this substantially reduces the multiplication count by a factor of 76 from 2592 to 34.

Less-Than

For less-than comparison, we base our implementation on Nishide's protocol [103]. However, we apply modifications to again reduce the overall number of required multiplications by more than a factor of 10. Nishide's protocol is quite comprehensive and built on a stack of subprotocols for least-significant bit extraction (LSB), operations on bitwise-shared secrets, and (bitwise) random number sharing. The protocol uses the observation that $a < b$ is determined by these three predicates:

- $a < p/2$
- $b < p/2$
- $(a-b) < p/2$

Each predicate is computed by a call of the LSB protocol for $2a$, $2b$, and $2(a-b)$. If $a < p/2$, no wrap-around modulo p occurs when computing $2a$, hence $LSB(2a) = 0$. However, if $a > p/2$, a wrap-around will occur and $LSB(2a) = 1$. Knowing one of the predicates in advance, e.g., because b is not secret but publicly known, saves one of the three LSB calls and hence $1/3$ of the multiplications.

We omit reproduction of the entire protocol and focus on the modifications we apply. An important subprotocol in Nishide's construction is $PrefixOr$. Given a sequence of shared bits $[a_1], \dots, [a_l]$ with $a_i \in \{0,1\}$, $PrefixOr$ computes the sequence $[b_1], \dots, [b_l]$ such that $b_i = \vee_{j=1}^{i} a_j$. Nishide's $PrefixOr$ requires only 7 rounds but $17l$ multiplications. We implement $PrefixOr$ based on the fact that $b_i = b_{i-1} \vee a_i$ and $b_1 = a_1$. The logical `OR` ($\vee$) can be computed using a single multiplication:

$$[x] \vee [y] := [x] + [y] - [x][y].$$

Thus, our $PrefixOr$ requires $l-1$ rounds and only $l-1$ multiplications.

Without compromising security properties, we replace the $PrefixOr$ in Nishide's protocol by our optimized version and call the resulting comparison protocol $lessThan$. A call of $lessThan([a],[b])$ outputs $[1]$ if $a < b$ and $[0]$ otherwise. The overall complexity of $lessThan$ is $24l + 5$ multiplications in $2l + 10$ rounds as compared to Nishide's version with $279l + 5$ multiplications in 15 rounds.

Short Range Check

To further reduce multiplications for comparing small numbers, we devise a check for short ranges, based on our *equal* operation. Consider one wanted to compute $[a] < T$, where T is a small public constant, e.g., $T = 10$. Instead of invoking $lessThan([a], T)$ one can simply compute the polynomial

$$[\phi] := [a]([a]-1)([a]-2)\ldots([a]-(T-1)).$$

If the value of a is between 0 and $T-1$, exactly one term of $[\phi]$ will be zero and hence $[\phi]$ will evaluate to [0]. Otherwise, $[\phi]$ will be non-zero. Based on this, we define a protocol for checking short public ranges that returns [1] if $x \leq [a] \leq y$ and [0] otherwise:

$$shortRange([a],x,y) := equal\big(0, \prod_{i=x}^{y}([a]-i)\big)$$

The complexity of *shortRange* is $(y-x)+l+k-2$ multiplications in $l + \log_2(y-x)$ rounds. Computing $lessThan([a],y)$ requires $16l+5$ multiplications (1/3 is saved because y is public). Hence, regarding the number of multiplications, computing $shortRange([a],0,y-1)$ instead of $lessThan([a],y)$ is beneficial roughly as long as $y \leq 15l$.

Optimized Implementation

In addition to optimizing the design of basic primitives, we also optimize their implementation. First, each connection between two players, along with the corresponding computation and communication tasks, is handled by a separate thread. This limits the impact of varying communication latencies and response times. Furthermore, it lets protocols benefit from multi-core systems for computation-intensive tasks. Secondly, in order to reduce network overhead, intermediate results of parallel operations sent to the same destination are collected and transfered in a single big message instead of many small messages.

The implementation of all the basic primitives is made available in the SEPIA library [128] under the LGPL license. The architecture and design of the SEPIA library are described in chapter 10.

Framework	SEPIA	VIFF	FairplayMP
Technique	Shamir sharing	Shamir sharing	Garbled circuits
Platform	Java	Python	Java
Multipl./s	82,730	326	1.6
Equals/s	2,070	2.4	2.3
LessThans/s	86	2.4	2.3

Table 9.1: *Comparison of framework performance in operations per second with $m = 5$.*

9.3 Benchmark of Basic Operations

In this section we compare the resulting performance of basic SEPIA operations to those of other frameworks such as FairplayMP [12] and VIFF v0.7.1 [48]. Besides performance, one aspect to consider is, of course, usability. Whereas the SEPIA library currently only provides an API to developers, FairplayMP allows the specification of protocols in a high-level language called SFDL and VIFF integrates nicely into the Python language. Furthermore, VIFF implements asynchronous protocols and provides additional functionality, such as security against malicious adversaries and support of MPC based on homomorphic cryptosystems.

Tests were run on 2x Dual Core AMD Opteron 275 machines with 1Gb/s LAN connections. To guarantee a fair comparison, we used the same settings for all frameworks. In particular, the semi-honest model, 5 computation nodes, and 32 bit secrets were used.

General Results

Table 9.1 shows the average number of parallel operations per second for each framework. SEPIA clearly outperforms VIFF and FairplayMP for all operations and is thus much better suited when performance of parallel operations is of main importance. As an example, a run of our event correlation protocol (see section 11.1) taking 3 minutes with SEPIA would take roughly 2 days with VIFF. This extends the range of *practically* runnable MPC protocols significantly.

The results confirm that the garbled boolean circuit approach used by FairplayMP is better suited to perform comparisons than arithmetic operations, such as multiplications. For arithmetic (Shamir) sharing, as used by SEPIA and VIFF, it is exactly the other way round. VIFF outperforms FairplayMP for multiplications and matches it for comparison operations.

Even for multiplications, SEPIA is faster than VIFF, although both rely on the same scheme. We assume this can largely be attributed to the completely asynchronous protocols implemented in VIFF. Whereas asynchronous protocols are very efficient for dealing with malicious adversaries, they make it hard to reduce network overhead by exchanging intermediate results of all parallel operations at once in a single big message. Also, there seems to be a bottleneck in parallelizing large numbers of operations. In fact, when benchmarking VIFF, we noticed that after some point, adding more parallel operations significantly slowed down the average running time per operation.

In SEPIA, approximately $3/4$ of the time spent for $lessThan$ is used for generating sharings of random numbers used in the protocol. These random sharings are independent of input data and could be generated prior to the actual computation, allowing to perform 380 $lessThan$s per second in the same setting.

Rounds versus Multiplications

Notably, SEPIA's *equal* operation is around 24 times faster than its $lessThan$ operation, which requires 24 times more multiplications, but at the same time also twice the number of rounds. This confirms the conjecture made in section 9.1 that with many parallel operations, the number of multiplications becomes the dominating factor in running time.

SEPIA versus Sharemind

Sharemind [16] is another MPC framework using *additive* secret sharing to implement multiplications and greater-or-equal (GTE) comparison. Sharemind is implemented in C++ to maximize performance. Unfortunately, it supports only 3 computation nodes ($m = 3$), i.e., if any two computation nodes collude, the system is broken. Therefore, it is less general than the other frameworks (which support any number of computation nodes) and cannot be directly compared to the results in Table 9.1. However, regarding performance for $m = 3$, Sharemind is comparable to SEPIA. According to [16], Sharemind performs up to 160,000 multiplications and around 330 GTE operations per second. With only 3 computation nodes, SEPIA performs around 145,000 multiplications and 145 $lessThan$s per second (615 with pregenerated randomness). Sharemind does not directly implement $equal$, but it could be implemented using 2 invocations of GTE, leading to ≈ 115 operations/s. SEPIA's *equal* is clearly faster with up to $3,400$ invocations/s. SEPIA demonstrates that operations based on Shamir shares are not necessarily slower than operations in the additive sharing scheme. The key to performance is rather an implementation, which is optimized for a large number of parallel operations. Thus, SEPIA combines speed with the flexibility of Shamir shares, which support any number of computation nodes and are, to a certain degree, robust against node failures.

Computation versus Communication

Depending on the operation type and network conditions, the overall running time is dominated by either local computation or communication time for transferring intermediate values. Private addition is clearly dominated by computation, since it does not require synchronization. For operations built using private multiplications, the network bandwidth is crucial. With 10 Mb/s links, communication time is clearly dominant, requiring 80% of the total running time. With 100 Mb/s links, communication time goes down to 32% and with 1 Gb/s links it is only 21%. However, even if we rule out network bandwidth by running all players on a single node, communication requires still 10% of the total time. A further breakdown of this minimum communication time shows that encryption with SSL is only responsible for about 10% and the remaining 90% must be attributed to the network stack. A complete breakdown of running time for 10 Mb/s and 100 Mb/s is shown in Figure 9.2.

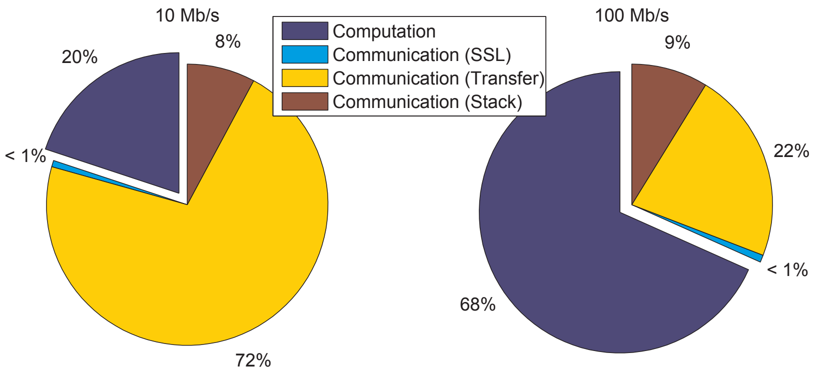


Figure 9.2: *Running time breakdown for distributed multiplications with SEPIA over 10 Mb/s links (left) and 100 Mb/s links (right).*

Chapter 10

SEPIA – A System Overview

Engineers operate at the interface between science and society.

Dean Gordon Brown

In this chapter, we describe the SEPIA library[1], in which we have implemented a complete set of basic MPC primitives as well as our optimized operations from chapter 9. All the privacy-preserving protocols developed in the next chapter use the SEPIA library.

The chapter starts with a definition of the two basic roles in SEPIA: input peers and privacy peers. It then specifies the adversary model and security assumptions made. In section 10.3, the design and API are detailed and section 10.4 illustrates the use of SEPIA with a concrete programming example.

10.1 Two Roles: Input and Privacy Peers

The players in MPC are usually true peers with equal duties and benefits. Each of them contributes input data to the computation, takes part in the computation, and receives the final result. This approach, however, does not scale well if we want to collect input data from hundreds or even thousands of participants. In particular, the computation and communication costs scale quadrat-

[1]**SEPIA** is short for **Se**curity through **P**rivate **I**nformation **A**ggregation.

ically with the number of players. In our system, as depicted in Figure 1.3 on page 10, we therefore define two different roles to separate data provisioning from the computation task:

Input peers: The n input peers want to jointly compute the value of a public function $f(x_1,\ldots,x_n)$ on their private data x_i without disclosing anything about x_i. They apply Shamir's secret sharing scheme to their private data and distribute the shares to the privacy peers.

Privacy peers: The m privacy peers perform the computation of $f()$ by simulating a trusted third party (TTP). They receive shares of the input peers' private data and perform MPC operations on the shared secrets. In the end, they reconstruct the computation result and send it back to the input peers.

Each entity can take both roles, acting only as an input peer, privacy peer or both[2].

Although the number of privacy peers m has a quadratic impact on the total communication and computation costs, there are also m privacy peers sharing the load. That is, if the network capacity is sufficient, the overall running time of the protocols scales linearly with m rather than quadratically. However, there is also a benefit in using many privacy peers, because the number of tolerated colluding privacy peers scales linearly with m, as well. Hence, the choice of m involves a privacy-performance tradeoff. The separation of roles into input and privacy peers allows the tuning of this tradeoff independently of the number of input providers.

10.2 Adversary Model and Security Assumptions

We use the semi-honest adversary model (see section 8.2) for privacy peers. The privacy and correctness guarantees provided by our protocols are determined by Shamir's secret sharing scheme. In particular, the protocols are secure for $t < m/2$ semi-honest privacy peers, i.e., as long as the majority of privacy peers is honest. Even if some of the input peers do not trust each other,

[2]Similar role definitions are also made by other frameworks. For instance, FairplayMP [12] defines three roles: input players, computation players, and result players. The first two are similar to the input and privacy peers in SEPIA.

we believe it is realistic to assume that they will agree on a set of most-trusted participants (or external entities) for hosting the privacy peers. Also, we believe it is realistic to assume that the privacy peers indeed follow the protocol. If they are operated by input peers, they are likely interested in the correct outcome of the computation themselves and will therefore comply. External privacy peers are selected due to their good reputation or are being payed for a service. In both cases, they will do their best not to offend their customers by tricking the protocol.

Note that even though MPC guarantees that no information is leaked *from the computation process*, learning the final value of $f()$ might be sensitive in itself. We emphasize that *it is the responsibility of the input peers to verify that learning $f()$ is acceptable*, in the same way as they have to verify this when using a real TTP. One approach of doing this systematically is *differential privacy*, which is discussed in section 14.5.

Prior to running the protocol, the m privacy peers set up a secure, i.e., confidential and authentic, channel to each other. In addition, each input peer creates a secure channel to each privacy peer. We assume that the required public keys and/or certificates have been securely distributed beforehand.

10.3 Design and API

In this section we discuss the design of SEPIA, it's functional building blocks, and the main API hooks.

Java versus C++

SEPIA is written in Java to provide platform independence. It is an interesting question to ask whether performance of SEPIA could be improved significantly by porting it to C++. Since the Java virtual machine uses just-in-time (JIT) compilation, it compiles hotspots to native code during runtime anyway. In current benchmarks [73], Java goes almost head to head with C++. If time consumption of local computation is of main importance, one should consider implementing Shamir's sharing scheme in hardware, as it has been done recently to capture traffic from gigabit networks [149]. However, independently of the programming language and hardware optimizations, network bandwidth can still be a limiting factor, at least if link speeds are less than 100 Mb/s (see Figure 9.2). In summary, we think porting SEPIA to C++ is not of top priority.

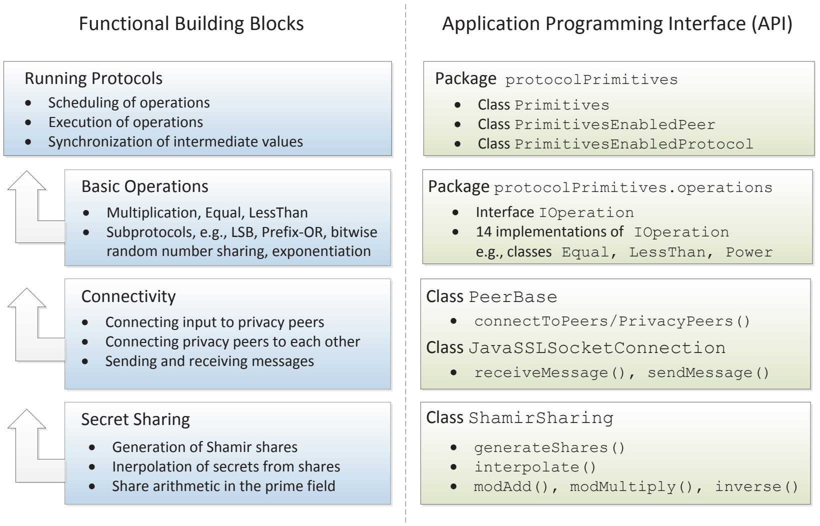

Functional Building Blocks
Application Programming Interface (API)
Running Protocols
Scheduling of operations
Execution of operations
Synchronization of intermediate values
Basic Operations
Multiplication, Equal, LessThan
Subprotocols, e.g., LSB, Prefix-OR, bitwise random number sharing, exponentiation
Connectivity
Connecting input to privacy peers
Connecting privacy peers to each other
Sending and receiving messages
Secret Sharing
Generation of Shamir shares
Inerpolation of secrets from shares
Share arithmetic in the prime field
Package protocolPrimitives
Class Primitives
Class PrimitivesEnabledPeer
Class PrimitivesEnabledProtocol
Package protocolPrimitives.operations
Interface IOperation
14 implementations of IOperation e.g., classes Equal, LessThan, Power
Class PeerBase
connectToPeers/PrivacyPeers()
Class JavaSSLSocketConnection
receiveMessage(), sendMessage()
Class ShamirSharing
generateShares()
interpolate()
modAdd(), modMultiply(), inverse()

Figure 10.1: *Functional building blocks and corresponding API elements of the SEPIA library.*

Building Blocks

We now discuss the functional building block as illustrated in Figure 10.1 and also describe the most important API hooks.

The SEPIA library implements Shamir's secret sharing scheme as introduced in section 8.1. The class `ShamirSharing` offers methods for generating shares from private data, interpolating secrets from shares, and performing arithmetic operations in the prime field of order p. For instance, modular addition, multiplication, element inversion, and square root are supported.

The class `PeerBase`, from which input and privacy peer implementations are derived, provides methods for connecting input peers to privacy peers and also privacy peers to each other. Addresses of privacy peers are read from a local configuration file. As already mentioned, communication is secured using SSL. The class `JavaSSLSocketConnection` provides methods for sending and receiving messages. Messages are passed as objects and serialized to sockets using Java object serialization. Optionally, data written to sockets can be compressed. Keys and certificates for securing the SSL socket are read from a local Java keystore. It is possible to delay the start of a computation until a minimum number of input and/or privacy peers are connected. This gives the input peers the ability to define an acceptable level of privacy by only participating in the computation if i) a certain number of other input peers contribute to the aggregate result, and ii) there are enough privacy peers present to guarantee an honest majority.

