Privacy-Preserving Distributed Network Troubleshooting—Bridging the Gap between Theory and Practice

MARTIN BURKHART and XENOFONTAS DIMITROPOULOS, ETH Zurich

Today, there is a fundamental imbalance in cybersecurity. While attackers act more and more globally and coordinated, network defense is limited to examine local information only due to privacy concerns. To overcome this privacy barrier, we use secure multiparty computation (MPC) for the problem of aggregating network data from multiple domains. We first optimize MPC comparison operations for processing high volume data in near real-time by not enforcing protocols to run in a constant number of synchronization rounds. We then implement a complete set of basic MPC primitives in the SEPIA library. For parallel invocations, SEPIA's basic operations are between 35 and several hundred times faster than those of comparable MPC frameworks. Using these operations, we develop four protocols tailored for distributed network monitoring and security applications: the entropy, distinct count, event correlation, and top-k protocols. Extensive evaluation shows that the protocols are suitable for near real-time data aggregation. For example, our top-k protocol PPTKS accurately aggregates counts for 180,000 distributed IP addresses in only a few minutes. Finally, we use SEPIA with real traffic data from 17 customers of a backbone network to collaboratively detect, analyze, and mitigate distributed anomalies. Our work follows a path starting from theory, going to system design, performance evaluation, and ending with measurement. Along this way, it makes a first effort to bridge two very disparate worlds: MPC theory and network monitoring and security practices.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Algorithms, Design, Experimentation, Measurement, Security

Additional Key Words and Phrases: Applied cryptography, secure multiparty computation, collaborative network security, anomaly detection, network management, root-cause analysis, aggregation

ACM Reference Format:
Burkhart, M. and Dimitropoulos, X. 2011. Privacy-preserving distributed network troubleshooting—Bridging the gap between theory and practice. ACM Trans. Info. Syst. Sec. 14, 4, Article 31 (December 2011), 30 pages.
DOI = 10.1145/2043628.2043632 http://doi.acm.org/10.1145/2043628.2043632

1. INTRODUCTION

For almost thirty years, MPC techniques [Yao 1982] 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 dataset is also securely computable using MPC techniques [Goldreich et al. 1987]. However, designing solutions that are practical in terms of running time and communication overhead is far from trivial. For this reason, MPC techniques have attracted almost exclusively theoretical interest in the last decades. Recently, optimized basic primitives, such as comparisons [Damgård et al. 2006; Nishide and Ohta 2007], make progressively possible the use of MPC in

This work was supported by the Swiss Commission for Technology and Innovation (project no. 11623.2 PFES-ES) and by the DEMONS project funded by the EU 7th Framework Programme (G.A. no. 257315).
Author's address: email: burkhart@tik.ee.ethz.ch.

DOI 10.1145/2043628.2043632 http://doi.acm.org/10.1145/2043628.2043632

simple real-world applications. Remarkably, the first real-world application of MPC, an auction, was demonstrated in 2009 [Bogetoft et al. 2009].

A wealth of network security and monitoring problems have better solutions if a group of organizations collaborates and aggregates private data to jointly perform a computation. For example, IDS alert correlation requires the joint analysis of private alerts [Yegneswaran et al. 2004; Lincoln et al. 2004]. Similarly, aggregation of private data is useful for alert signature extraction [Parekh et al. 2006], collaborative anomaly detection [Ringberg 2009], multi-domain traffic engineering [Machiraju and Katz 2004], and detecting traffic discrimination [Tariq et al. 2009]. Despite the clear benefits of collaboration, organizations presently largely avoid sharing sensitive data as they are afraid of violating privacy laws, empowering attackers, or offering an advantage to their competitors.

Using MPC for collaborative network monitoring and security applications appears as an ideal match. MPC provides an excellent solution to the privacy-utility tradeoff: full utility is in theory attainable as any function can be turned into an MPC protocol, while the computation allows for information theoretic privacy, that is, the strongest form of privacy, of input data. However, the main drawback of MPC is its computational overhead. Network monitoring and security applications impose much stricter requirements on the performance of MPC protocols than, for example, the input bids of an auction. This is because they typically need to process voluminous input data in an online fashion. For example, anomaly detectors monitor online how traffic is distributed over port numbers, IP address ranges, and other traffic features. Moreover, network monitoring and security protocols need to meet *near real-time* guarantees[1]. This is not presently possible with existing MPC frameworks.