The most important package in SEPIA is the `protocolPrimitives` package. It defines a standard interface for MPC operations, called `IOperation`. All MPC operations, such as multiplications, equality testing, less-than comparisons, and several important subprotocols, implement the `IOperation` interface and can be scheduled for execution via the `Primitives` class. Operations can easily be composed, i.e., each operation can be called as a sub-operation from within a parent operation. In table 10.1, we give an overview over the 14 currently implemented operations. Note that there is no special operation for addition, since private addition requires no interaction between the privacy peers. In practice, private addition is implemented with a single call of `ShamirSharing.modAdd(shareOfA, shareOfB)`.

In full-blown protocols, large numbers of these basic operations need to be composed and executed in parallel. In addition, most of these operations consist of several synchronization rounds, as we have discussed in chapter 9. The functionality for scheduling, executing, and synchronizing intermediate data exchange for many parallel operations is implemented in the `protocolPrimitives` package. The user of the SEPIA library only needs to bind the data shares to operations, schedule these operations using the `Primitives` class, and call `doOperations()` from within protocol threads. This effectively hides the complexity of specific operations and their distributed execution from the user.

Multi-Threading and Reduction of Network Overhead

In order to limit the impact of varying communication latencies and response times of privacy peer connections, each connection, along with the corresponding computation and communication tasks, is handled by a separate thread. Therefore, the number of threads running on each privacy peer corresponds to $m-1$ (i.e., the number of total privacy peers minus one).[3] This also implies that SEPIA protocols benefit from multi-core systems for computation-intensive tasks.

[3]In addition, there is one thread for each input peer connection. But these threads are not used to perform computations.

Another performance improvement implemented in SEPIA is the reduction of network overhead. If a protocol executes a large number of parallel operations this also causes a large number of synchronization messages to be transmitted. For each pair of privacy peers, each operation requires an intermediate message per synchronization round. In order to reduce synchronization overhead, intermediate results of parallel operations sent to the same destination are collected and transfered in a single big message instead of many small messages. This greatly reduces the overhead for message handling and encapsulation. At the same time, by producing one big chunk of data, it allows network protocols to efficiently use available bandwidth.

10.4 Programming Example

In this section, we round off the chapter with a concrete programming example. Alluding to Yao's millionaires' problem [153], we assume three millionaires M1, M2, and M3 that want to privately compare their fortunes to learn which of them is the richest. The protocol privately compares the fortunes pairwise using SEPIA's `lessThan` operation. The results of these comparisons are then publicly reconstructed.

Figure 10.2 shows example code for an input peer representing one of the millionaires. First, the input peer generates shares of its fortune (lines 7-8). The shares are then sent to the privacy peers (lines 11-13), for which example code is shown in Figure 10.3. The privacy peer first receives the shares from the three input peers. It then schedules `lessThan` comparisons for all combinations of input secrets (lines 6-8). The execution and synchonization is started by calling `doOperations()` (line 10). After SEPIA has executed all scheduled operations, it returns from the call and the privacy peer retrieves shares of the computation results (lines 13-16). In this example, the results are shares of pairwise fortune comparisons. At this point, the actual computation is already finished and the privacy peer schedules reconstruction of the pairwise comparisons (lines 19-21). Again, these operations are executed by calling `doOperations()`. The results of the reconstruction operations are no longer shares of secrets but the actual values of the secrets. Since we reconstruct the results of `lessThan` operations, the outcome is 1 if the first secret was smaller than the second or 0 otherwise (lines 25-27). Assuming that no two fortunes are exactly the same, we can now conclude that

1. `!isM1poorerThanM2 && !isM1poorerThanM3` $\Rightarrow$ M1 is the richest!

```java
ShamirSharing sharing = new ShamirSharing();
sharing.setFieldPrime(1401085391); // 31 bit
sharing.setNrOfPrivacyPeers(nrOfPrivacyPeers);
sharing.init();

// Sharing my fortune
long[] myFortune = new long[]{12345678};
long[][] shares = sharing.generateShares(myFortune);

// Send shares to each privacy peer
for(int i=0; i<nrOfPrivacyPeers; i++) {
  connection[i].sendMessage(shares[i]);
}
```

Figure 10.2: *Example code for an input peer. In this example, it is a millionaire sharing his fortune.*

2. `isM1poorerThanM2 && !isM2poorerThanM3` $\Rightarrow$ M2 is the richest!
3. `isM2poorerThanM3 && isM1poorerThanM3` $\Rightarrow$ M3 is the richest!

Note that the millionaires do not only learn who is the richest, but also who is the second richest and the poorest. If they wanted to avoid this, they could easily perform the above computations for the final predicates “Mx is the richest” in MPC.[4]

The source code of the SEPIA library and the privacy-preserving protocols developed in the next chapter is available under the LGPL license on the SEPIA project web page [128]. The web page also provides pre-configured examples for the protocols and a user manual. The user manual describes in more detail how to use and configure the protocols. Moreover, it includes a step-by-step tutorial on how to develop new protocols.

[4]The logical AND (`&&`) is simply a multiplication and the NOT (`!`) of $[x]$ is given by $1-[x]$.

```
... // receive the shares of M1, M2, and M3 (from input peers)
ProtocolPrimitives primitives = privacyPeer.getPrimitives();

if(barrier.await()==0) { // choose one thread
  // Schedule comparisons of all the secrets
  primitives.lessThan(0, new long[]{shareOfM1, shareOfM2});
  primitives.lessThan(1, new long[]{shareOfM2, shareOfM3});
  primitives.lessThan(2, new long[]{shareOfM1, shareOfM3});
}
doOperations(); // Process operations in parallel

if(barrier.await()==0) { // choose one thread
  // Retrieve shares of the comparison results
  long shareOfM1LessThanM2 = primitives.getResult(0);
  long shareOfM2LessThanM3 = primitives.getResult(1);
  long shareOfM1LessThanM3 = primitives.getResult(2);

  // Schedule reconstruction of comparison results
  primitives.reconstruct(0, new long[]{shareOfM1LessThanM2});
  primitives.reconstruct(1, new long[]{shareOfM2LessThanM3});
  primitives.reconstruct(2, new long[]{shareOfM1LessThanM3});
}
doOperations(); // Process operations in parallel

boolean isM1poorerThanM2 = (primitives.getResult(0)==1);
boolean isM2poorerThanM3 = (primitives.getResult(1)==1);
boolean isM1poorerThanM3 = (primitives.getResult(2)==1);
```

Figure 10.3: *Example code for a privacy peer comparing the fortunes of three millionaires M1, M2, and M3 to find which of them is the richest.*

Operation class name	Description
`BatchGenerateBitwise-RandomNumbers`	Operation for creating many bitwise random numbers at once. Uses `Generate-BitwiseRandomNumber`.
`BitwiseLessThan`	Implements less-than on bitwise shared numbers. Used by `LeastSignificant-Bit`.
`Equal`	Performs equaliy testing. See section 9.2.
`GenerateBitwiseRandom-Number`	Generates random numbers that are shared bit-by-bit. Used by `Least-SignificantBit`.
`GenerateRandomBit`	Generates a single random bit (see [103]). Used by `GenerateBitwiseRandom-Number`.
`GenerateRandomNumber`	Generates a shared random number. Used by `GenerateRandomBit`.
`LeastSignificantBit`	Computes the least significant bit of a secret (see [103]). Used by `LessThan`.
`LessThan`	Implements our optimized less-than comparison. See section 9.2.
`LinearPrefixOr`	Computes the prefix-OR (see section 9.2). Used by `BitwiseLessThan`.
`Multiplication`	Basic private multiplication (see section 8.1). Used by most other operations.
`Power`	Raises a secret to a given (public) power using square-and-multiply. Used by `Equal`.
`Product`	Multiplies a sequence of f factors in $\log_2(f)$ rounds. Used by `SmallInterval-Test`.
`Reconstruction`	Reconstructs a shared secret. See section 8.1.
`SmallIntervalTest`	Implements the short range check. See section 9.2.

Table 10.1: *MPC operations implemented in SEPIA, in alphabetical order.*

Chapter 11

Privacy-Preserving Protocols

Relying on the government to protect your privacy is like asking a peeping tom to install your window blinds.

John Perry Barlow

In this chapter, we compose the basic operations introduced in chapter 8 and chapter 9 into full-blown protocols for network event correlation and statistics aggregation. We conclude the chapter with a taxonomy of application scenarios for which we believe privacy-preserving protocols are useful.

In general, the design of efficient composite MPC protocols faces two main challenges: Firstly, as discussed in chapter 9, synchronization costs of MPC protocols can only be amortized if many of their operations are independent and can be performed in parallel. Hence, one challenge in designing new protocols is indeed to use as little MPC operations as possible but at the same time *as many parallelizable MPC operations as possible*. In practice, it is possible that a protocol variant A requiring 1,000 basic operations outperforms variant B with only 100 operations, if A's operations are independent and B's operations need to be performed sequentially.

Secondly, algorithms have to be *data-oblivious*. Whereas ordinary algorithms may contain conditional statements, such as "if $(a < b)$ then do

X, otherwise do Y", MPC protocols cannot simply compute predicates like "$a < b$" and decide what to do based on the value of the predicate. MPC algorithms guarantee that not a single bit about input data is leaked. Therefore, learning the value of such predicates may constitute a privacy breach already and has to be omitted. Hence, MPC algorithms have to be specified in a data-independent way. Consequently, simple and heavily-used data structures, such as arrays that provide access to an element at index i within one CPU cycle, are no longer available if i has to remain secret [49]. This severely complicates algorithm design and requires a well-balanced tradeoff between performance, complexity, and accuracy considerations.

Each of the developed protocols is designed to run on continuous streams of input traffic data partitioned into time windows of a few minutes. For sake of simplicity, the protocols are specified for a single time window. The performance of all protocols is evaluated in detail in chapter 12. An overview over the notation used for the different protocols is given in Table 11.1.

11.1 Event Correlation

The first protocol we present enables the input peers to privately aggregate arbitrary network events. An event e is defined by a key-weight pair $e = (k, w)$. This notion is generic in the sense that keys can be defined to represent arbitrary types of network events, which are uniquely identifiable. The key k could for instance be the source IP address of packets triggering IDS alerts, or the source address concatenated with a specific alert type or port number. It could also be the hash value of extracted malicious payload or represent a uniquely identifiable object, such as popular URLs, of which the input peers want to compute the total number of hits. The weight w reflects the impact (count) of this event (object), e.g., the frequency of the event in the current time window or a classification on a severity scale.

Each input peer shares at most s local events per time window. The goal of the protocol is to reconstruct an event if and only if a minimum number of input peers T_c report the same event and the aggregated weight is at least T_w. The rationale behind this definition is that an input peer does not want to reconstruct local events that are unique in the set of all input peers, exposing sensitive information asymmetrically. But if the input peer knew that, for example, three other input peers report the same event, e.g., a specific intrusion alert, he would be willing to contribute his information and collaborate.

SEPIA Library and Basic Secret Sharing			
n:	# of input peers	m:	# of privacy peers
p:	order of field $\mathbb{Z}_p$, prime	l:	bit size of secrets, $l = \log_2(p)$
t:	degree of polynomials	PP:	short for privacy peer
Event Correlation			
T_c:	min. # of reporting input peers	T_w:	min. aggregate weight
e_{ij}:	event j of input peer i	s:	# of events per input peer
k_{ij}:	key of event e_{ij}	w_{ij}:	weight of event e_{ij}
C_{ij}:	aggregate count of e_{ij}	W_{ij}:	aggregate weight of e_{ij}
w_{max}:	maximum allowed weight		
Vector Addition			
r:	dimension of input vectors	d_i:	vector of input peer i
Entropy Computation			
p_k:	probability of item k	q:	parameter of Tsallis entropy
s_k^i:	counts (item k, input peer i)	s_k:	aggregate count (item k)
S:	overall item count		
Distinct Count			
s_k^i:	local presence (item k, input peer i)	c_k^i:	$c_k^i = \neg s_k^i$
c_k:	aggregate negated count for item k	K:	size of item domain
Top-k Queries			
k:	number of top items (top-k)	H	size of hash arrays
k_j^i:	key at position j (input peer i)	v_j^i:	value at position j (input peer i)
V_j:	aggregate value at position j	τ:	threshold in binary search
S:	# of hash arrays	M:	max. expected value

Table 11.1: *Table of notations.*

Likewise, an input peer might only be interested in reconstructing events of a certain impact, having a non-negligible aggregated weight.

More formally, let $[e_{ij}] = ([k_{ij}], [w_{ij}])$ be the shared event j of input peer i with $j \leq s$ and $i \leq n$. Then we compute the aggregated count C_{ij} and weight W_{ij} according to (11.1) and (11.2) and reconstruct e_{ij} iff (11.3) holds.

$$[C_{ij}] := \sum_{i' \neq i, j'} equal([k_{ij}], [k_{i'j'}]) \tag{11.1}$$

$$[W_{ij}] := \sum_{i' \neq i, j'} [w_{i'j'}] \cdot equal([k_{ij}], [k_{i'j'}]) \tag{11.2}$$

$$([C_{ij}] \geq T_c) \wedge ([W_{ij}] \geq T_w) \tag{11.3}$$

1. **Share Generation:** Each input peer i shares s distinct events e_{ij} with $w_{ij} < w_{max}$ among the privacy peers (PPs).
2. **Weight Verification:** Optionally, the PPs compute and reconstruct $lessThan([w_{ij}], w_{max})$ for all weights to verify that they are smaller than w_{max}. Misbehaving input peers are disqualified.
3. **Key Verification:** Optionally, the PPs verify that each input peer i reports distinct events, i.e., for each event index a and b with $a < b$ they compute and reconstruct $equal([k_{ia}], [k_{ib}])$. Misbehaving input peers are disqualified.
4. **Aggregation:** The PPs compute $[C_{ij}]$ and $[W_{ij}]$ according to (11.1) and (11.2) for $i \leq \hat{i}$ with $\hat{i} = min(n - T_c + 1, n)$. [a] All required *equal* operations can be performed in parallel.
5. **Reconstruction:** For each event $[e_{ij}]$, with $i \leq \hat{i}$, condition (11.3) has to be checked. Therefore, the PPs compute

   $$[t_1] = shortRange([C_{ij}], T_c, n), \quad [t_2] = lessThan(T_w - 1, [W_{ij}])$$

   Then, the event is reconstructed iff $[t_1] \cdot [t_2]$ returns 1. The set of input peers with $i > \hat{i}$ reporting a reconstructed event $\overline{r} = (\overline{k}, \overline{w})$ is computed by reusing all the *equal* operations performed on $\overline{r}$ in the aggregation step. That is, input peer i' reports $\overline{r}$ iff $\sum_j equal([\overline{k}], [k_{i'j}])$ equals 1. This can be computed using local addition for each remaining input peer and each reconstructed event. Finally, all reconstructed events are sent to all input peers.

[a] For instance, if $n = 10$ and $T_c = 7$, each event that needs to be reconstructed according to (11.3) must be reported by at least one of the first 4 input peers. Hence, it is sufficient to compute the C_{ij} and W_{ij} for the first $n - T_c + 1 = 4$ input peers.

Figure 11.1: *Algorithm for event correlation protocol.*

Reconstruction of an event e_{ij} includes the reconstruction of k_{ij}, C_{ij}, W_{ij}, and the list of input peers reporting it, but the w_{ij} remain secret. The detailed algorithm is given in Figure 11.1.

Input Verification

In addition to merely implementing the correlation logic, we devise two optional input verification steps. In particular, the privacy peers check that shared weights are below a maximum weight w_{max} and that each input peer shares distinct events. These verifications are *not* needed to secure the computation process, but they serve two purposes. First, they protect from misconfigured input peers and flawed input data. Secondly, they protect against input peers that try to deduce information from the *final computation result*. For instance, an input peer could add an event $T_c - 1$ times (with a total weight of at least T_w) to find out whether any other input peers report the same event. These input verifications mitigate such attacks.

Probe Response Attacks

If aggregated security events are made publicly available, this enables probe response attacks against the system [15]. The goal of probe response attacks is not to learn private input data but to identify the sensors of a distributed monitoring system. To remain undiscovered, attackers then exclude the known sensors from future attacks against the system. While defending against this in general is an intractable problem, [130] identified that the suppression of low-density attacks provides some protection against basic probe response attacks. Filtering out low-density attacks in our system can be achieved by setting the thresholds T_c and T_w sufficiently high.

Complexity

The overall complexity, including verification steps, is summarized below in terms of operation invocations and rounds:

equal:	$O\big((n-T_c)ns^2\big)$
lessThan:	$(2n-T_c)s$
shortRange:	$(n-T_c)s$
multiplications:	$(n-T_c)\cdot(ns^2+s)$
rounds:	$7l+\log_2(n-T_c)+26$

The protocol is clearly dominated by the number of *equal* operations required for the aggregation step. It scales quadratically with s, however, depending on T_c, it scales linearly or quadratically with n. For instance, if T_c has a constant offset to n (e.g., $T_c = n-4$), only $O(ns^2)$ *equal*s are required. However, if $T_c = n/2$, $O(n^2s^2)$ *equal*s are necessary.

Optimizations

To avoid the quadratic dependency on s, one could use an MPC version of a binary search algorithm that finds a secret $[a]$ in a sorted list of secrets $\{[b_1],\ldots,[b_s]\}$ with $\log_2 s$ comparisons by comparing $[a]$ to the element in the middle of the list, here called $[b_*]$. Then a new list is constructed, being

1. **Share Generation:** Each input peer i shares its input vector $\mathbf{d_i} = (x_1, x_2, \ldots, x_r)$ among the PPs. That is, the PPs obtain n vectors of sharings $[\mathbf{d_i}] = ([x_1], [x_2], \ldots, [x_r])$.
2. **Summation:** The PPs compute the sum $[\mathbf{D}] = \sum_{i=1}^{n} [\mathbf{d_i}]$.
3. **Reconstruction:** The PPs reconstruct all elements of $\mathbf{D}$ and send them to all input peers.