In this work we bridge the gap between MPC theory and network monitoring and security practices. In the theory front, we first introduce optimized basic MPC comparison operations implemented with performance of concurrent execution in mind, which is needed for processing voluminous input data. By not enforcing protocols to run in a constant number of rounds, we are able to 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. This demonstrates that the widely-used constant-round design standard is not a panacea in MPC protocol design: allowing many parallel invocations and removing the constant-round constraint reduces total running time.

Using our basic operations as foundation, we then design, implement, and evaluate SEPIA (Security through Private Information Aggregation), an MPC library for efficiently aggregating multi-domain network data. A typical setup for SEPIA is depicted in Figure 1, 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 [Shamir 1979]. 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 description of our security assumptions and a discussion of their implications is presented in Section 5.

We then design four MPC protocols on top of SEPIA's basic primitives [Burkhart et al. 2010; Burkhart and Dimitropoulos 2010]. Our protocols are inspired from specific

[1]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.

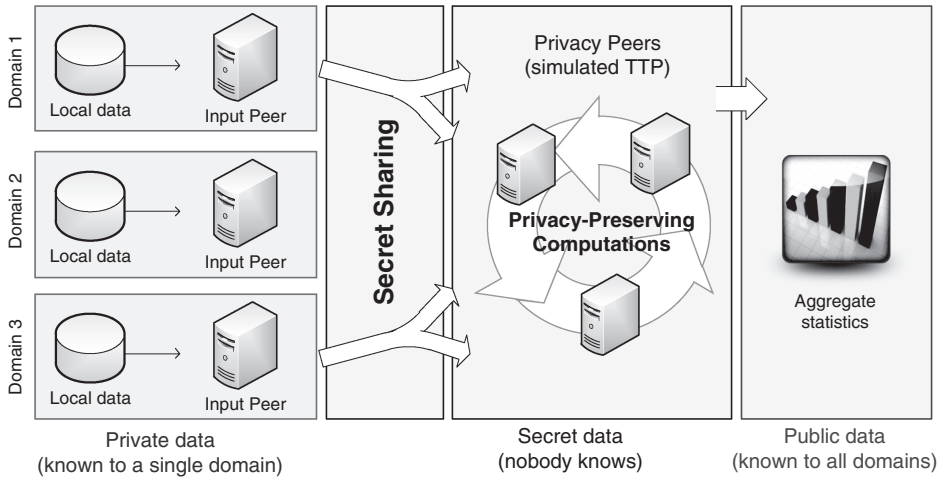


Fig. 1. Deployment scenario for SEPIA.

network monitoring and security applications, but at the same time they are also general-purpose and can be used for other applications. The *entropy protocol* allows input peers to aggregate local histograms and to compute the entropy of the aggregate histogram. Similarly, the *distinct count protocol* finds and reveals the number of distinct, non-zero aggregate histogram bins. These two metrics are commonly used in anomaly detection. The *event correlation protocol* correlates arbitrary events across networks and only reveals the exact events that appear in a minimum number of input peers and have aggregate frequency above a configurable threshold. Finally, our *top-k protocol* uses a sketch data structure to minimize expensive comparison operations, estimates the global top-k items over private input lists, and scales much better than the event correlation protocol. The event correlation and top-k protocols can be used, for example, to implement a privacy-preserving distributed alert aggregation application, similarly to what DShield[2] provides as a trusted third party. In addition, we implement the four protocols, along with a state-of-the-art *vector addition protocol* for aggregating additive timeseries or local histograms, in the SEPIA library.

We extensively evaluate our protocols on a local cluster and on PlanetLab, a distributed Internet-wide testbed, using real-world network traffic traces. We find that our basic operations are much more scalable than the corresponding operations of other general-purpose MPC frameworks. In addition, we show that our four protocols can be used in practical scenarios meeting near real-time processing constraints.

Finally, to connect the dots between MPC theory and network monitoring and security practices, we apply our protocols on the traffic data of 17 networks collected during the global Skype outage in August 2007. We demonstrate several different ways the networks could have collaborated to better detect, troubleshoot, and mitigate the 2-day anomaly than each isolated network could have accomplished alone. This is the first work to apply MPC on real traffic traces and to demonstrate that MPC-based collaborative network monitoring and security applications are both feasible from the computational overhead point of view and useful for addressing real-world network problems.

[2]www.dshield.org.

In summary, we make the following contributions.

(1) We introduce efficient MPC comparison operations, which outperform constant-round alternatives for many parallel invocations.
(2) We design four novel MPC protocols tailored for but not restricted to specific network monitoring and security applications.
(3) We introduce the SEPIA library, in which we implement a complete set of basic operations optimized for parallel execution, our four MPC protocols, and a state-of-the-art vector addition protocol. We make SEPIA publicly available [Burkhart et al. 2010].
(4) We extensively evaluate the performance of SEPIA and of our protocols in realistic settings using real traffic traces.
(5) We apply our protocols to traffic traces from 17 networks and show how they enable the collaborative detection and troubleshooting of the global 2007 Skype outage.

The article is organized as follows: We introduce the computation scheme in the next section and present our optimized comparison operations in Section 3. In Section 4, we give an overview over the SEPIA system and specify the adversary model and security assumptions. We present our privacy-preserving protocols in Section 5 and evaluate their performance in Section 6. We apply our protocols to real network data in Section 7, demonstrating SEPIA's benefits in distributed troubleshooting and early anomaly detection. Finally, we discuss related work in Section 8 and conclude our paper in Section 9.

2. PRELIMINARIES

Our implementation is based on Shamir secret sharing [Shamir 1979]. In order to *share* a secret value s among a set of m players, the dealer generates a random polynomial f of degree $t = \lfloor (m-1)/2 \rfloor$ over a prime field $\mathbb{Z}_p$ with $p > s$, such that $f(0) = s$. Each player $i = 1 \ldots m$ then receives an evaluation point $s_i = f(i)$ of f. s_i is called the share of player 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. For simplicity of presentation, we use $[s]$ to denote the vector of shares $(s_1, \ldots, s_m)$ and call it a *sharing* of s. In addition, we use $[s]_i$ to refer to s_i. The reconstruction of a sharing is denoted by $s = \mathit{reconstruct}([s])$. Unless stated otherwise, we choose p with 62 bits such that arithmetic operations on secrets and shares can be performed by CPU instructions directly, not requiring software algorithms to handle big integers.

2.1. Addition and Multiplication

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: $[a+b]_i = [a]_i + [b]_i$. 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, that is, $[a+c]_i = [a]_i + c$. 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 guarantee randomness and to correct the degree of the new polynomial [Ben-Or et al. 1988; Gennaro et al. 1998]. 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 $[d_i]$. Together, the players then perform a distributed Lagrange interpolation to compute $[ab] = \sum_i \lambda_i [d_i]$ where λ_i are the Lagrange coefficients. Thus,

a distributed multiplication requires a synchronization round with m^2 messages, as each player i sends to each player j the share $[d_i]_j$.

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 and public constants. 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.

2.2. Communication

A set of independent multiplications, for instance, $[ab]$ 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 high delays of individual messages, potentially leading to a reordering of message arrival. 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.

2.3. 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, Goldwasser, and Wigderson [Ben-Or et al. 1988]. In particular, they showed that in the semi-honest model, where adversarial players follow the protocol but try to learn as much as possible by sharing the information they received, 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 [Canetti 2001]. Because the scheme is information-theoretically secure, that is, 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 $p = 2$. Because we use SSL for implementing secure channels, the *overall system* relies on PKI and symmetric encryption and is only computationally secure.