Figure 11.2: *Algorithm for vector addition protocol.*

the first or second half of the original list, depending on $lessThan([a],[b_*])$. The procedure is repeated recursively until the list has size 1. This allows comparison of all events of two input peers with only $O(s \log_2 s)$ instead of $O(s^2)$ comparisons.

To further reduce the number of *equal* operations, the protocol can be adapted to receive *incremental updates* from input peers. That is, input peers submit a list of events in each time window and inform the privacy peers, which event entries have a different key from the previous window. Then, only comparisons of updated keys have to be performed and overall complexity is reduced to $O(us(n - T_c))$, where u is the number of changed keys in that window. This requires, of course, that information on input set dynamics is not considered private.

11.2 Network Traffic Statistics

In this section, we present protocols for the computation of multi-domain traffic statistics including the aggregation of additive traffic metrics, the computation of feature entropy, and the computation of distinct item count. These statistics find various applications in network monitoring and management.

11.2.1 Vector Addition

To support basic additive functionality on time series and histograms, we implement a vector addition protocol. Each input peer i holds a private r-dimensional input vector $\mathbf{d_i} \in \mathbb{Z}_p^r$. Then, the vector addition protocol computes the sum $\mathbf{D} = \sum_{i=1}^{n} \mathbf{d_i}$. We provide a brief description of the corresponding SEPIA protocol in Figure 11.2. This protocol requires no distributed multiplications and only one round.

1. **Share Generation:** Each input peer holds an r-dimensional private input vector $\mathbf{s^i} \in \mathbb{Z}_p^r$ representing the local item histogram, where r is the number of items and s_k^i is the count for item k. The input peers share all elements of their $\mathbf{s^i}$ among the PPs.
2. **Summation:** The PPs compute the item counts $[s_k] = \sum_{i=1}^n [s_k^i]$. Also, the total count $[S] = \sum_{k=1}^r [s_k]$ is computed and reconstructed.
3. **Exponentiation:** The PPs compute $[(s_k)^q]$ using square-and-multiply.
4. **Entropy Computation:** The PPs compute the sum $\sigma = \sum_k [(s_k)^q]$ and reconstruct σ. Finally, at least one PP uses σ to (locally) compute the Tsallis entropy $H_q(Y) = \frac{1}{q-1}(1 - \sigma/S^q)$.

Figure 11.3: *Algorithm for entropy protocol.*

11.2.2 Entropy Computation

The computation of the entropy of feature distributions has been successfully applied in network anomaly detection, e.g. [22, 83, 87, 157]. Commonly used feature distributions are, for example, those of IP addresses, port numbers, flow sizes or host degrees. The Shannon entropy of a feature distribution Y is $H(Y) = -\sum_k p_k \cdot \log_2(p_k)$, where p_k denotes the probability of an item k. If Y is a distribution of port numbers, p_k is the probability of port k to appear in the traffic data. The number of flows (or packets) containing item k is divided by the overall flow (packet) count to calculate p_k. Tsallis entropy is a generalization of Shannon entropy that also finds applications in anomaly detection [139, 157]. The 1-parametric Tsallis entropy is defined as:

$$H_q(Y) = \frac{1}{q-1}\Big(1 - \sum_k (p_k)^q\Big). \tag{11.4}$$

and has a direct interpretation in terms of moments of order q of the distribution. In particular, the Tsallis entropy is a generalized, non-extensive entropy that, up to a multiplicative constant, equals the Shannon entropy for $q \to 1$. For generality, we select to design an MPC protocol for the Tsallis entropy.

Entropy Protocol

A straight-forward approach to compute entropy is to first find the overall feature distribution Y and then to compute the entropy of the distribution. In particular, let p_k be the overall probability of item k in the union of the private

1. **Share Generation:** Each input peer i shares its negated local counts $c_k^i = \neg s_k^i$ among the PPs.
2. **Aggregation:** For each item k, the PPs compute $[c_k] = [c_k^1] \wedge [c_k^2] \wedge \dots [c_k^n]$. This can be done in $\log_2 n$ rounds. If an item k is reported by any input peer, then c_k is 0.
3. **Counting:** Finally, the PPs build the sum $[\sigma] = \sum [c_k]$ over all items and reconstruct σ. The distinct count is then given by $K - \sigma$, where K is the size of the item domain.

Figure 11.4: *Algorithm for distinct count protocol.*

data and s_k^i the local count of item k at input peer i. If S is the total count of the items, then $p_k = \frac{1}{S} \sum_{i=1}^n s_k^i$. Thus, to compute the entropy, the input peers could simply use the addition protocol to add all the s_k^i's and find the probabilities p_k. Each input peer could then compute $H(Y)$ locally. However, the distribution Y can still be very sensitive as it contains information for each item, e.g., per address prefix. For this reason, we aim at computing $H(Y)$ without reconstructing any of the values s_k^i or p_k. Because the rational numbers p_k can not be shared directly over a prime field, we perform the computation separately on private numerators (s_k^i) and the public overall item count S. The entropy protocol achieves this goal as described in Figure 11.3. It is assured that sensitive intermediate results are not leaked and that input and privacy peers *only* learn the final entropy value $H_q(Y)$ and the total count S. S is not considered sensitive as it only represents the total flow (or packet) count of all input peers together. This can be easily computed by applying the addition protocol to volume-based metrics. The complexity of this protocol is $r \log_2 q$ multiplications in $\log_2 q$ rounds.

11.2.3 Distinct Count

In this section, we devise a simple distinct count protocol leaking no intermediate information. Let $s_k^i \in \{0,1\}$ be a boolean variable equal to 1 if input peer i sees item k and 0 otherwise. We first compute the logical `OR` of the boolean variables to find if an item was seen by any input peer or not. Then, simply summing the number of variables equal to 1 gives the distinct count of the items. According to De Morgan's Theorem, $a \vee b = \neg(\neg a \wedge \neg b)$. This means the logical `OR` can be realized by performing a logical `AND` on the negated variables. This is convenient, as the logical `AND` is simply the product of two variables. Using this observation, we construct the protocol described in Figure 11.4. This protocol guarantees that only the distinct count is learned from the computation; the set of items is *not* reconstructed. However, if the input peers agree that the item set is not sensitive it can easily be reconstructed after step 2. The complexity of this protocol is $(n-1)r$ multiplications in $\log_2 n$ rounds.

11.3 Top-k Queries

Even though the event correlation protocol can be used to identify distributed heavy-hitters, it relies on the configuration of thresholds controlling which items are reconstructed. However, in presence of an anomaly, traffic conditions are expected to change and previous thresholds might become inappropriate, leading to a revelation of way too much sensitive information or no information at all. The PPTKS[1] protocol, which is introduced in the following section, allows the specification of the exact number of elements k that will be revealed, automatically adapting to dynamic traffic conditions. However, depending on the application scenario, either fixed or dynamic thresholds may be desirable. In addition, PPTKS applies probabilistic data structures to scale much better for large item distributions. Whereas the event correlation protocol is efficient for aggregating around 100 local events, e.g., the local top-100 IP addresses, PPTKS is designed to consider the full local item distributions, enabling the aggregation of hundreds of thousands or even millions of items. Considering the full item distributions is necessary, because in theory, a global top-10 item might be ranked number 11 or more in all local distributions. Only considering the local top-10 items is therefore not guaranteed to always identify the real global top-10. We show in chapter 12 that using PPTKS, it is possible to aggregate 180,000 distinct IP addresses within a few minutes.

11.3.1 Top-k protocol PPTK

First, we introduce the privacy-preserving top-k protocol *PPTK*, which is the central component of the final *PPTKS* protocol.

[1]**PPTKS** is short for **P**rivacy-**P**reserving **T**op-**k** using **S**ketches.

Input data

In the beginning, each input peer locally holds a set of *items*. An item is defined by an identifying key and a corresponding value. The goal of the protocol is to compute the k items with the biggest aggregate values over all input sets, without disclosing information about non-top-k items. The aggregate value of an item is simply the sum of its local values. Also, the protocol should not reveal which input peers contribute to a top-k item.

First, the input peers store keys and values of their items in one-dimensional hash arrays of size H. Hash values are generated using a public hash function h with $h(x) \in [0, \ldots, H-1]$. The precise choice of the array size H involves a tradeoff between accuracy and performance. The smaller H is chosen, the more collisions occur and the less accurate are the computations. The bigger H is chosen, the more MPC operations need to be performed on vectors of size H.

The input peers generate a vector of keys $\mathbf{k} = \{k_0, k_1, \cdots, k_{H-1}\}$ and values $\mathbf{v} = \{v_0, v_1, \cdots, v_{H-1}\}$, initialized to zero. Then, for each item, they store key and value at the position given by the hash of the key. If local collisions occur, the input peer only reports key and value of the item with the bigger value. That is, for each item a the input peers do:

$$\left.\begin{array}{r} v_{h(key(a))} = value(a) \\ k_{h(key(a))} = key(a) \end{array}\right\} \quad \text{if} \quad value(a) > v_{h(key(a))}$$

Note that we use hashes solely for storing items from a typically large and sparse space of keys in a more compact form, which makes private aggregation much more efficient as items with the same key fall into the same bucket. We do *not* rely on the privacy properties of hash functions. While it is generally hard to infer x from $h(x)$, it can be practically feasible for small domains. For instance, if x is an IPv4 address, it is relatively easy to brute-force the entire address range and find x from $h(x)$. Hence, in our protocol, the privacy of input values is solely protected by the secret sharing scheme.

Aggregation

Each input peer shares the vectors $\mathbf{k}$ and $\mathbf{v}$ among the privacy peers. Let $[k^i_j]$ and $[v^i_j]$ be the sharings at position j in the key and value vectors of input peer i, respectively. Then, the privacy peers aggregate the values for all items simply by building the sum over all n inputs:

$$[V_j] = [v_j^1] + [v_j^2] + \cdots + [v_j^n]$$

The value V_j is an approximation of the aggregate value of the largest item that is hashed at position j. The vector $\mathbf{V}$ contains the aggregated values for all positions. Local and global collisions introduce error. If local collisions occur, a node selects the bigger item. Thus, the dropped item loses support in the global aggregation and its value is underestimated. Global collisions occur if the keys of two input peers, say nodes 1 and 2, at position j differ, i.e., $k_j^1 \neq k_j^2$. We use a collision resolution algorithm (see Algorithm 2) to detect global collisions for the top-k items and to correct the introduced error. As we show in the evaluation section, these errors can be easily manipulated to become arbitrarily small.

Finding the top-k items

The privacy peers now hold sharings of all aggregate item values. The next step is to find the indices of the k biggest values in $[\mathbf{V}]$. The keys of these values represent the sought top-k items. The basic idea following [142] and [1] is to identify a threshold value τ that separates the k-th from the $(k+1)$-th item by performing a binary search over the range of values. For each candidate threshold, the number of values above the threshold are privately computed and compared to k. The threshold is increased if more than k items are larger, otherwise it is decreased. Once the correct threshold τ is found, the indices of all values greater than τ are returned. For sake of simplicity, we assume here that all values are different. We denote the maximum expected value by M. Then, the binary search for τ is guaranteed to finish in $\log_2 M$ rounds. If we have an estimate of τ based on past observations, we can, of course, reduce average search convergence time. For instance, in our implementation, we set the initial value of τ in line 5 to twice the value of the previous τ.

The detailed algorithm is given in Algorithm 1. Note that the algorithm does not reconstruct the number of values above/below intermediate threshold values, but only decides whether the threshold is too high or not. The algorithm requires $(H+1)\log_2 M$ invocations of $lessThan$ and $\log_2 M$ invocations of $equal$. The vast majority of these operations are independent and can be performed in parallel.

Resolving global collisions

Algorithm 1 Find indices of the top-k elements in a vector of sharings.

```
Require: A shared vector $[\mathbf{V}]$ with elements $[V_0]$ to $[V_{H-1}]$
 1: $match = 0$
 2: $lbound = 0$ {lower bound}
 3: $ubound = M$ {upper bound}
 4: while $match == 0$ do
 5:     $\tau = \lceil (lbound + ubound)/2 \rceil$
 6:     $[biggercount] = share(0)$
 7:     for $j = 0$ to $H - 1$ do
 8:         $[lt_j] = lessThan([V_j], \tau)$
 9:         $[biggercount] = [biggercount] + (1 - [lt_j])$
10:     end for
11:     $match = recon(equal([biggercount], k))$
12:     if $match == 0$ then
13:         $toohigh = recon(lessThan([biggercount], k))$
14:         if $toohigh == 1$ then
15:             $ubound = \tau$
16:         else
17:             $lbound = \tau$
18:         end if
19:     end if
20: end while
21: $count = 0$ {start item reconstruction}
22: for $j = 0$ to $H - 1$ do
23:     if $recon([lt_j]) == 0$ then
24:         $topi_{count} = j$
25:         $count = count + 1$
26:     end if
27: end for
28: return $(\mathbf{topi})$
```

Once we know the indices of the top k values, we resolve global collisions and reconstruct the top-k keys and values. To compute a top-k value $[V_j]$, we aggregated n local values from the input peers that correspond to up to n different keys. Which of these keys should be the key assigned to $[V_j]$? In this step we find the key *maxk* with the biggest contribution to the aggregate value $[V_j]$. We then compute the final item for index j by reconstructing *maxk* along with its aggregate value *maxv*.

Algorithm 2 gives the details of our global collision resolution protocol. The basic idea of the algorithm is to first compare all the potentially conflicting keys (line 3). Then, for each pair of keys, the conditional values $condleft_{xy}$ are computed. Given keys with index x and y, $condleft_{xy}$ holds the value associated with key x, if and only if keys x and y are equal. Otherwise, zero is stored. These conditional values are used in line 12 to compute the aggregate values of all items that share the same key. The final loop in lines 18-22 simply sweeps through the input sets of all input peers and keeps the maximum aggregate value and its key.

Algorithm 2 Reconstruct key with the biggest contribution to the aggregate value in position j (j is fixed for one run).

```
Require: Shared keys $[k_j^1], [k_j^2], \cdots, [k_j^n]$, and corresponding values $[v_j^1], [v_j^2], \cdots, [v_j^n]$.
 1: for $x = 1$ to $n$ do
 2:     for $y = x+1$ to $n$ do
 3:         $[e_{xy}] = equal([k_j^x], [k_j^y])$ {Note that $e_{xy} == e_{yx}$}
 4:         $[condleft_{xy}] = [e_{xy}] * [v_j^x]$
 5:         $[condleft_{yx}] = [e_{xy}] * [v_j^y]$
 6:     end for
 7: end for
 8: for $x = 1$ to $n$ do
 9:     $[aggrv_x] = [v_j^x]$
10:     for $y = 1$ to $n$ do
11:         if $y \neq x$ then
12:             $[aggrv_x] = [aggrv_x] + [condleft_{yx}]$
13:         end if
14:     end for
15: end for
16: $[maxv] = [aggrv_1]$ {store the max value}
17: $[maxk] = [k_j^1]$ {store key with max value}
18: for $x = 2$ to $n$ do
19:     $[comp] = lessThan([maxv], [aggrv_x])$
20:     $[maxv] = [comp] * [aggrv_x] + (1 - [comp]) * [maxv]$
21:     $[maxk] = [comp] * [k_j^x] + (1 - [comp]) * [maxk]$
22: end for
23: return $(maxk, maxv)$
```

The algorithm requires $n(n-1)/2$ invocations of *equal* and $n-1$ invocations of $lessThan$. All the computations of the intermediate values $[e_{xy}]$ in line 3 are independent and can be performed in parallel. The algorithm needs to be called once for each of the k top-k items. All of these calls are also independent and can be executed in parallel. Note that although the algorithm needs $O(n^2)$ invocations of *equal*, the number of input peers n will typically be small, i.e., much smaller than H or the number of items aggregated.

Accuracy

Depending on H and the item distribution, local and global hash collisions can introduce error. In this section, we use real traffic data to evaluate the accuracy of the top-k items found by PPTK. We ran simulations on TCP destination port and IP address distributions of the six biggest customer networks of SWITCH. In the scenario, the six customers want to compute the aggregate top-k items over their data sets. We used traffic from a whole day in August 2007, resulting in 96 timeslots of 15 minutes. For each timeslot, a new random seed for the hash function was used. We computed the correct set σ of

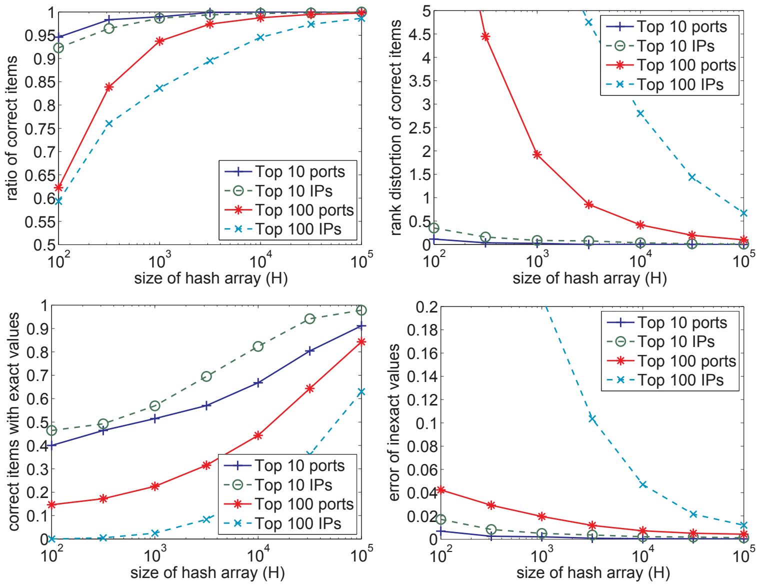


Figure 11.5: *Statistics for top 10/100 ports and top 10/100 IP addresses.*

aggregate top-k items and the approximate set $\tilde{\sigma}$ as calculated by our protocol. We then compared the two to evaluate the accuracy of PPTK using the following metrics:

1. Percentage of correct top-k items: $|\sigma \cap \tilde{\sigma}|/k$.
2. The average rank distortion of items in $\sigma \cap \tilde{\sigma}$, where the rank distortion of an item is the absolute difference between its true and approximate rank. For instance, if the rank 1 item is reported in rank 4, then its rank distortion is 3.
3. Percentage of items in $\sigma \cap \tilde{\sigma}$ with *exact* values.
4. Average relative error of values that are in $\sigma \cap \tilde{\sigma}$ but are not *exact*.