3. MAKING MULTIPARTY COMPUTATION PRACTICAL

Unlike addition and multiplication, comparison of two shared secrets is a very expensive operation. The complexity of an MPC protocol is typically assessed counting the number of distributed multiplications and 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 [Bar-Ilan and Beaver 1989; Beaver et al. 1990; Gennaro et al. 2002]. For instance, the authors of FairplayMP explain why they chose an algorithm that

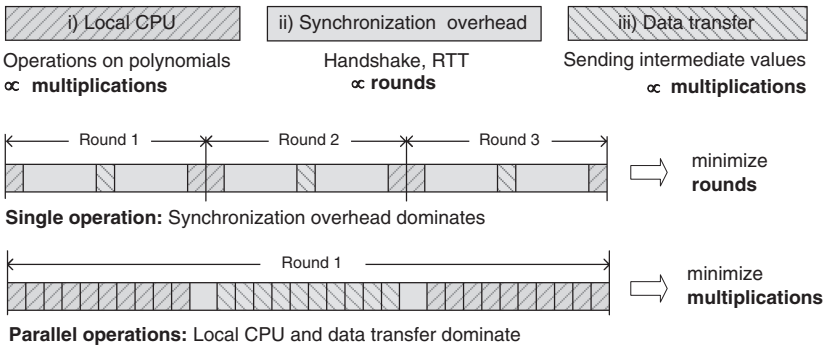


Fig. 2. Source of delay in composite MPC protocols.

evaluates arbitrary functionality in a constant number of only 8 synchronization rounds [Ben-David et al. 2008, page 2].

> 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.

Presently, the design of MPC protocols widely follows a *constant-round* paradigm. Constant-round means that the number of synchronization rounds in a protocol does not depend on input parameters.

3.1. State-of-the-art Comparison Operations

Focusing on comparison operations, Damgård et al. [2006] introduced the bit-decomposition protocol that achieves comparison by decomposing shared secrets into a shared bitwise representation. On shares of individual bits, comparison is straightforward. With $l = \log_2(p)$, the protocols in Damgård et al. [2006] 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 [2007] 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 Nishide and Ohta [2007] requires 2592 distributed multiplications, each triggering m^2 messages to be transmitted over the network. This is clearly not acceptable if we want to compare lists of shared IP addresses against each other.

3.2. 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 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, that is, not varying at runtime.

As illustrated in Figure 2, 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 Nishide and Ohta [2007]). 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, iii) becomes 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 fewer 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.

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

3.3. Optimized Equality Check

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

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

Using (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.[3] 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 this example for comparing IPv4 addresses, this substantially reduces the multiplication count by a factor of 76 from 2592 to 34.

3.4. Optimized Less-Than

For less-than comparison, we base our implementation on Nishide's protocol [Nishide and Ohta 2007]. However, we apply modifications to again reduce the overall number

[3]Asymptotically, this brings equality testing down to a single multiplication per bit of the secret, which matches the complexity for doing equality checks on bitwise shared secrets. The use of expensive bit-decomposition protocols [Damgård et al. 2006] is therefore unnecessary, at least for equality testing.

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 the three predicates $a < p/2$, $b < p/2$, and $a - b < p/2$. 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, for instance, because b is not secret but publicly known, saves one of the three LSB calls and hence 1/3 of the multiplications.

Due to space restrictions we omit reproducing the entire protocol but focus on the modifications we apply. An important subprotocol in Nishide's construction is *PrefixOr*. Given a sequence of shared bits $[a_1], \ldots, [a_l]$ with $a_i \in \{0, 1\}$, *PrefixOr* computes the sequence $[b_1], \ldots, [b_l]$ 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.

3.5. Optimized Implementation

In addition to optimizing the design of basic primitives, we also optimize their implementation. First, each connection, 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 SEPIA protocols benefit from multi-core systems for computation-intensive tasks. Second, 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[4] under the LGPL license.