All metrics except the average rank distortion take values in the range $[0, 1]$. For metrics 1 and 3, a value of 1 is best whereas for metrics 2 and 4 a value of 0 is optimal.

With port distributions, the number of distinct items N is smaller or equal to 65,535. Figure 11.5 shows the accuracy metrics for the computation of top-10/100 ports and IP addresses. The size of the hash array H is varied between 100 and $100,000$. For top-10 ports, rather small choices of H produce very good results. For example, for $H = 1,000$, the ratio of correctly identified top-k items is 98.95% and the rank distortion of these items is only 0.023. 51.4% of the items have an exact values while the remaining have an average relative error of only 0.2%. This is surprising as H is very small compared to N in this case. If we raise H to $10,000$, the ratio of correct items goes to 1.

Accurately reconstructing the top-100 ports requires bigger choices of H. In a heavy-tailed distribution, the top-10 items are much more stable with values usually further apart from each other compared to items with ranks between 90 and 100. Therefore, the fluctuation in ranks 90 to 100 is much higher and small inaccuracies have more impact. Still, a choice of $H = 10,000$ reconstructs 98.7% of the top-100 ports correctly, with 41.7% exact values and only 0.7% error on the non-exact values. The rank distortion is expected to be higher than for top-10 computation, as the maximum distortion per item is 100 instead of 10. However, the average rank distortion of 0.41 in the top-100 is acceptable, especially as most of this distortion is due to the lower ranked items.

The number of distinct IP addresses in the aggregate distribution varies between 70,000 during the night and approximately 180,000 during the day. As with ports, relatively small values of H lead to very good results for the top-10 items. For $H = 10,000$, 99.7% of the top-10 IP addresses are reconstructed correctly. For those with non-exact values, the average error is only 0.2%. However, for the top-100 addresses, only $H = 31,600$ (or higher) leads to acceptable results, i.e., 97.3% correct items with 36.1% correct values and 2.1% error in the non-exact values.

11.3.2 Top-k protocol PPTKS

The running time of PPTK scales linearly with H, requiring $O(H)$ *lessThan* operations. Therefore, if we choose $H = 10,000$ instead of $H = 1,000$, we improve accuracy but also entail a 10-fold running time increase. For computing top-k reports, an alternative leading to lower error on top-k items is to use multiple smaller hash arrays of equal size instead of one large hash array. In this section we improve the accuracy of PPTK by using sketches, leading to a better tradeoff between accuracy and running time. Instead of a single hash array of size H, we use S hash arrays with pairwise independent hash functions. The advantage of using two or more hash arrays is that the probability of observing the same hash collisions in multiple arrays is very low. Therefore, multiple arrays allow the correction of errors introduced by hash collisions in a single array.

Composition of PPTK Instances

With PPTK, collisions always lead to an underestimation of item values. If a collision occurs locally, the input peer drops the smaller key, which leads to an underestimation of this key's value. If a collision occurs globally, our collision resolution protocol reconstructs only the key with the largest value. All other keys are dropped, again leading to an underestimation of their values. Therefore, the accuracy of values can be monotonically increased by adding more hash arrays and by always selecting the maximum value for each key. For example, if we have three arrays and the aggregate counts for port 80 are 15,176, 16,002, and 15,995, then we know that 16,002 is closest to the true count of port 80 and that for the other counts some contributions to port 80 were dropped due to collisions.

PPTKS, the sketch-version of PPTK, proceeds as follows:

1. Run PPTK S times using pairwise independent hash functions.
2. For each distinct key in the S top-k sets, store the maximum occurring value.
3. Sort the new items by descending value and return the largest k items.

PPTKS with $S = 1$ corresponds to PPTK.

Analysis of reconstructed information

Besides the final top-k items, PPTKS discloses some additional information. An item appearing in two or more of the S top-k lists may have different value estimates. The difference between two estimates indicates a contribution that was dropped. However, it does not reveal any other information, like the identity or number of input peers that reported the item.

Assume an item that is not part of the true top-k set but appears in one or more of the S approximate top-k sets. The item could only slip in one of the S top-k sets because a true top-k item was underestimated. This means,

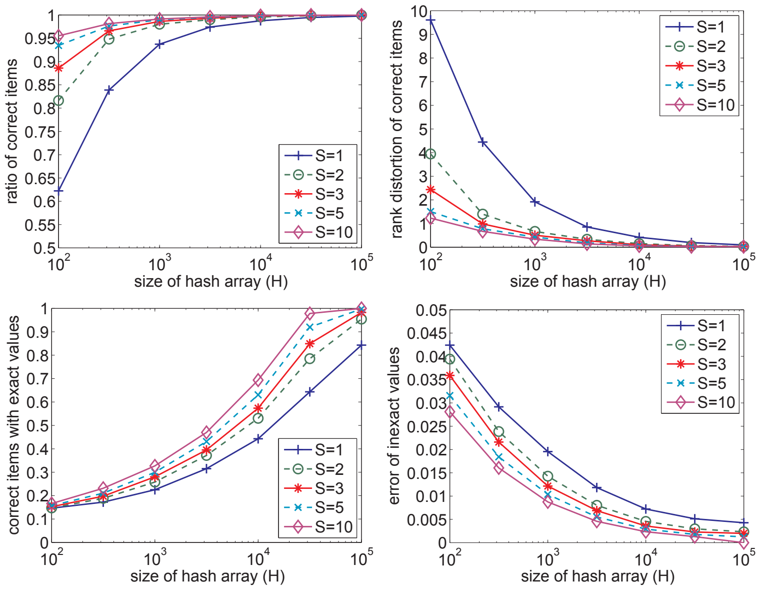


Figure 11.6: *Statistics for top 100 ports using sketches with S hash arrays.*

that the false-positive item for which we learn its approximate value must be directly following the true top-k items. If we compute the top-100 items and the ratio of correct items for each top-k set is 95%, then we learn (on average) information about the top-105 items. However, if we monitor the top-100 items continuously over time, chances are high that we will learn these values anyway due to fluctuations around rank 100. If this additional information is considered sensitive, the aggregation of the S sets can easily be performed in MPC as well, using a slightly modified version of Algorithm 2.

Accuracy

Figure 11.6 and Figure 11.7 illustrate the accuracy improvements achieved by applying PPTKS for top-100 ports and IP addresses, respectively. Generally, PPTKS improves accuracy significantly for small values of H, whereas the improvement is smaller for higher H. This is intuitively clear, because for higher values of H accuracy is already high and the room for further improvement is small. Another observation is that the first additional array, i.e.,

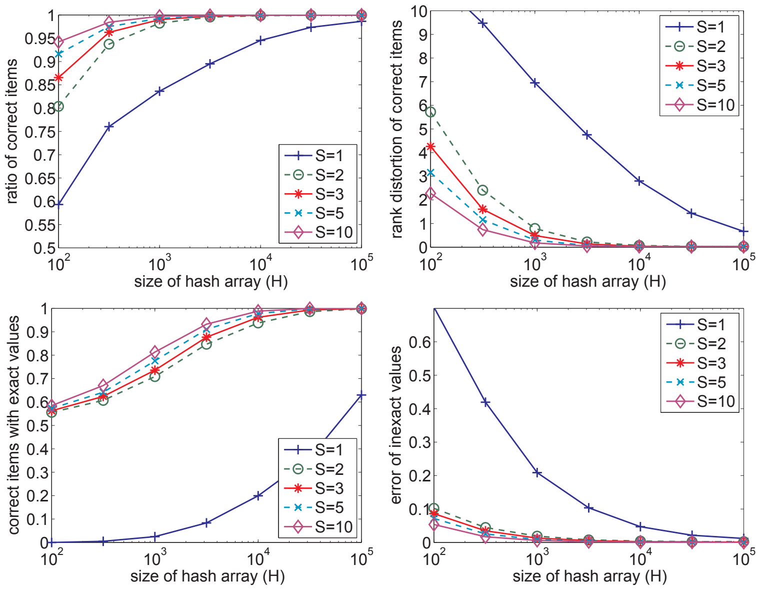


Figure 11.7: *Statistics for top 100 IP addresses using sketches with S hash arrays.*

increasing $S = 1$ to $S = 2$, improves accuracy significantly, while using $S > 3$ does not help as much. Particularly striking is the case of top-100 IP addresses shown in Figure 11.7. For $H = 1,000$, using $S = 2$ instead of $S = 1$ increases the ratio of correct items from 83.6% to 98.2%. At the same time, the rank distortion of correct items is greatly reduced from 7.0 to 0.8. Furthermore, while the ratio of exact values was as low as 2.5% for $S = 1$, it is boosted to 70.8% for $S = 2$. Last but not least, the average error of inexact values is reduced from 20.8% to 1.9%. To achieve a similar accuracy improvement with a single hash array, H would have to be raised to $100,000$, leading to 100 times longer running time. With $H = 1,000$ and 2 arrays, the same improvement is achieved by merely doubling running time.

11.4 Taxonomy of Applications

Rounding off this chapter, we now provide a high-level view on different use cases for privacy-preserving computations in the field of networking.

Providing organizations with incentives to share data in a privacy-preserving manner is a challenging problem, for it is typically safer for an organization to simply avoid sharing. However, as privacy preserving technologies become more mature and well-known, this is likely to change and to open new directions for multi-domain privacy-preserving analysis applications. We envision four distinct aggregation scenarios using SEPIA. The first scenario is aggregating information coming from multiple domains of one large (international) organization. This aggregation is presently not always possible due to privacy concerns and heterogeneous jurisdiction. The second scenario is analyzing private data owned by independent organizations with a mutual benefit in collaborating. Local ISPs, for example, might collaborate to detect common attacks. A third scenario provides access to researchers for evaluating and validating traffic analysis or event correlation prototypes over multi-domain network data. For example, national research, educational, and university networks could provide SEPIA input and/or privacy peers that allow analyzing local data according to submitted MPC scripts. Finally, one last scenario is the privacy-preserving analysis of end-user data, i.e., end-user workstations can use SEPIA to collaboratively analyze and cross-correlate local data.

Based on these scenarios, we see three different classes of possible SEPIA applications.

Network Security

Over the last years, considerable research efforts have focused on distributed data aggregation and correlation systems for the identification and mitigation of coordinated wide-scale attacks. In particular, aggregation enables the (early) detection and characterization of attacks spanning multiple domains using data from IDSes, firewalls, and other possible sources [7, 53, 89, 154, 156]. Recent studies [77] show that coordinated wide-scale attacks are prevalent: 20% of the studied malicious addresses and 40% of the IDS alerts are attributed to coordinated wide-scale attacks. Furthermore, strongly correlated groups profiting most from collaboration have less than 10 members and are stable over time, which is well-suited for SEPIA protocols.

In order to counter such attacks, Yegneswaran *et al.* [154] presented DOMINO, a distributed IDS that enables collaboration among nodes. They evaluated the performance of DOMINO with a large set of IDS logs from over 1600 providers. Their analysis demonstrates the significant benefit that is obtained by correlating the data from several distributed intrusion data sources. The major issue faced by such correlation systems is the lack of data privacy. In their work, Porras *et al.* survey existing defense mechanisms and propose several remaining research challenges [112]. Specifically, they point out the need for efficient privacy-preserving data mining algorithms that enable traffic classification, signature extraction, and propagation analysis.

Profiling and Performance Analysis

A second category of applications relates to traffic profiling and performance measurements. A global profile of traffic trends helps organizations to cross-correlate local traffic trends and identify changes. In [127] the authors estimate that 50 of the top-degree ASes together cover approximately 90% of global AS-paths. Hence, if large ASes collaborate, the computation of global Internet statistics is within reach. One possible statistic is the total traffic volume across a large number of networks. This statistic, for example, could have helped in the dot-com bubble in the late nineties, since the traffic growth rate was overestimated by a factor of 10, easing the flow of venture capital to Internet start-ups [104]. In addition, performance-related applications can benefit from an “on average” view across multiple domains. Data from multiple domains can also help to locate a remote outage with higher confidence, and to trigger proper detour mechanisms. A number of additional MPC applications related to performance monitoring are discussed in [121].

Research Validation

Many studies are obliged to avoid rigorous validation or have to re-use a small number of old traffic traces [41, 132]. This situation clearly undermines the reliability of the derived results. In this context, SEPIA can be used to establish a privacy-preserving infrastructure for research validation purposes. For example, researchers could provide MPC scripts to SEPIA nodes running at universities and research institutes.

Chapter 12

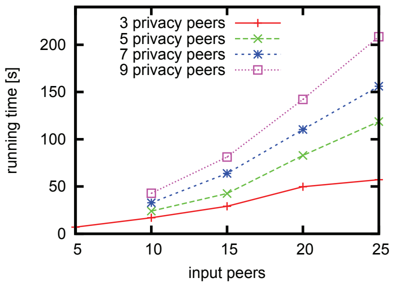
Performance Evaluation

For a moment, nothing happened.
Then, after a second or so,
nothing continued to happen.

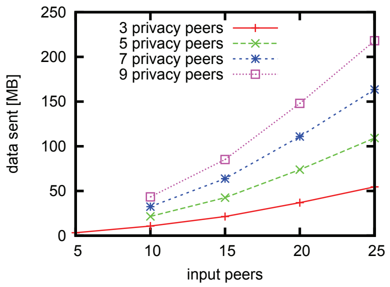
Douglas Adams

In this chapter we evaluate the resource requirements of our protocols in terms of running time and network bandwidth. We also explore the impact of running selected protocols on PlanetLab where hardware, network delay, and bandwidth are very heterogeneous.

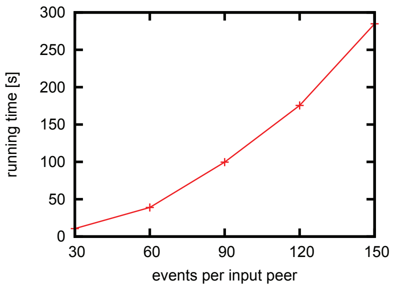
We assessed the CPU and network bandwidth requirements of our protocols, by running different aggregation tasks with real and simulated network data. For each protocol, we ran several experiments varying the most important parameters. We varied the number of input peers n between 5 and 25 and the number of privacy peers m between 3 and 9, with $m < n$. The experiments were conducted on a shared cluster comprised of several public workstations; each workstation was equipped with a 2x Pentium 4 CPU (3.2 GHz), 2 GB memory, and 100 Mb/s network. Each input and privacy peer was run on a *separate* host. In our plots, each data point reflects the average over 10 time windows. Background load due to user activity could not be totally avoided. section 12.2 discusses the impact of single slow hosts on the overall running time.



(a) Average round time ($s = 30$).



(b) Data sent per privacy peer ($s = 30$).



(c) Round time vs. s (n=10, m=3).

Figure 12.1: *Round statistics for event correlation with $T_c = n/2$. s is the number of events per input peer.*

12.1 Event Correlation

For the evaluation of the event correlation protocol, we generated artificial event data. It is important to note that our performance metrics do not depend on the actual values used in the computation, hence artificial data are just as good as real data for these purposes.

Running Time

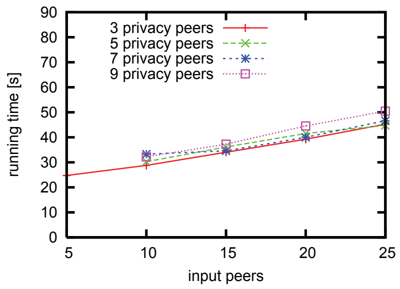
Figure 12.1 shows evaluation results for event correlation with $s = 30$ events per input peer, each with 24-bit keys for $T_c = n/2$. We ran the protocol including weight and key verification. Figure 12.1(a) shows that the average running time per time window always stays below 3.5 min and scales quadratically with n, as expected. Investigation of CPU statistics shows that with increasing n also the average CPU load per privacy peer grows. Thus, as long as CPUs are not used to capacity, local parallelization manages to compensate parts of the quadratical increase. With $T_c = n - const$, the running time as well as the number of operations scale linearly with n. Although the total communication cost grows quadratically with m, the running time dependence on m is rather linear, as long as the network is not saturated. The dependence on the number of events per input peer s is quadratic as expected without optimizations (see Figure 12.1(c)).

To study whether privacy peers spend most of their time waiting due to synchronization, we measured the user and system time of their hosts. All the privacy peers were constantly busy with average CPU loads between 120% and 200% (four cores in total) for the various operations.[1] Communication and computation between privacy peers is implemented using separate threads to minimize the impact of synchronization on the overall running time. Thus, SEPIA profits from multi-core machines. Average load decreases with increasing need for synchronization from multiplications to *equal*, over *lessThan* to event correlation. Nevertheless, even with event correlation, processors are very busy and not stalled by the network layer.

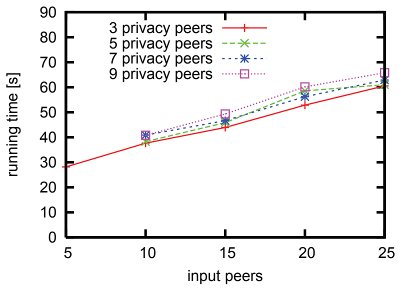
Bandwidth requirements

Besides running time, the communication overhead imposed on the network is an important performance measure. Since data volume is dominated by privacy peer messages, we show the average bytes sent *per privacy peer* in one time window in Figure 12.1(b). Similar to running time, data volume scales roughly quadratically with n and linearly with m. In addition to the transmitted data, each privacy peer receives about the same amount of data from the other input and private peers. If we assume a 5-minute clocking of the event correlation protocol, an average bandwidth between 0.4 Mbps (for $n = 5$, $m = 3$) and 13 Mbps (for $n = 25$, $m = 9$) is needed per privacy peer. Assuming a 5-minute interval and sufficient CPU/bandwidth resources, the maximum number of supported input peers before the system stops working in real-time ranges from around 30 up to roughly 100, depending on protocol parameters.