3.6. Benchmark of Basic Operations

In this section we compare the resulting performance of basic SEPIA operations to those of other frameworks such as FairplayMP [Ben-David et al. 2008] and VIFF v0.7.1 [Damgård et al. 2009]. Besides performance, one aspect to consider is, of course, usability. Whereas the SEPIA library currently only provides an API to developers, FairplayMP allows to write 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.[5] 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. Unlike VIFF and SEPIA, which use an information-theoretically secure scheme,

[4]http://www.sepia.ee.ethz.ch.

[5]We do not consider 1Gb/s to be a very realistic setting for our scenarios. However, as discussed later in Section 3.6.3, the gain of using 1Gb/s instead of 100Mb/s for total running time is only about 11%, because the running time is bounded by local computation.

Table I. Comparison of Framework Performance with $m = 5$

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

FairplayMP requires the choice of an adequate security parameter k. We set $k = 80$, as suggested by the authors [Ben-David et al. 2008].

3.6.1. General Results. Table I 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 5.2) taking 3 minutes with SEPIA would take roughly 2 days with VIFF. This extends the range of practically runnable MPC protocols significantly.

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.

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

Notably, SEPIA's *equal* operation is approximately 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 our conjecture that with many parallel operations, the number of multiplications becomes the dominating factor in running time.

3.6.2. SEPIA versus Sharemind. Sharemind [Bogdanov et al. 2008] 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 ($m = 3$), that is, 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 I. However, regarding performance for $m = 3$, Sharemind is comparable to SEPIA. According to Bogdanov et al. [2008], 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 pre-generated randomness). Sharemind does not directly implement *equal*, but it could be implemented using 2 invocations of GTE, leading to $\approx$115 operations/s. SEPIA's *equal* is clearly faster with up to 3,400 invocations/s. SEPIA demonstrates that operations based on Shamir shares have comparable performance to operations based on 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 more robust against node failures.

3.6.3. 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 10Mb/s links, communication time is clearly dominant, requiring 80% of the total running time. With 100Mb/s links, communication time goes down to 32% and with 1Gb/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.

4. SYSTEM OVERVIEW

Our system, as depicted in Figure 1, has a set of n users called *input peers*. The 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. In addition, we have m players called *privacy peers* that perform the computation of $f()$ by simulating a trusted third party (TTP). Each entity can take both roles, acting only as an input peer, privacy peer or both.

4.1. Adversary Model and Security Assumptions

We use the semi-honest (aka honest-but-curious) adversary model for privacy peers. That is, honest privacy peers follow the protocol and do not combine their information. Semi-honest privacy peers do follow the protocol but try to infer as much as possible from the values (shares) they learn, also by combining their information. 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, that is, as long as the majority of privacy peers is honest. The selection of privacy peers is subject to an offline decision finding process preceding the actual aggregation phase. Even if some of the input peers do not trust each other, we believe it is realistic to assume that they will agree on a set of most-trusted participants and/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 have concrete incentives not to offend the input peers, that is, their customers.

The function $f()$ is specified *as if* a TTP was available. MPC guarantees that no information is leaked *from the computation process*. However, just learning the resulting value $f()$ could allow inference of sensitive information. For example, if the input bit of all input peers 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. 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 to do this is *differential privacy* [Dwork 2008; McSherry and Mahajan 2010], which systematically randomizes answers to database queries to prevent inference of sensitive input data. 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 [Duan 2009]. Note, however, that differential

privacy assumes input data records to be independent. Therefore, any application of differential privacy to packet or flow trace databases needs 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.

Prior to running the protocol, the m privacy peers set up a secure, that is, 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.

4.2. Privacy-Performance Tradeoff

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 [Burkhart et al. 2010]. On the other hand, the number of tolerated colluding privacy peers also scales linearly with m. Hence, the choice of m involves a privacy-performance tradeoff. The separation of roles into input and privacy peers allows to tune this tradeoff independently of the number of input providers.

5. PROTOCOLS FOR DISTRIBUTED NETWORK MONITORING AND SECURITY

In this section, we introduce our privacy-preserving protocols that are built using the basic operations introduced in Section 3. The protocols for network statistics (Section 5.1) and event correlation (Section 5.2) are presented in a summarized form. More details on these protocols are available in [Burkhart et al. 2010]. The design of the top-k protocol PPTK(S) is given in more detail in Section 5.3 and Section 5.4.

The main protection goal of these protocols is to hide sensitive information derived from network traffic, such as packet payloads, IDS alerts, user profiles, service statistics, or link utilization. As an example, it is usually easy to map IP addresses to desktop computers and their owners. Aggregating traffic by IP address allows the construction of 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, along with the services deployed, 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. Hence, we aim at protecting the privacy of users, servers, and networks. Moreover, the leakage of sensitive business information must be prevented.

On the technical side, the design of efficient composite MPC protocols faces two main challenges: First, as discussed in Section 3, synchronization costs of MPC protocols can only be amortized if many operations are independent and can be performed in parallel. Hence, one challenge in designing new protocols is to use as little MPC operations as possible but at the same time as many parallelizable MPC operations as possible. Second, algorithms must be *data-oblivious*. MPC algorithms guarantee that the computation process does not leak even a single bit about input data. Therefore, learning the value of intermediate predicates, such as $a < b$ may constitute a privacy breach and has to be omitted. Consequently, simple and heavily used data structures, such as arrays providing access to element at index i within one CPU cycle, are no longer available if i has to remain secret [Damgård et al. 2011]. 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 Section 6.

5.1. 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. While these metrics are often used for network anomaly detection, we emphasize that we merely compute aggregate basic metrics and do not implement turnkey anomaly detectors. For example, with our protocols one can compute a matrix of entropy values for different traffic features and for different network flows and then apply the state-of-the-art detectors based on PCA [Lakhina et al. 2005] or KLE [Brauckhoff et al. 2009b] to detect anomalies. An alternate direction would be to implement the entire detection process, for instance, based on PCA or KLE, in MPC. However, this is beyond the scope of this article.

5.1.1. 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 $\mathbf{D} = \sum_{i=1}^{n} \mathbf{d_i}$. This protocol requires no distributed multiplications and only one round.

5.1.2. Entropy Computation. The computation of the entropy of feature distributions has been successfully applied in network anomaly detection [Lakhina et al. 2005; Brauckhoff et al. 2009b; Li et al. 2006; Tellenbach et al. 2011]. 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 [Tellenbach et al. 2011]. The 1-parametric Tsallis entropy is defined as:

$$H_q(Y) = \frac{1}{q-1}\left(1 - \sum_k (p_k)^q\right)$$

and has a direct interpretation in terms of moments of order q of the distribution. In particular, the Tsallis entropy is a generalized, nonextensive 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.

A straight-forward approach to compute the entropy would be to privately aggregate local item histograms using the above vector addition protocol, then reconstruct the aggregate histogram and calculate the entropy in a non-private manner. However, the aggregate distribution can still be very sensitive as it contains information for each item, for instance, per address prefix. For this reason, we compute $H(Y)$ without reconstructing any of the individual or aggregate item counts.

Because the rational numbers p_k can not be shared directly over a prime field, we perform the computation separately on private numerators (the individual item counts) and the public overall item count S. 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.

5.1.3. Distinct Count. Next, we devise a simple distinct count protocol leaking no intermediate information. The input peers provide Boolean histograms denoting for each item whether or not it is present in the local distribution. We then compute the logical OR of each histogram position to find which items were reported by any input peer. The logical OR can be computed according to $[x] \vee [y] = [x] + [y] - [x][y]$. Then, simply summing up all the bins of the aggregate boolean histogram gives the distributed count of distinct items which is then reconstructed.

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 before the final step. The complexity of this protocol is $(n-1)r$ multiplications in $\log_2 n$ rounds.