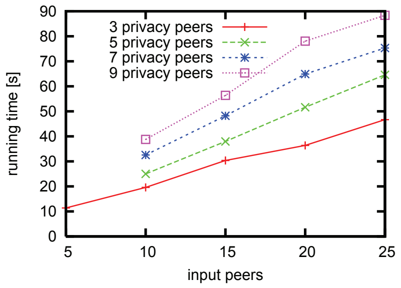
[1]When run on a 32-bit platform, up to twice the CPU load was observed, with similar overall running time. This difference is due to shares being stored in `long` variables, which are more efficiently processed on 64-bit CPUs.



(a) Addition of port histogram.



(b) Entropy of port distribution.



(c) Distinct AS count.

Figure 12.2: *Network traffic statistics: mean running time per time window versus n and m, measured on a department-wide cluster. All tasks were run with an input set size of 65k items.*

12.2 Network Traffic Statistics

For evaluating the network statistics protocols, we used unsampled NetFlow data captured from the border routers of the SWITCH network. We first extracted traffic flows belonging to different customers of SWITCH and assigned an independent input peer to each organization's trace. For each organization, we then generated SEPIA input files, where each input field contained either the values of volume metrics to be added or the local histogram of feature distributions for collaborative entropy (distinct count) calculation. In this section we focus on the running time and bandwidth requirements only. We performed the following tasks over ten 5-minute windows:

1. **Volume Metrics:** Adding 21 volume metrics containing flow, packet, and byte counts, both total and separately filtered by protocol (TCP, UDP, ICMP) and direction (incoming, outgoing). For example, Figure 13.1 on page 144 plots the total and local number of incoming UDP flows of six organizations for an 11-day period.
2. **Port Histogram:** Adding the full destination port histogram for incoming UDP flows. SEPIA input files contained 65,535 fields, each indicating the number of flows observed to the corresponding port. These local histograms were aggregated using the addition protocol.
3. **Port Entropy:** Computing the Tsallis entropy of destination ports for incoming UDP flows. The local SEPIA input files contained the same information as for histogram aggregation. The Tsallis exponent q was set to 2.
4. **Distinct count of AS numbers:** Aggregating the count of distinct source AS numbers in incoming UDP traffic. The input files contained 65,535 columns, each denoting if the corresponding source AS number was observed. For this setting, we reduced the field size p to 31 bits because the expected size of intermediate values is much smaller than for the other tasks.

Running Time

For task 1, the average running time was below 1.6 s per time window for all configurations, even with 25 input and 9 privacy peers. This confirms that addition-only is very efficient for low volume input data. Figure 12.2 summarizes the running time for tasks 2 to 4. The plots show on the y-axes the average running time per time window versus the number of input peers on the x-axes. In all cases, the running time for processing one time window was below 1.5 minutes. The running time clearly scales linearly with n. Assuming a 5-minute interval, we can estimate by extrapolation the maximum number of supported input peers before the system stops working in real-time. For the conservative case with 9 privacy peers, the supported number of input peers is approximately 140 for histogram addition, 110 for entropy computation, and 75 for distinct count computation. We observe, that for single round protocols (addition and entropy), the number of privacy peers has only little impact on the running time. For the distinct count protocol, the running time increases linearly with both n and m. Note that the shortest running time for distinct count is even lower than for histogram addition. This is due to the reduced field size (p with 31 bits instead of 62), which reduces both CPU and network load.

Bandwidth Requirements

For all tasks, the data volume sent per privacy peer scales perfectly linear with n and m. Therefore, we only report the maximum volume per time window with 25 input and 9 privacy peers. For addition of volume metrics, the data volume is 141 KB and increases to 4.7 MB for histogram addition. Entropy computation requires 8.5 MB and finally the multi-round distinct count requires 50.5 MB. For distinct count, to transfer the total of $2 \cdot 50.5 = 101$ MB within 5 minutes, an average bandwidth of roughly 2.7 Mbps is needed per privacy peer.

Internet-wide Experiments

In our evaluation setting hosts have homogeneous CPUs, network bandwidth and low round trip times (RTT). In practice, however, SEPIA's goal is to aggregate traffic from remote network domains, possibly resulting in a much more heterogeneous setting. For instance, *high delay* and *low bandwidth* directly affect the waiting time for messages. Once data have arrived, the *CPU model and clock rate* determine how fast the data are processed and can be distributed for the next round.

Recall from chapter 11 that each operation and protocol in SEPIA is designed in rounds. Communication and computation during each round run in parallel. But before the next round can start, the privacy peers have to synchronize intermediate results and therefore wait for the slowest privacy peer to finish. The overall running time of SEPIA protocols is thus affected by the slowest CPU, the highest delay, and the lowest bandwidth rather than by the average performance of hosts and links. Therefore we were interested to see whether the performance of our protocols breaks down if we take it out of the homogeneous LAN setting. Hence, we ran SEPIA on PlanetLab [111] and repeated task 4 (distinct AS count) with 10 input and 5 privacy peers on globally distributed PlanetLab nodes. Table 12.1 compares the LAN setup with two PlanetLab setups A and B.

RTT was much higher and average bandwidth much lower on PlanetLab. The only difference between PlanetLab A and B was the choice of some nodes with slower CPUs. Despite the very heterogeneous and globally distributed setting, the distinct count protocol performed well, at least in PlanetLab A.

	LAN	PlanetLab A	PlanetLab B
Max. RTT	1 ms	320 ms	320 ms
Bandwidth	100 Mb/s	$\geq$ 100 Kb/s	$\geq$ 100 Kb/s
Slowest CPU	2 cores 3.2 GHz	2 cores 2.4 GHz	1 core 1.8 GHz
Running time	25.0 s	36.8 s	110.4 s

Table 12.1: *Comparison of LAN and PlanetLab settings.*

Most importantly, it still met our near real-time requirements. From PlanetLab A to B, running time went up by a factor of 3. However, this can largely be explained by the slower CPUs. The distinct count protocol consists of parallel multiplications, which make efficient use of the CPU and local addition, which is solely CPU-bound. Let us assume, for simplicity, that clock rates translate directly into MIPS. Then, computational power in PlanetLab B is roughly 2.7 times lower than in PlanetLab A. Of course, the more rounds a protocol has, the bigger is the impact of RTT. But in each round, the network delay is only a constant offset and can be amortized over the number of parallel operations performed per round. For many operations, CPU and bandwidth are the real bottlenecks.

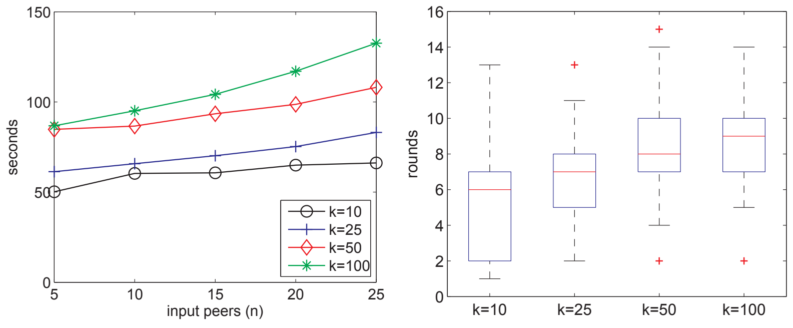
While aggregation in a heterogeneous environment is possible, SEPIA *privacy peers* should ideally be deployed on dedicated hardware, to reduce background load, and with similar CPU equipment, so that no single host slows down the entire process.

12.3 Top-k Queries

In this section, we report results on the actual running time of PPTKS with varying protocol parameters.

Setup

Computations were run on 2x Dual Core AMD Opteron 275 machines with 1Gb/s LAN connections. Each privacy peer was run on a separate host such that all communication between them crossed the network. We computed distributed top-k reports for real port and IP address distributions. The input data were again taken from the six biggest customers of the SWITCH network,



(a) Mean total running time.

(b) Rounds needed for binary search. Mean time per binary search round is 9.1s.

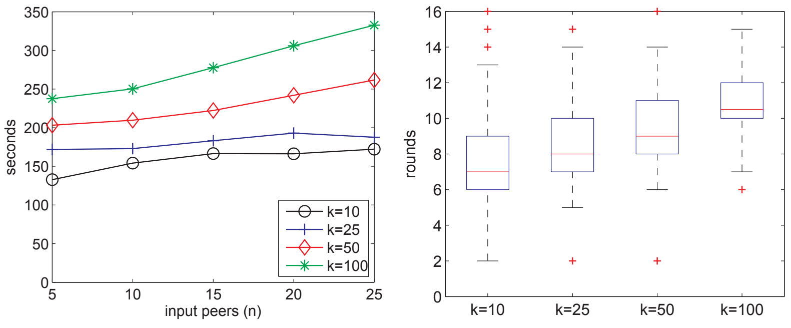
Figure 12.3: *Running time statistics for top-k* port *reports ($m = 5$, $l = 24$ bits).*

as in section 11.3.1. For configurations with more than six input peers, we reused input data in a round-robin way. We chose $H = 1,000$, $S = 2$ to guarantee high accuracy even for $k = 100$ and used $m = 5$ privacy peers. We varied the number of input peers n between 5 and 25, and k between 10 and 100.

Discussion

Figure 12.3 and Figure 12.4 show the running time statistics for top port and IP address reports, respectively. Each combination of n and k was run 50 times. On the left, the mean running time of PPTKS for each configuration is shown in seconds. For port reports, the running time varies between 50.2 and 132.6 seconds, whereas IP address reports take between 132.8 and 332.7 seconds. The main reason port reports are faster is the smaller field size p. It was sufficient to use a p with 24 bits to represent the port values and intermediate aggregate counts for each port. For IP address reports, we needed a p with 33 bits to represent IP addresses (the keys) as shared secrets. Recall from chapter 9 that the number of distributed multiplications required for $equal$s and $lessThan$s scales linearly with $l = \log_2 p$. As a result, a single round of binary search takes only 9.1s for ports and twice as long for IP addresses.

The binary search phase contributes most to the running time of PPTKS and introduces a significant variance. On the right side (Figure 12.3(b) and



(a) Mean total running time.

(b) Rounds needed for binary search. Mean time per binary search round is 18.3s.

Figure 12.4: *Running time statistics for top-k* IP address *reports ($m = 5$, $l =$ 33 bits).*

Figure 12.4(b)), we show the distribution of the number of binary search rounds necessary to find the correct τ, separating the k-th from the $(k+1)$-th value. In each box, the central mark represents the median. The edges of the box are the 25th (Q_1) and 75th (Q_3) percentiles and the whiskers extend to the most extreme data points not considered outliers. Outliers are points larger than $Q_3 + 1.5(Q_3 - Q_1)$ or smaller than $Q_1 - 1.5(Q_3 - Q_1)$ and are plotted individually. With increasing k, more binary search rounds are needed. This can be explained by the heavy-tailed nature of port and IP address distributions. For instance, with IP address reports, the gap size between items ranked 10 and 11 is in the order of 1000-2000. That is, the binary search terminates as soon as τ falls somewhere in this gap. For items ranked 100 and 101, the gap size is only about 10-30 and τ needs to be much more precise. Consequently, more rounds are needed for $k = 100$ than for $k = 10$. Nevertheless, variation is quite big. While for $k = 10$ in Figure 12.3(b) the median of rounds is 6, the difference between Q_3 and Q_1 is 5 rounds. The running time variation of 5 binary search rounds corresponds to 45s which is almost as much as the mean total running time. Increasing the number of hash arrays S will increase the mean number of binary search rounds and reduce the variability at the same time, because for the binary search to finish, all the individual binary searches in each hash array have to finish.

The running time increases with n due to the collision resolution phase that compares the keys of all input peers for each bin and also selects the maximum of all the n contributed keys in the end.

There is still room for performance improvements. Firstly, if k is smaller than 100, smaller choices of H already lead to very high precision reports. For instance, for top-10 ports, $H = 316$ and $S = 2$ yield 99.9% correct items (see section 11.3.1). This choice reduces overall running time by a factor of 3. Secondly, if it is guaranteed that all input keys and values as well as the aggregate values are smaller than $p/2$, the number of multiplications for $lessThan$s could be cut in half (see section 9.2). Thirdly, comparing secrets to small public values (e.g., $k = 10$, or small values of τ) can be done more efficiently by using the *shortRange* operation. This could further reduce running time of the binary search phase for small values of k and item distributions with rather low item counts.

Chapter 13

Collaborative Network Troubleshooting in Practice

All for one, and one for all!

The Three Musketeers

In this chapter, we finally apply the developed collaborative protocols to real traffic traces and analyze how they are useful in troubleshooting traffic anomalies, exemplified by the 2007 Skype outage.

The Skype outage in August 2007 started from a Windows update triggering a large number of system restarts. In response, Skype nodes scanned cached host-lists to find supernodes causing a huge distributed scanning event lasting two days [120, 140]. We used NetFlow traces of the actual up- and downstream traffic of the 17 biggest customers of the SWITCH network. The traces span 11 days from the 11th to 22nd and include the Skype outage (on the 16th/17th) along with other smaller anomalies. We ran SEPIA's vector addition protocol and the top-k protocol PPTKS on these traces and investigated how the organizations can benefit by correlating their local view with the aggregate view.

We first computed per-organization and aggregate time series of the UDP flow count metric and applied a simple detector to identify anomalies. For each time series, we used the first 4 days to learn its mean μ and standard deviation σ, defined the normal region to be within $\mu \pm 3\sigma$, and detected anoma-

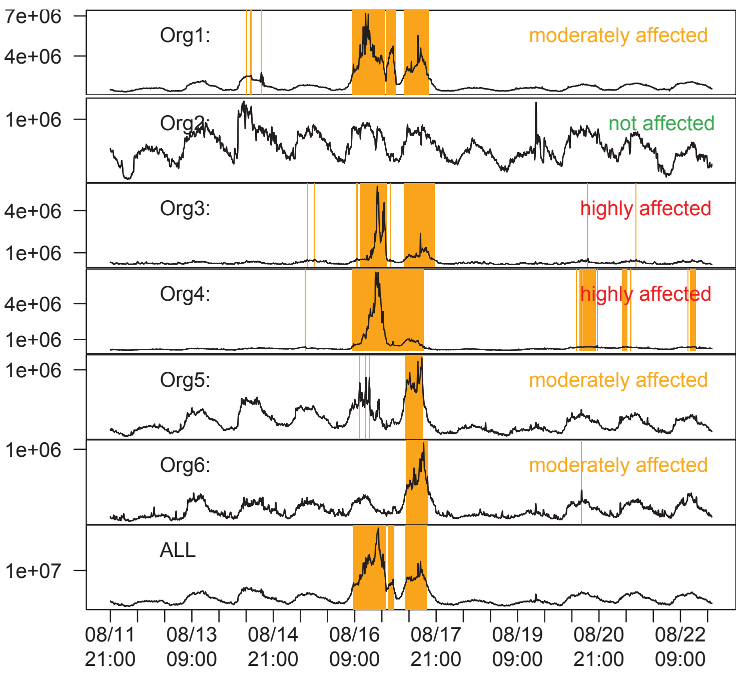


Figure 13.1: *Flow count in 5' windows with anomalies for the biggest organizations and aggregate view (ALL). Each organization sees its local and the aggregate traffic. The 2007 Skype anomaly is visible in the middle.*

lous time intervals. In Figure 13.1 we illustrate the local time series for the six largest organizations and the aggregate time series. We rank organizations based on their decreasing average number of daily flows and use their rank to identify them. In the figure, we also mark the detected anomalous intervals. Observe that in addition to the Skype outage, some organizations detect other smaller anomalies that took place during the 11-day period.

13.1 Anomaly Correlation

Using the aggregate view, an organization can find if a local anomaly is the result of a global event that may affect multiple organizations. Knowing the global or local nature of an anomaly is important for steering further trou-

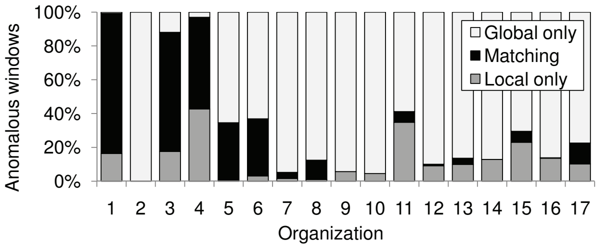


Figure 13.2: *Correlation of local and global anomalies for organizations ordered by size (#1 is the biggest).*

bleshooting steps. Therefore, we first investigate how the local and global anomalous intervals correlate. For each organization, we compared the local and aggregate anomalous intervals and measured the total time an anomaly was present: 1) only in the local view, 2) only in the aggregate view, and 3) both in the local and aggregate views, i.e., the *matching anomalous intervals*. Figure 13.2 illustrates the corresponding time fractions. We observe a rather small fraction, i.e., on average 14.1%, of local-only anomalies. Such anomalies lead administrators to search for local targeted attacks, misconfigured or compromised internal systems, misbehaving users, etc. In addition, we observe an average of 20.3% matching anomalous windows. Knowing an anomaly is both local and global steers an affected organization to search for possible problems in popular services, in widely-used software, like Skype in this case, or in the upstream providers. A large fraction (65.6%) of anomalous windows is only visible in the global view. In addition, we observe significant variability in the patterns of different organizations. In general, larger organizations tend to have a larger fraction of matching anomalies, as they contribute more to the aggregate view. While some organizations are highly correlated with the global view, e.g., organization 3 that notably contributes only 7.4% of the total traffic, other organizations are barely correlated, e.g., organizations 9 and 12. Organization 2 has no local anomalies at all.

13.2 Relative Anomaly Size

Not all organizations are affected equally strongly. We define *relative anomaly size* to be the ratio of the detection metric value during an anoma-

Org #	3	5	6	7	13	17
lag [hours]	1.2	2.7	23.4	15.5	4.8	3.6

Table 13.1: *Organizations profiting from an early anomaly warning by aggregation.*

lous interval over the detection threshold. Organizations 3 and 4 had relative anomaly sizes 11.7 and 18.8, which is significantly higher than the average of 2.6. Using the average statistic, organizations can compare the relative impact of an attack. Organization 2, for instance, had anomaly size 0 and concludes that there was a large anomaly taking place but they were not affected. Most of the organizations conclude that they were indeed affected, but less than average. Organizations 3 and 4, however, have to spend thoughts on why the anomaly was so disproportionately strong in their networks.

13.3 Early-warning

An interesting question is whether organization could use this type of information to build an early-warning system for global or large-scale anomalies. The Skype anomaly did not start concurrently in all locations, since the Windows update policy and reboot times were different across organizations. We measured the lag between the time the Skype anomaly was first observed in the aggregate and local view of each organization. In Table 13.1 we list the organizations that had considerable lag, i.e., above an hour. Notably, one of the most affected organizations (6) could have learned the anomaly almost one day ahead. However, as shown in Figure 13.2, for organization 2 this would have been a false positive alarm. To profit most from such an early warning system in practice, the aggregate view should be annotated with additional information, such as the number of organizations or the type of services affected from the same anomaly. In this context, our event correlation protocol is useful to decide whether similar anomaly signatures are observed in the participating networks. Anomaly signatures can be extracted automatically using actively researched techniques [21, 114].

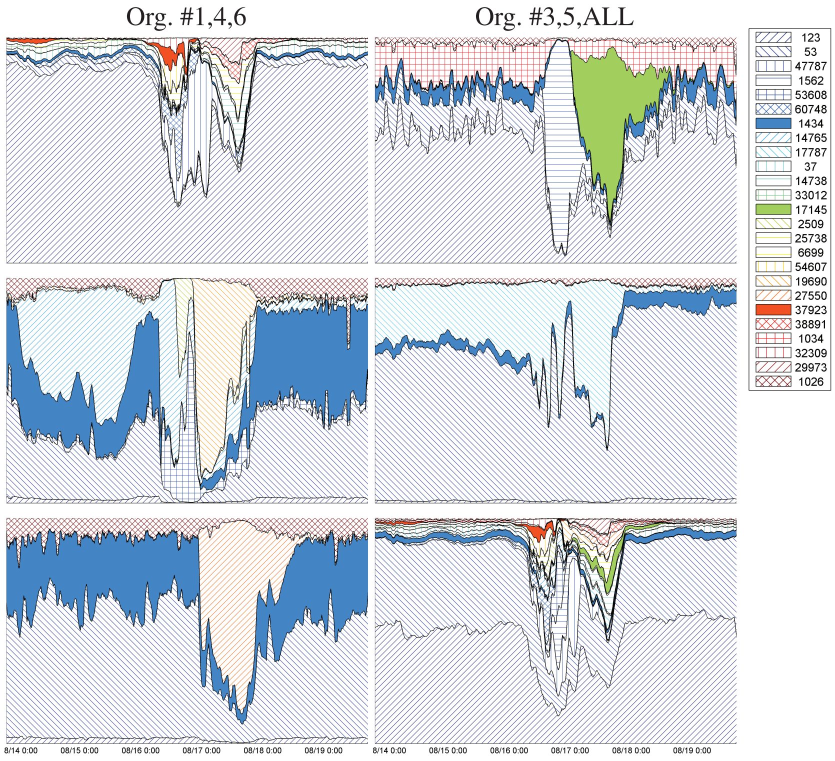


Figure 13.3: *Global top-25 incoming UDP destination ports and their local visibility 6 days around the 2007 Skype anomaly.*

13.4 Anomaly Troubleshooting

So far, organizations have learned that some global anomaly is going on. However, to actually troubleshoot the anomaly, more detailed information is needed.

We applied PPTKS to the six biggest customers' UDP traffic and show top-k statistics for incoming destination ports in Figure 13.3, and outgoing destination IP addresses in Figure 13.4. The plots show the share of each port (IP address) of the total traffic in terms of flow count. Statistics are shown for affected organizations #1, and #3-6. The aggregate statistics are shown on the

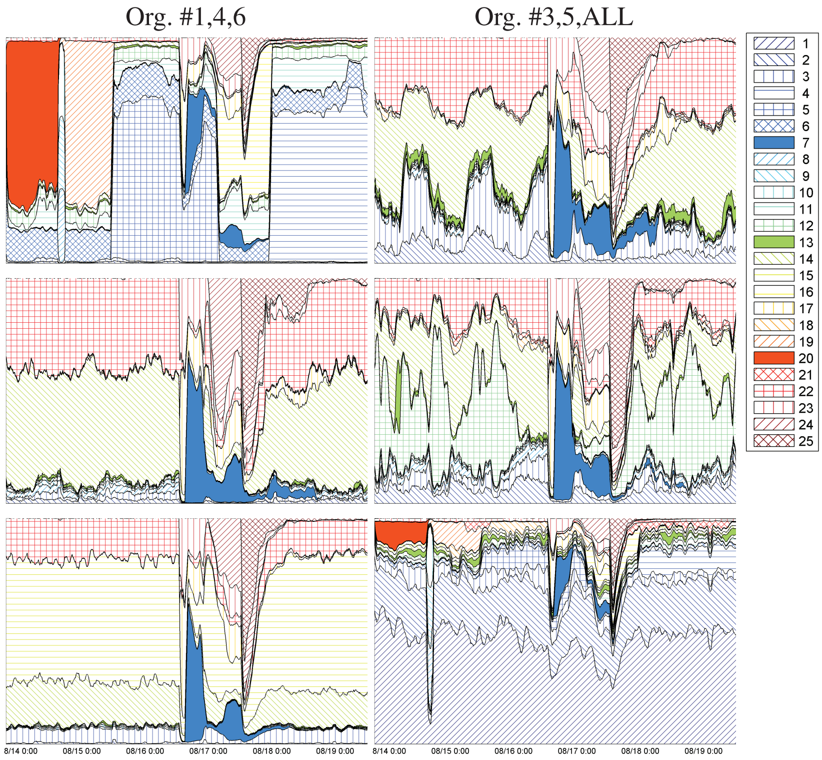


Figure 13.4: *Global top-25 outgoing UDP destination IP addresses and their local visibility 6 days around the 2007 Skype anomaly.*

bottom right (ALL). The covered period spans 6 days around the Skype outage. Before the anomaly, the traffic mix is dominated by NTP (port 123) and DNS (port 53). Also, ports 1434 and 1026 have significant support across several organizations. Port 1434 is associated with Microsoft SQL Monitor and the Slammer worm, which is still trying to spread. Port 1026 is presumably used for attempted spamming of the Windows Messenger service.

When the anomaly starts, organizations see a sudden increase in activity on specific high port numbers. Connections also originate mainly from a series of dynamic ports. Some of the scanned high ports are extremely prevalent, e.g., destination port 19690 accounts for 93% of all flows of organization

#4, at the peak rate (see Figure 13.3). Investigation of the traffic shows that most of the anomalous flows within organizations #3 and #4 are targeted at a single IP address and originate from thousands of distinct source addresses connecting repeatedly up to 13 times per minute. These patterns indicate that the two organizations host popular supernodes, attracting a lot of traffic to specific ports. Other organizations mainly host client nodes and see uniform scanning, while organization #2 (not shown) has banned Skype completely.

Interestingly, as can be seen in Figure 13.3, each organization is affected on distinct dynamic destination ports that are not shared with other organizations. For instance, organization #3 is affected on ports 1562 and 17145, whereas #4 is affected on port 19690 and #6 on port 27550. Each organization can conclude that their anomalous port is not shared with others from the aggregate plot. By using the absolute flow count numbers in Figure 13.1, they are able to calculate the absolute flows for each port in the aggregate plot. From these numbers they can conclude that they are the main contributors to the counts of their anomalous ports. Furthermore, they learn that other organizations are affected in a similar way, because the aggregate plot reveals an increase in a number of different dynamic ports similar to their own local anomaly.

Unlike anomalous ports, which are unique in each organization, Figure 13.4 reveals that there is a subset of anomalous external IP addresses[1] that all organizations have in common. In particular, addresses 7, and 23-25 start being active at the start of the anomaly and cause a major part of the anomalous traffic in each organization. Organizations can deduce from the aggregate plot that these external hosts are not unique to their own network. This gives a strong indication that all organizations indeed see parts of *a single global anomaly* instead of unrelated local events.

Based on these types of collaborative analyses, organizations can easily determine the scope and learn details of distributed anomalies, identify its probable root causes and take appropriate measures to mitigate damage. Also, local anomalies can be identified as such, by learning that other organizations are not affected.

[1] IP addresses were replaced by identifiers for privacy reasons.

Chapter 14

Related Work on Privacy-Preserving Technologies

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

In this chapter we review related work on privacy-preserving data analysis. In the fist part, we give an overview over secure multiparty computation (MPC). We discuss theoretic results and summarize applications. In the second part, we briefly touch techniques based on randomization and data perturbation. We then discuss related work for the privacy-preserving top-k problem and approaches that use specific system architectures to analyze data in a privacy-preserving way.

14.1 Secure Multiparty Computation

MPC is a cryptographic framework that allows a set of players to evaluate arbitrary functionality on their private data sets, without the need for a trusted third-party. The foundations of MPC were built in [153], introducing the famous millionaires' problem in which two millionaires want to find which of them is richer without disclosing their amount of wealth to each other. Goldreich et al. [68] and Chaum et al. [38] generalized the approach and provided strong feasibility results. In particular, they show that any distributed protocol can be carried out with cryptographic security as long as a majority of players is honest. Assuming perfectly secure channels, [13] and [37] proposed protocols that are unconditionally secure against a minority of semi-honest adversaries and a 1/3-minority of malicious adversaries. The important issue of composability of cryptographic functions was addressed in [34] by proposing the UC-framework for security proofs of distributed protocols.

Although it is theoretically possible to evaluate any computable function using MPC, it is far from trivial to build solutions that are practical in terms of computation and communication costs. After the initial theoretic results, 26 years passed until the first real-world application of MPC took place. Only in 2008, a legally binding sugar-beet auction with more than one thousand bidders was performed in Denmark [17].

Homomorphic Encryption

With homomorphic encryption schemes, also known as privacy homomorphisms (PHs), it is possible to perform privacy-preserving computations on encrypted values without learning secret input data. An encryption algorithm is called *homomorphic* if performing a specific algebraic operation in the ciphertext domain is equivalent to performing a (possibly different) algebraic operation in the plaintext domain (typically addition or multiplication). For instance, given cleartexts a, b and encryption/decryption functions $\mathcal{E}$ and $\mathcal{D}$. Then the operation $\oplus$ is additively homomorphic if the following holds:

$$a+b=\mathcal{D}(\mathcal{E}(a)\oplus\mathcal{E}(b))$$

Among the first works on PHs were Rivest et al. [119]. More recently, several other schemes have been proposed, including the ones by Benaloh [14], Paillier [107], or ElGamal [57]. For a comprehensive survey, please refer to [62].

A disadvantage of homomorphic encryption schemes is that basic operations are usually very expensive. Depending on the scheme, encryption operations have to be performed on large numbers with 1024 or more bits. Furthermore, the above schemes are only *partially* homomorphic. That is, they support only one type of homomorphism (either additive or multiplicative).

Only very recently, a fully homomorphic (but less efficient) scheme supporting addition *and* multiplication was proposed by Gentry [67].

Even though fully homomorphic encryption (FHE) is dubbed by some the "Holy Grail" of cryptography, its applicability is limited to two-party scenarios, also known as *secure function evaluation* (SFE). Using FHE, it is possible to outsource private computations of a single client to an external entity, e.g., a cloud provider. However, more general approaches are needed for implementing *multi-client* private computations [144].

Secret Sharing

The most efficient way of doing MPC is using secret sharing over small fields. These approaches typically use field sizes of 32 or 64 bit, which means that arithmetic operations on shares can be performed by CPU instructions directly. The simplest sharing scheme is *additive* secret sharing (e.g., [6]). A secret value is split up into random values that sum up to the secret value. Each random value is then distributed to a different player. This scheme allows very efficient private addition, but it does not naturally support multiplications. Also, all the shares have to be returned in order to reconstruct a secret value. If a single player is missing, reconstruction is impossible.

Shamir's secret sharing scheme [129] uses polynomials to represent secrets and evaluation points as shares. It allows the implementation of a (n,t)-threshold scheme, in which any group of t or more players can reconstruct the secret but any group of $t-1$ players has no information about it. For more details on Shamir's scheme, please refer to section 8.1.

Applications

MPC has been applied to various problem in privacy-preserving data mining [91, 145]. Usually, these solutions apply relatively expensive primitives based on homomorphic encryption and Yao's solution to the millionaires' problem for comparing values. They typically focus on efficiency in terms of algorithm complexity and do not evaluate the performance with implementations.

Specific problems addressed include computing the k-th ranked element of the union of data sets [1], decision tree learning using ID3 [90], association rule mining, and EM clustering [42]. Privacy-preserving set operations, such as set union, intersection, and element count, have also attracted quite some attention [42, 64, 123].

Roughan et al. [122] first proposed the use of MPC techniques (based on efficient secret sharing) for a number of applications relating to traffic measurements, including the estimation of global traffic volume and performance measurements [121]. In addition, the authors identified that MPC techniques can be combined with commonly-used traffic analysis methods and tools, such as time-series algorithms [6] and sketch data structures. However, they neither considered comparison operations nor implemented their ideas.

Implementations

Only recently, the community has started to implement frameworks for performing MPC. FairplayMP [12] uses an approach based on garbled circuits [10] to evaluate any program in a constant number of synchronization rounds. FairplayMP is closed-source and is implemented in Java. It comes with a high-level language called SFDL, which allows to easily declare MPC programs.

VIFF [48] is a comprehensive framework supporting Shamir shares and homomorphic encryption. It is the only currently available framework that deals with malicious adversaries and implements asynchronous MPC. It is implemented in Python and released under an LGPL license.

Sharemind [16] uses additive secret sharing as a basis and implements multiplication and comparison operations on top of it. Unfortunately, it is restricted to 3 computation nodes only. Sharemind's implementation language is C++ and its source code is available "on request".

Regarding performance, Sharemind and SEPIA clearly outperform the other frameworks. For a detailed comparison, please refer to section 9.3.

14.2 Data Sanitization and Randomization

Data correlation systems that provide strong privacy guarantees for the participants often achieve data privacy by means of (partial) data sanitization based on bloom filters [135] or cryptographic functions [85, 89]. Also, randomized data perturbation techniques are often used in privacy-preserving data mining [2, 145]. However, it has been shown that these techniques often preserve less privacy than expected due to predictable structures in the spectral domain [75]. In general, sanitization and randomization are not lossless processes and therefore impose the same unavoidable tradeoff between data privacy and utility discussed in detail in Part I of this thesis.

14.3 Architectural Approaches

Chow et al. [39] and Applebaum et al. [5] avoid the privacy-utility tradeoff by means of cryptographic data obfuscation. Chow et al. proposed a two-party query computation model to perform privacy-preserving querying of distributed databases. In addition to the databases, their solution comprises three entities: the randomizer, the computing engine, and the query frontend. Local answers to queries are randomized by each database and the aggregate results are de-randomized at the frontend. Applebaum et al. present a semi-centralized solution for the collaboration among a large number of participants in which responsibility is divided between a proxy and a central database. In a first step the proxy obliviously blinds the clients' input, consisting of a set of keyword/value pairs, and stores the blinded keywords along with the non-blinded values in the central database. On request, the database identifies the (blinded) keywords that have values satisfying some evaluation function and forwards the matching rows to the proxy, which then unblinds the respective keywords. Finally, the database publishes its non-blinded data for these keywords. As opposed to these approaches, SEPIA does not depend on two central entities but in general supports an arbitrary number of distributed privacy peers, is provably secure, and more flexible with respect to the functions that can be executed on the input data.

14.4 Privacy-Preserving Top-k Queries

A lot of work [3, 36, 58, 59, 81, 96] has focused on algorithms for answering distributed one-time top-*k* queries in the non-privacy-preserving setting, where the main goal is to minimize communication cost. In particular, these works assume a set of parties holding local lists of items and corresponding values. A top-*k* query then finds the items with the top-*k* aggregate values using typically exact methods. For example, with the well-known algorithm by Fagin [58], parties output local items in a descending order until the output has *k* items in common. The candidate items in the output are then guaranteed to include the top-*k*. On the other hand, a number of related works, such as [8], adopt the data streaming model, where each party locally observes an online stream of items and corresponding values, and use approximate methods to answer top-k monitoring queries, which continuously report the k largest values obtained from distributed data streams.

Most related to our work, [142,143] study the problem of finding the top-k matching items over vertically-partitioned private data, where each party holds different item attributes. They use privacy-preserving data mining and MPC techniques to extend Fagin's algorithm [58]. Although Vaidya's algorithm can solve our problem in principle, the number x of disclosed candidate items can be very large if item sets are disjoint to some degree. The assumption made in [142] is that all items are reported by all sites. However, this is not the case with our type of data, i.e., port and IP address distributions. In our experiments, on average only 114 out of 124,000 IP addresses are common in all 6 sites. Therefore, for finding the top say 120 IP addresses, Vaidya's algorithm would disclose the complete set of IP addresses ($x = 124,000$), since the output never has 120 items in common. Furthermore, also the running time of the algorithm scales with x. The authors recently implemented their algorithm [143] with Fairplay [95], which for $x = 10,000$ requires more than 10 hours for a two-party computation.