5.2. Event Correlation

The next protocol we present is more complex and 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), for instance, 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, for instance, a specific intrusion alert, he would be willing to contribute his information and collaborate. Likewise, an input peer might only be interested in reconstructing events of a certain impact, having a nonnegligible 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 (2) and (3) and reconstruct e_{ij} iff (4) holds.

$$[C_{ij}] := \sum_{i' \neq i, j'} equal([k_{ij}], [k_{i'j'}]) \tag{2}$$

$$[W_{ij}] := \sum_{i' \neq i, j'} [w_{i'j'}] \cdot equal([k_{ij}], [k_{i'j'}]) \tag{3}$$

$$([C_{ij}] \geq T_c) \wedge ([W_{ij}] \geq T_w) \tag{4}$$

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 presented in Burkhart et al. [2010, Section 4.1]. 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.

5.2.1. 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.

5.2.2. Probe Response Attacks. If aggregated security events are made publicly available, this enables probe response attacks against the system [Bethencourt et al. 2005]. 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, Shmatikov and Wang [2007] 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.

5.2.3. Limitations. 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 sensitive information or no information at all. The PPTKS protocol, which is introduced in the following sections, allows to specify the exact number of elements k that will be revealed, automatically adapting to dynamic traffic conditions. Depending on the application scenario, either fixed or dynamic thresholds may be desirable. In addition, PPTKS uses probabilistic data structures to scale much better for large item distributions. Whereas the event correlation protocol is efficient for aggregating few dozens of local events, for instance, 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 Section 6 that using PPTKS, it is possible to aggregate 180,000 distinct IP addresses within few minutes.

5.3. Top-k Protocol PPTK

We now design the privacy-preserving top-k protocol PPTK and evaluate its accuracy using real traffic traces. PPTK is the central component of the final PPTKS protocol described in the next section and also in Burkhart and Dimitropoulos [2010].

5.3.1. 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 $\mathbf{k} = \{k_0, k_1, \ldots, k_{H-1}\}$ and values $\mathbf{v} = \{v_0, v_1, \ldots, v_{H-1}\}$, 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:

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

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.

5.3.2. Aggregation. Each input peer shares the vectors $\mathbf{k}$ and $\mathbf{v}$ among the privacy peers. Let $[k_j^i]$ and $[v_j^i]$ 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, that is, $k_j^1 \neq k_j^2$. We use a collision resolution algorithm (see Alg. 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.

5.3.3. 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 Vaidya and Clifton [2005] and Aggarval et al. [2004] 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 $(H+1)\log_2 M$

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

Require: A shared vector [**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: $match = reconstruct(equal([biggercount], k))$
12: **if** $match == 0$ **then**
13: $toohigh = reconstruct(lessThan([biggercount], k))$
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** $reconstruct([lt_j]) == 0$ **then**
24: $topi_{count} = j$
25: $count = count + 1$
26: **end if**
27: **end for**
28: **return** (**topi**)

invocations of *lessThan* and $\log_2 M$ invocations of *equal*. The vast majority of these operations are independent and can be performed in parallel.

5.3.4. Resolving Global Collisions. 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.[6] 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.

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

[6]Note that Algorithm 2 fixes a minor bug we discovered during implementation of our pseudo-code in Burkhart and Dimitropoulos [2010].

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

Require: Shared keys $[k_j^1], [k_j^2], \ldots, [k_j^n]$, 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** $(reconstruct([maxk]), reconstruct([maxv]))$

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.

5.4. 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 trade-off 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 to correct errors introduced by hash collisions in a single array.

5.4.1. 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.

5.4.2. 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, 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.

5.5. Accuracy

We have thoroughly evaluated the accuracy of PPTKS in Burkhart and Dimitropoulos [2010]. Here, we provide a summary of the main results.

Depending on H and the item distribution, local and global hash collisions can introduce error. We used real traffic data to evaluate the accuracy of the top-k items found by PPTKS. We ran simulations on TCP destination port and IP address distributions of the six biggest customer networks of SWITCH [SWITCH]. In the scenario, the six customers want to compute the aggregate top-k items over their datasets. We used traffic of an entire day in August 2007, resulting in 96 timeslots of 15 minutes. For each timeslot, a new random seed for the hash function was used. The number of distinct IP addresses in the aggregate distribution varied between 70,000 during the night and approximately 180,000 during the day. We computed the correct set σ of aggregate top-k items and the approximate set $\widetilde{\sigma}$ as calculated by our protocol. We then compared the two to evaluate the accuracy of PPTKS using the following metrics:

(1) Percentage of correct top-k items: $|\sigma \cap \widetilde{\sigma}|/k$.
(2) The average rank distortion of items in $\sigma \cap \widetilde{\sigma}$, 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.