[151] introduced an algorithm for finding the k largest values among parties holding private sets of values. The protocol preserves privacy in a probabilistic way by randomizing values before distributing them among parties and avoids using cryptographic primitives. In contrast, our work focuses on the more challenging problem of aggregating items and finding the top-k aggregate items.

Our approach 1) answers distributed one-time top-k queries probabilistically, 2) uses sketches to enable very efficient MPC protocols and to accurately estimate the top-k items, and 3) provides strong privacy guarantees.

14.5 Differential Privacy

The goal of MPC is to specify a function $f()$ that is evaluated on the distributed data sets *as if* a TTP was available. MPC guarantees that no information is leaked *from the computation process*. That is, it solves the problem of *how* to compute $f()$ in a privacy-preserving way. An orthogonal problem is to find out *what* is safe to compute. Just learning the resulting value $f()$ could allow the inference of sensitive information. For example, if the private input bits must remain secret, computing the logical AND of all input bits is insecure in itself: if the final result was 1, all input bits must be 1 as well and are thus no longer secret. In MPC, *it is the responsibility of the input providers to verify that learning $f()$ is acceptable*, in the same way as they have to verify this when using a real TTP.

One approach to do this is *differential privacy* [56, 98], which systematically randomizes answers to database queries to prevent inference of sensitive input data. It guarantees that from answers to queries, it is statistically impossible to infer the presence or absence of single records in the database, e.g., medical records about a specific person. Differential privacy and MPC complement each other very well. Using differential privacy, it is possible to specify a randomized output $\widetilde{f}()$ that is safe for public release. Using MPC, it is possible to actually compute $\widetilde{f}()$ in a privacy-preserving manner, without relying on a TTP. Intuitively, the stronger $f()$ aggregates input data, the less randomness needs to be added [54]. As with anonymization, the utility of $\widetilde{f}()$ compared to $f()$ has to be evaluated for specific use cases.

An important detail to be aware of is that differential privacy assumes input data records to be independent. Therefore, application of differential privacy to traffic data requires additional consideration of inter-packet (flow) dependencies. Also, the protection goal is typically to protect privacy of users, hosts, and networks and not of individual packets or flows. Dominant behavior in user profiles is often determined by hundreds or thousands of such flows. It is unlikely that the absence or presence of single flows impacts such profiles noticeably. Therefore, differential privacy in network data requires modeling at a higher level.

Chapter 15

Conclusions

The future is here. It's just not widely distributed yet.

William Gibson

The Internet has become an important pillar of today's economy and needs to be protected against increasingly professionalized cybercrime. Collaborative approaches based on distributed Internet measurements are crucial to ensure understanding of the constantly growing Internet and to stand the pace of attack sophistication in network security. Unfortunately, privacy concerns hinder such collaborative approaches. From a legal perspective, network data are sensitive personal data and can not easily be shared. But internal security policies and competitive environments also hinder collaborative approaches in network security and monitoring.

In the first part of this thesis, we explored the delicate privacy-utility tradeoff involved in anonymization, the state-of-the-art solution for stripping sensitive information from network data. We showed how anonymization degrades the utility of traffic data for statistical network anomaly detection. At the same time, all anonymization techniques preserving a notion of hosts are susceptible to traffic injection attacks. These attacks allow the injection of arbitrary traffic patterns into network traffic and, in turn, the de-anonymization of data. These attacks are hard to detect and difficult to prevent. Hence, the privacy protection delivered by anonymization in practice is much weaker than expected. If the notion of hosts is abandoned, for instance by truncating bits of

IP addresses, host anonymity can be increased at the cost of further decreasing data utility. As a notable exception, entropy metrics achieve a well-balanced privacy-utility tradeoff for the detection of scans and (D)DoS attacks.

To escape the ill-fated privacy-utility tradeoff, we proposed a radically different approach in Part II of this thesis. We proposed to use secure multiparty computation (MPC) techniques for aggregating private collections of network traffic data. Although requirements for CPU and network resources are substantial with MPC protocols, they achieve strong cryptographic privacy guarantees and allow working with undistorted data. We showed how basic MPC primitives can be optimized to meet the challenges of analyzing voluminous network data in near real-time. We designed, implemented, and evaluated a number of privacy-preserving protocols for collaborative network security and monitoring. In particular, we developed protocols for distributed anomaly detection based on volume-metrics, feature entropies, and distinct element counts. Moreover, we developed protocols for correlating arbitrary events across multiple domains and for generating distributed top-k reports. We evaluated the resource requirements of our protocols and showed that they operate in near real-time. Using traffic from 17 SWITCH customers, we evaluated the usefulness of our protocols for anomaly troubleshooting in practice. As exemplified by the Skype outage in 2007, these collaborative approaches allow individual networks to detect and characterize distributed anomalies. Specifically, root-cause identification is simplified by comparing local views with the aggregate view.

Implementations of our optimized primitives and the collaborative protocols are made available in the SEPIA library under an LGPL license. SEPIA has attracted attention in the community and is used in DEMONS, an FP7 project of the European Union. The goal of DEMONS is to develop a framework for decentralized, cooperative, and privacy-preserving network monitoring. Moreover, the Swiss Commission for Technology and Innovation (CTI) has funded the PPFA project with the goal of integrating SEPIA with TNPFA, IBM's network flow analyzer appliance. In particular, the PPFA project aims at building a commercial multi-domain privacy-preserving network flow analyzer.

On the whole, we think the power of network data anonymization is overrated. Anonymization protects against accidental disclosure, may assist in compliance, and provides evidence of good faith. However, it cannot resist determined attempts to break it. For this reason, it must be integrated with legal, social, and technical means to achieve the aims of better data sharing for research and operations. In contrast to anonymization, MPC-based solutions provide proven privacy guarantees that facilitate alignment with legal compliance. Furthermore, they do not require distortion of input data. Certainly, MPC protocols impose a substantial running time overhead compared to non-private protocols. Yet, the range of networking problems having efficient MPC solutions is wide and will continue to grow in the future with optimized algorithms and primitives.

15.1 Critical Assessment

The evaluation of the privacy-utility tradeoff in anonymization is by nature dependent on a specific application scenario and attacker capabilities. We chose to evaluate the utility of anonymized data with regard to network anomaly detection due to the importance of this field both for network security and monitoring. Other applications of network data might result in a different tradeoff. For instance, if accounting is the main application, truncation of IP addresses might preserve address prefixes up to the required length and preserve complete utility. Our evaluation of privacy is based on an attacker capable of injecting traffic into networks. While assuming a strong attacker enables getting to the bottom of a problem, privacy might actually be better in practice. For the evaluation of IP address truncation, we rely on an average host anonymity metric. In that case, individual hosts might be less but also more exposed than the average.

In the design of SEPIA, we make the assumption that adversarial privacy peers are *semi-honest*. Recently, efficient solutions for dealing with malicious adversaries have been proposed [11,48], requiring only linear communication complexity. In order to make our protocols secure against malicious adversaries, the only thing needed is to adapt the implementation of the addition and multiplication operations. The design of the protocols is independent of the adversary model.

Our privacy-preserving protocols guarantee that no information is leaked from the computation process. However, releasing the final computation result might be delicate in itself. SEPIA delegates the responsibility for verifying the safety of releasing the final result to the participants. Intuitively, the more entities that participate and the stronger data are aggregated, the better privacy is protected.

15.2 Future Work

We believe that our work is only a first step paving the way for many applications of MPC in network security and monitoring. There are many directions for future work, including optimization of the SEPIA library, assessing the sensitivity of output data, and harmonizing legal frameworks for data sharing. However, the most interesting and multifaceted venue of future work is to explore the actual benefits of collaborative approaches over single-domain solutions for many specific applications in network security and monitoring.

Refining the SEPIA library

A general way of improving performance for all protocols is to optimize the basic primitives, such as multiplications, comparisons, or random number sharing. We see some potential in optimizing the `lessThan` comparison, especially by improving the way shared bits of random numbers are generated. During runtime, performance measurements and intelligent scheduling could assist in choosing appropriate primitives. For instance, the choice for implementations with constant rounds or minimum number of multiplications could be taken dynamically at runtime. Similarly, `lessThan` and `shortRange` operations could be chosen dynamically to optimize performance. Moreover, caching of secret intermediate predicates between subsequent comparison operations allows a substantial speedup of certain protocol instances. Regarding usability, a high-level scripting language would allow abstraction from the library API and facilitate use for first-time users.

Sensitivity of Output

As discussed above, safety guarantees for releasing final computation results are missing in SEPIA. Recent approaches, such as differential privacy, try to address this problem (see section 14.5). However, we think that application of these techniques to network traffic data needs further investigation, addressing data record dependencies and modeling privacy at the proper level of abstraction, i.e., for users, hosts, and networks.

Exploring the Power of Collaboration

Now that the means for preserving data privacy are available, collaborative approaches can be developed, deployed, and evaluated with real data. The focus should be on exploring the benefit of collaborating compared to isolated solutions. The proposed methodology provides a generic way for transforming non-private algorithms into privacy-preserving algorithms. Nevertheless, in certain cases, manual fine-tuning or acceptance of probabilistic results may be required to obtain real-world practicality of MPC solutions.

In the introduction and section 11.4 we allude to many problems that would profit from collaborative approaches:

- aggregation of multi-domain, regional, and global Internet statistics
- multi-domain performance measurements and traffic engineering
- detection of traffic discrimination
- anomaly/intrusion detection and root-cause analysis
- botnet detection
- malware signature generation and propagation analysis
- active defense against ongoing cyberattacks
- recommendation systems (e.g., collaborative black-listing)
- building a “virtual”, aggregate, and diverse reference data set for validating measurement studies

While we investigated the use of MPC for networking applications, the methodology is generic and, in principle, applicable to other fields, such as the aggregation and correlation of sensitive medical or financial data.

An intricate problem is to motivate participants to contribute their own data and to do this faithfully. In general, if privacy of input data is sufficiently protected and participants are interested in correct results themselves, this should not be a problem. However, in certain cases, participants may try to cheat the system in order to gain advantages or delude others by providing wrong or manipulated input data.[1] To limit the impact of “wrong” input data, we incorporated the privacy-preserving verification of input data in the event correlation protocol (see section 11.1). However, static verification rules do not provide complete protection. More rigorous frameworks, such as game theory, are needed to analyze the payoffs of different strategies in multi-player settings [74].

[1]Note that considering malicious instead of semi-honest adversaries does not solve the problem. Solutions secure against malicious adversaries only guarantee that players act consistently and do not change values they are committed to *during the process of a computation*. Detecting whether the players submit *correct* data in the first place is an orthogonal (and very difficult) problem.

Scalability

The most expensive operations in SEPIA are comparison operations, followed by multiplications. If an application scenario does not require these operations, the much simpler additive sharing scheme can be used. As we showed in [116], this scheme allows to divide computations into a preparatory offline and an online phase. During the online phase, private summation is possible with zero communication overhead. This allows massive-scale implementations of privacy-preserving functionality based on addition, such as certain set operations or anonymous publishing. More research into which applications are feasible with addition-only functionality would further broaden the range of applications with efficient MPC solutions.

Another direction of research is to leverage cloud computing for MPC computations. Although privacy in cloud computing is usually synonymous with homomorphic encryption, this approach does not really allow true multi-client settings [144]. A more flexible architecture would be to deploy SEPIA privacy peers in different clouds. The lightweight, and potentially numerous, input peers could be deployed on standard PCs or even mobile devices.

Legislation

Last but not least, data protection legislation for network traffic needs to be revised and homogenized across national boundaries. In particular, it should specify more precisely the requirements for privacy protection measures and what types of analyses are legitimate. Simply stating that data need to be "anonymized" somehow is not sufficient. Moreover, explicit exceptions for research are desirable. We believe that MPC with its proven privacy guarantees lends itself to facilitate legal compliance in the future. Considering that legislation often lags behind technological development in the Internet ("code is law"), a next step could be to standardize privacy-preserving methods and back them with reference implementations.

15.3 Publications

The work presented in this thesis is based on the following publications:

P01: **On the Utility of Anonymized Flow Traces for Anomaly Detection**
Martin Burkhart, Daniela Brauckhoff, Martin May
ITC Specialist Seminar on Network Usage and Traffic (ITC SS), 2008.

P02: **The Risk-Utility Tradeoff for IP Address Truncation**
Martin Burkhart, Daniela Brauckhoff, Martin May, Elisa Boschi
ACM Workshop on Network Data Anonymization (NDA), 2008.

P03: **The Role of Network Trace Anonymization under Attack**
Martin Burkhart, Dominik Schatzmann, Brian Trammell, Elisa Boschi, Bernhard Plattner
ACM SIGCOMM Computer Communication Review (CCR), 40(1), 2010.

P04: **SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics**
Martin Burkhart, Mario Strasser, Dilip Many, Xenofontas Dimitropoulos
USENIX Security Symposium, 2010.

P05: **Fast Privacy-Preserving Top-k Queries using Secret Sharing**
Martin Burkhart, Xenofontas Dimitropoulos
IEEE International Conference on Computer Communications and Networks (ICCCN), 2010.

P06: **Privacy-Preserving Distributed Network Troubleshooting – Bridging the Gap between Theory and Practice**
Martin Burkhart, Xenofontas Dimitropoulos
(under submission)

In addition, these publications were coauthored during this thesis:

P07: **Beyond Shannon: Characterizing Internet Traffic with Generalized Entropy Metrics**
Bernhard Tellenbach, Martin Burkhart, Didier Sornette, Thomas Maillart
Passive and Active Measurement Conference (PAM), 2009.

P08: **Inferring Spammers in the Network Core**
Dominik Schatzmann, Martin Burkhart, Thrasyvoulos Spyropoulos
Passive and Active Measurement Conference (PAM), 2009.

P09: **Accurate Network Anomaly Classification with Generalized Entropy Metrics**
Bernhard Tellenbach, Martin Burkhart, Dominik Schatzmann, David Gugelmann, Didier Sornette
(under submission)

P10: **Peeling Away Timing Error in NetFlow Data**
Brian Trammell, Bernhard Tellenbach, Dominik Schatzmann, and Martin Burkhart
Passive and Active Measurement Conference (PAM), 2011.

P11: **Reduce to the Max: A Simple Approach for Massive-Scale Privacy-Preserving Collaborative Network Measurements**
Fabio Ricciato, Martin Burkhart
Workshop on Traffic Monitoring and Analysis (TMA), 2011.

Bibliography

[1] G. Aggarval, N. Mishra, and B. Pinkas. Secure Computation of the kth-Ranked Element. In *EUROCRYPT*, 2004.

[2] C. C. Aggarwal and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer Publishing Company, 2008.

[3] R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top-k queries. In *International conference on Very large data bases (VLDB)*, 2007.

[4] M. Alllman and V. Paxson. Issues and etiquette concerning use of shared measurement data. In *Internet Measurement Conference (IMC)*, 2007.

[5] B. Applebaum, H. Ringberg, M. J. Freedman, M. Caesar, and J. Rexford. Collaborative, privacy-preserving data aggregation at scale. In *Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[6] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *Proc. ACM WPES'04*, October 2004.

[7] ATLAS. Active Threat Level Analysis System. `http://atlas.arbor.net`.

[8] B. Babcock and C. Olston. Distributed top-k monitoring. In *ACM SIGMOD international conference on Management of data*, 2003.

[9] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *ACM Symposium on Principles of distributed computing (PODC)*, 1989.

[10] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *ACM symposium on Theory of computing (STOC)*, 1990.

[11] Z. Beerliová-Trubıniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography Conference (TCC)*, 2008.

[12] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *Conference on Computer and communications security (CCS)*, 2008.

[13] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM symposium on Theory of computing (STOC)*, 1988.

[14] J. Benaloh. Dense probabilistic encryption. In *Workshop on Selected Areas of Cryptography*, 1994.

[15] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In *14th USENIX Security Symposium*, 2005.

[16] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *European Symposium on Research in Computer Security (ESORICS)*, 2008.

[17] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography*, 2009.

[18] E. Boschi. Legal requirements and issues in network traffic data protection. In *ACM Workshop on Network Data Anonymization (NDA)*, 2008.

[19] E. Boschi and B. Trammell. IP Flow Anonymization Support. *Internet RFCs, ISSN 2070-1721*, RFC 6235, 2011.

[20] A. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[21] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly extraction in backbone networks using association rules. In *Internet Measurement Conference (IMC)*, 2009.

[22] D. Brauckhoff, K. Salamatian, and M. May. Applying PCA for Traffic Anomaly Detection: Problems and Solutions. In *INFOCOM*, 2009.

[23] T. Brekne and A. Årnes. Circumventing IP-address pseudonymization. In *IASTED International Conference on Communications and Computer Networks*, 2005.

[24] 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.

[25] J. Brickell and V. Shmatikov. The cost of privacy: Destruction of data-mining utility in anonymized data publishing. In *ACM conference on Knowledge discovery and data mining (SIGKDD)*. ACM, 2008.

[26] 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)*, Berlin, Germany, Oct. 2008.

[27] M. Burkhart, D. Brauckhoff, M. May, and E. Boschi. The risk-utility tradeoff for IP address truncation. In *ACM Workshop on Network Data Anonymization (NDA)*, Oct. 2008.

[28] M. Burkhart and X. Dimitropoulos. Privacy-Preserving Distributed Network Troubleshooting – Bridging the Gap between Theory and Practice. *(under submission)*.

[29] M. Burkhart and X. Dimitropoulos. Fast privacy-preserving top-k queries using secret sharing. In *International Conference on Computer Communications and Networks (ICCCN)*, 2010.

[30] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Plattner. The role of network trace anonymization under attack. *Computer Communication Review (CCR)*, 40(1):5–11, 2010.

[31] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *19th USENIX Security Symposium*, August 2010.

[32] A. Burstein. An Uneasy Relationship: Cyber Security Information Sharing, Communications Privacy, and the Boundaries of the Firm. In *Workshop on the Economics of Information Security (WEIS)*, 2007.

[33] S. Cabuk, C. E. Brodley, and C. Shields. IP covert timing channels: design and detection. In *ACM conference on Computer and communications security (CCS)*, 2004.

[34] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.

[35] J. Carr. *Inside cyber warfare*. Mapping the cyber underworld. O'Reilly Media, Inc., 2009.

[36] K. Chang and S. Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *ACM SIGMOD international conference on Management of data*, 2002.

[37] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *ACM Symposium on Theory of Computing (STOC)*, 1988.

[38] D. Chaum, I. Damgård, and J. v. d. Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, 1987.

[39] S. S. M. Chow, J.-H. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *NDSS*. The Internet Society, 2009.

[40] Cisco Systems Inc. *NetFlow Services Solutions Guide*. available at `http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.pdf`.

[41] K. Claffy, M. Crovella, T. Friedman, C. Shannon, and N. Spring. Community-Oriented Network Measurement Infrastructure (CONMI) Workshop Report. *Computer Communication Review (CCR)*, 36(2):41, 2006.

[42] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):28–34, 2002.

[43] S. Coull, F. Monrose, M. Reiter, and M. Bailey. The challenges of effectively anonymizing network data. In *Cybersecurity Applications & Technology Conference For Homeland Security (CATCH)*, 2009.

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

[45] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *Network and Distributed System Security Symposium (NDSS)*, 2007.

[46] CSI Computer Crime and Security Survey 2009. `http://gocsi.com/survey`.

[47] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference (TCC)*, 2006.

[48] I. Damgård, M. Geisler, M. Krøigaard, and J. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Conference on Practice and Theory in Public Key Cryptography (PKC)*, 2009.

[49] I. Damgård, S. Meldgaard, and J. B. Nielsen. Perfectly Secure Oblivious RAM Without Random Oracles. In *Theory of Cryptography Conference (TCC)*, 2011.

[50] D. Dietrich. Bogons and bogon filtering. In *33rd meeting of the North American Network Operator's Group (NANOG 33)*, Feb. 2005.

[51] Directive 95/46/EC of the European Parliament and of the Council. OJ L 281, 23.11.1995, p. 31, October 1995.

[52] Directive 2002/58/EC of the European parliament and of the council. OJ L 201, 31.07.2002, p. 37, July 2002.

[53] DShield. The Internet Storm Center. `www.dshield.org`.

[54] Y. Duan. Differential privacy for sum queries without external noise. In *ACM Conference on Information and Knowledge Management (CIKM)*, 2009.

[55] G. T. Duncan, S. A. Keller-McNulty, and S. L. Stokes. Disclosure Risk vs. Data Utility: The R-U Confidentiality Map. Technical Report 121, National Institute of Statistical Sciences, December 2001.

[56] C. Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation (TAMC)*, 2008.

[57] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 2002.

[58] R. Fagin. Combining fuzzy information from multiple systems. In *ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1996.

[59] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, 2001.

[60] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon. Prefix-preserving IP address anonymization. *Comput. Networks*, 46(2):253–272, 2004.

[61] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[62] C. Fontaine and F. Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:15, 2007.

[63] M. Foukarakis, D. Antoniades, S. Antonatos, and E. Markatos. Flexible and high-performance anonymization of NetFlow records using anontool. In *SECURECOMM Conference*, 2007.

[64] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[65] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. On 2-round secure multiparty computation. In *CRYPTO*, 2002.

[66] 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.

[67] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM symposium on Theory of Computing (STOC)*. ACM, 2009.

[68] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM symposium on Theory of computing (STOC)*, 1987.

[69] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt, and L. Rizzo. The CoMo White Paper. *Intel Research Cambridge, Tech. Rep. IRCTR-04-017, September*, 2004.

[70] Initiative. Unlocking the power of mobile. `http://www.watblog.com/wp-content/uploads/2010/06/Initiative_MobileReport_Final_EMAIL.pdf` (visited Nov. 2010), 2010.

[71] ISC. Internet host count history. `http://www.isc.org/solutions/survey/history/` (visited March 2011).

[72] Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, 2010.

[73] Computer Language Benchmarks Game. `http://shootout.alioth.debian.org/u64q/java.php` (visited Feb. 2011).

[74] H. Kargupta, K. Das, and K. Liu. Multi-party, privacy-preserving distributed data mining using a game theoretic framework. In *Knowledge Discovery in Databases (PKDD)*, 2007.

[75] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *International Conference on Data Mining (ICDM)*, 2003.

[76] A. Karr, C. Kohnen, A. Oganian, J. Reiter, and A. Sanil. A framework for evaluating the utility of data altered to protect confidentiality. *The American Statistician*, 60(3):224–232, 2006.

[77] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating against common enemies. In *Internet Measurement Conference (IMC)*, 2005.

[78] kc claffy. A Day in the Life of the Internet: Proposed community-wide experiment. *ACM SIGCOMM Computer Communications Review*, 36(5):39–40, Oct. 2006.

[79] J. King, K. Lakkaraju, and A. Slagell. A taxonomy and adversarial model for attacks against network log anonymization. In *ACM symposium on Applied Computing (SAC)*, 2009.

[80] D. Koukis, S. Antonatos, and K. G. Anagnostakis. On the privacy risks of publishing anonymized IP network traces. In *Communications and Multimedia Security (CMS)*, 2006.

[81] F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. In *ACM symposium on Parallel algorithms and architectures (SPAA)*, 2007.

[82] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *ACM SIGCOMM*, Portland, August 2004.

[83] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, 2005.

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

[85] A. J. Lee, P. Tabriz, and N. Borisov. A privacy-preserving interdomain audit framework. In *Workshop on privacy in electronic society (WPES)*, 2006.

[86] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *International Conference on Data Engineering (ICDE)*, 2007.

[87] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Internet Measurement Conference (IMC)*, 2006.

[88] Y. Li, A. Slagell, K. Luo, and W. Yurcik. Canine: A combined conversion and anonymization tool for processing netflows for security. In *International Conference on Telecommunication Systems Modeling and Analysis*, 2005.

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

[90] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of cryptology*, 15(3):177–206, 2008.

[91] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

[92] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[93] S. Machiraju and R. H. Katz. Verifying global invariants in multi-provider distributed systems. In *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. ACM, 2004.

[94] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Internet measurement conference (IMC)*, 2009.

[95] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay-a secure two-party computation system. In *13th USENIX Security Symposium*, 2004.

[96] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[97] P. Matray, I. Csabai, P. Haga, J. Steger, L. Dobos, and G. Vattay. Building a prototype for network measurement virtual observatory. In *ACM Workshop on Mining network data (MineNet)*, 2007.

[98] F. McSherry and R. Mahajan. Differentially-private network trace analysis. In *ACM SIGCOMM*, 2010.

[99] G. Minshall. Tcpdpriv. `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`.

[100] J. Mirkovic. Privacy-safe network trace sharing via secure queries. In *ACM Workshop on Network Data Anonymization (NDA)*, 2008.

[101] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov. BotGrep: Finding P2P Bots with Structured Graph Analysis. In *19th USENIX Security Symposium*, 2010.

[102] Nfdump. `http://nfdump.sourceforge.net/`.

[103] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Conference on Theory and Practice of Public Key Cryptography (PKC)*, 2007.

[104] A. Odlyzko. Internet traffic growth: Sources and implications. *Optical transmission systems and equipment for WDM networking II*, 5247:1–15, 2003.

[105] P. Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *57 UCLA Law Review*, 2010. Available at `http://ssrn.com/abstract=1450006`.

[106] P. Ohm, D. Sicker, and D. Grunwald. Legal issues surrounding monitoring during network research (invited paper). In *Internet Measurement Conference (IMC)*, 2007.

[107] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[108] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *Computer Communication Review (CCR)*, 36(1):29–38, 2006.

[109] J. J. Parekh, K. Wang, and S. J. Stolfo. Privacy-preserving payload-based correlation for accurate malicious traffic detection. In *ACM Workshop on Large-scale Attack Defense (LSAD)*, 2006.

[110] M. Pietrzyk, J. Costeux, G. Urvoy-Keller, and T. En-Najjary. Challenging statistical classification for operational usage: the ADSL case. In *Internet measurement conference (IMC)*, 2009.

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

[112] P. Porras and V. Shmatikov. Large-scale collection and sanitization of network security data: risks and challenges. In *Workshop on New security paradigms (NSPW)*, 2006.

[113] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). *Internet RFCs, ISSN 2070-1721*, RFC 3917, 2004.

[114] S. Ranjan, S. Shah, A. Nucci, M. M. Munafò, R. L. Cruz, and S. M. Muthukrishnan. Dowitcher: Effective worm detection and containment in the internet core. In *INFOCOM*, 2007.

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

[116] F. Ricciato and M. Burkhart. Reduce to the Max: A Simple Approach for Massive-Scale Privacy-Preserving Collaborative Network Measurements. In *International Workshop on Traffic Monitoring and Analysis (TMA)*, 2011.

[117] H. Ringberg. *Privacy-Preserving Collaborative Anomaly Detection*. PhD thesis, Princeton University, 2009.

[118] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of PCA for traffic anomaly detection. In *ACM SIGMETRICS*, 2007.

[119] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations on Secure Computation*, 1978.

[120] D. Rossi, M. Mellia, and M. Meo. Understanding Skype Signaling. *Computer Networks*, 53(2):130–140, 2009.

[121] M. Roughan and Y. Zhang. Privacy-preserving performance measurements. In *SIGCOMM workshop on Mining network data (MineNet)*, 2006.

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

[123] Y. Sang, H. Shen, Y. Tan, and N. Xiong. Efficient protocols for privacy preserving matching against distributed datasets. In *Information and Communications Security (ICICS)*, 2006.

[124] D. Sauter. Invasion of Privacy Using Fingerprinting Attacks. Master Thesis MA-2008-22, ETH Zurich, 2009.

[125] J. G. Saxe. Blind Men and the Elephant. available at `http://wordinfo.info/unit/1/ip:3/il:B`.

[126] D. Schatzmann. Analyzing network traffic from the SWITCH network from 2003 until today. Master Thesis MA-2007-12, ETH Zurich, 2007.

[127] V. Sekar, Y. Xie, D. Maltz, M. Reiter, and H. Zhang. Toward a framework for internet forensic analysis. In *ACM HotNets-III*, 2004.

[128] SEPIA web page. `http://www.sepia.ee.ethz.ch`.

[129] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[130] V. Shmatikov and M. Wang. Security against probe-response attacks in collaborative intrusion detection. In *ACM Workshop on Large-scale Attack Defense (LSAD)*, 2007.

[131] A. Slagell, K. Lakkaraju, and K. Luo. FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs. In *USENIX Large Installation System Administration Conference (LISA)*, 2006.

[132] A. Slagell and W. Yurcik. Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization. In *Workshop on the Value of Security through Collaboration (SECOVAL)*, September 2005.

[133] A. Soule, H. Larsen, F. Silveira, J. Rexford, and C. Diot. Detectability of traffic anomalies in two adjacent networks. In *Passive And Active Measurement Conference (PAM)*, 2007.

[134] A. Soule, K. Salamatian, and N. Taft. Combining filtering and statistical methods for anomaly detection. In *Internet Measurement Conference (IMC)*, 2005.

[135] S. J. Stolfo. Worm and attack early warning. *IEEE Security and Privacy*, 2(3):73–75, 2004.

[136] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[137] SWITCH. The Swiss Education and Research Network. `http://www.switch.ch`.

[138] M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar. Detecting network neutrality violations with causal inference. In *Conference on Emerging networking experiments and technologies (CoNEXT)*, 2009.

[139] B. Tellenbach, M. Burkhart, D. Sornette, and T. Maillart. Beyond Shannon: Characterizing Internet Traffic with Generalized Entropy Metrics. In *Passive and Active Measurement Conference (PAM)*, 2009.

[140] B. Trammell and D. Schatzmann. A tale of two outages: a study of the Skype network in distress. In *2nd International Workshop on Traffic Analysis and Classification (TRAC)*, Istanbul, Turkey, 2011.

[141] UCRchive. Network measurement data and tools. `http://networks.cs.ucr.edu/ucrchive/measurement.htm` (visited Feb. 2011).

[142] J. Vaidya and C. Clifton. Privacy-preserving top-k queries. In *IEEE International conference on data engineering (ICDE)*, 2005.

[143] J. Vaidya and C. Clifton. Privacy-preserving kth element score over vertically partitioned data. *IEEE Trans. on Knowl. and Data Eng.*, 21(2):253–258, 2009.

[144] M. van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Workshop on Hot Topics in Security (HotSec)*, 2010.

[145] V. Verykios, E. Bertino, I. Fovino, L. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *ACM Sigmod Record*, 33(1):50–57, 2004.

[146] A. Wagner and B. Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. STCA security workshop, WET ICE 2005 Linköping, Sweden, 2005.

[147] S. Wei, J. Mirkovic, and E. Kissel. Profiling and clustering internet hosts. In *2006 International Conference on Data Mining*, 2006.

[148] WLHC: Worldwide LHC Computing Grid. `http://lcg.web.cern.ch/lcg/public/`.

[149] J. Wolkerstorfer. Secret-Sharing Hardware Improves the Privacy of Network Monitoring. In *Workshop on Data Privacy Management (DPM)*, 2010.

[150] M. Woo, J. Reiter, A. Oganian, and A. Karr. Global measures of data utility for microdata masked for disclosure limitation. *Journal of Privacy and Confidentiality*, 1(1):7, 2009.

[151] L. Xiong, S. Chitti, and L. Liu. Topk queries across multiple private databases. In *IEEE International conference on distributed computing systems (ICDCS)*, 2005.

[152] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *SIGCOMM*, 2005.

[153] A. Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science*, 1982.

[154] V. Yegneswaran, P. Barford, and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Network and Distributed System Security Symposium (NDSS)*, 2004.

[155] W. Yurcik, C. Woolam, G. Hellings, L. Khan, and B. Thuraisingham. Privacy/Analysis Tradeoffs in Sharing Anonymized Packet Traces: Single-Field Case. In *Conference on Availability, Reliability and Security (ARES)*, 2008.

[156] J. Zhang, P. Porras, and J. Ullrich. Highly predictive blacklisting. In *USENIX Security Symposium*, 2008.

[157] A. Ziviani, A. Gomes, M. Monsores, and P. Rodrigues. Network anomaly detection using nonextensive entropy. *Communications Letters, IEEE*, 11(12):1034–1036, 2007.

Acknowledgments

First and foremost, I thank Prof. Bernhard Plattner for enabling and supporting this thesis and for providing the freedom to explore own visions to his PhD students.

I am grateful to Martin May and Xenofontas Dimitropoulos for supervising this thesis and for many interesting discussions. I thank Douglas Dykeman from IBM Research for being a co-examiner of this thesis. I'd like to extend my thanks to Vassilis Zikas for assisting with MPC matters and to Andreas Kind, Thomas Locher, Marc Stöcklin, and Jeroen Massar for the good collaboration within the project on multi-domain traffic aggregation.

Special thank goes to SWITCH for providing access to their NetFlow data, which allowed evaluation of my ideas in practice. I thank Simon Leinen and Peter Haag from SWITCH for their support with the NetFlow data. My gratitude extends to Bernhard Tellenbach and Brian Trammell who have invested a lot of time into managing the CSG data cluster. Also, I want to thank Dominik Schatzmann and Bernhard Tellenbach for developing and supporting convenient tools for NetFlow data analysis [126].

I am deeply grateful to my collaborators and coauthors Daniela Brauckhoff, Mario Strasser, Dominik Schatzmann, Bernhard Tellenbach, Dilip Many, Brian Trammell, Elisa Boschi, David Gugelmann, Thrasyvoulos Spyropoulos, Didier Sornette, and Thomas Maillart. Special thank goes to my office mate Dominik Schatzmann for being such a good companion through the ups and downs of PhD student life. Also, I would like to thank all my colleagues from the Communication Systems Group (CSG) who have not been mentioned so far, in particular (in alphabetical order) Rainer Baumann, Ehud Ben Porat, Bernhard Distl, Thomas Dübendorfer, Stefan Frei, Francesco Fusco, Simon Heimlicher, Theus Hossmann, Eduard Glatz, Merkourios Karaliopoulos, Ariane Keller, Franck Legendre, Vincent Lenders, Hannes Lubich, Jose Mingorance-Puga, Wolfgang Mühlbauer, Stephan Neuhaus, Andreea Picu, Gabriel Popa, Ilias Raftopoulos, Sacha Trifunovic, Arno Wagner, and Jinyao Yan. Moreover, I thank Caterina Sposato, Beat Futterknecht, Thomas Steingruber, and Damian Friedli for providing smooth administrative and technical support.

Also, I want to thank the students and interns I supervised for their contributions to SEPIA and my research in general, namely (in reverse temporal order) Manuel Widmer, Christoph Renner, Dilip Many (in this role), Aarno Aukia, David Sauter, René Bühlmann, Lisa Hauser(-Barisic), and Debora Battisti.

Thank goes to Lukas Ruf for providing the LaTeX template for this thesis. I acknowledge Word Info (`www.wordinfo.info`) for the permission to use their illustration of the "Blind Men and the Elephant" (Figure 1.1).

Last but not least, I am deeply indebted to my parents, Ernst and Rosmarie, and my wife Mignon for their love and unconditional support throughout my studies and research.

Curriculum Vitae

Martin Burkhart was born in Münsterlingen, Switzerland on February 6, 1978. He is married and the proud father of little Sophia (born 2010).

Educational Curriculum Vitae

2007 - 2011	**Doctor of Sciences** Swiss Federal Institute of Technology (ETH) Zurich Advisor: Prof. Dr. Bernhard Plattner
1998 - 2003	**Master of Science in Computer Science** Swiss Federal Institute of Technology (ETH) Zurich
1993 - 1998	**Grammar school** Kantonsschule Romanshorn, Switzerland

Working and Teaching Experience

2007 - 2011	**Teaching Assistant** Swiss Federal Institute of Technology (ETH) Zurich
2003 - 2007	**Senior Software Engineer** adesso Schweiz AG
2000 - 2002	**Software Engineer** Glance AG