Figures 3 and 4 illustrate the accuracy improvements achieved by PPTKS for top-100 ports and IP addresses, respectively. Note that the configuration with $S = 1$ corresponds to PPTK. 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., 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 4. 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.

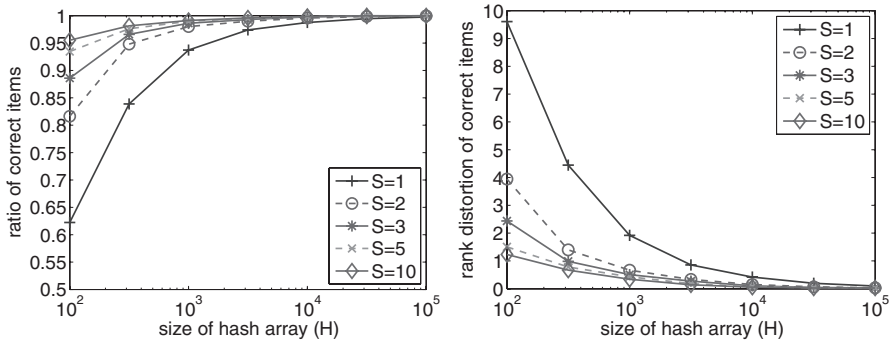


Fig. 3. Accuracy for top-100 ports using sketches with S hash arrays.

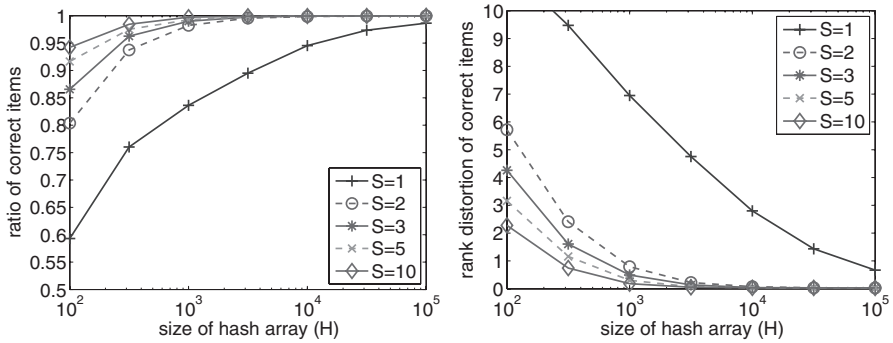


Fig. 4. Accuracy for top-100 IP addresses using sketches with S hash arrays.

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.

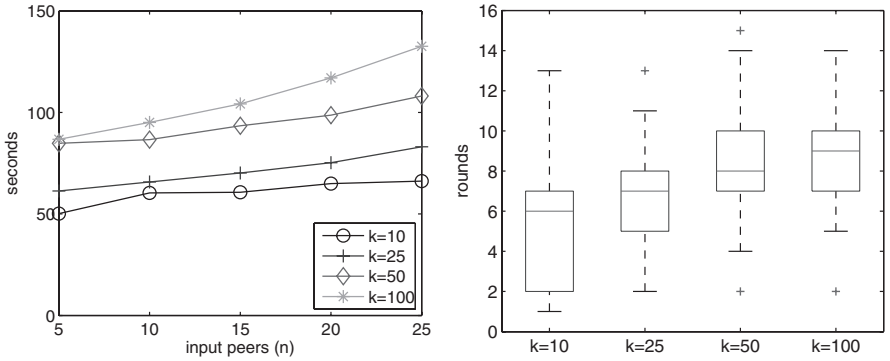
We emphasize that accurately answering top-10 queries instead of top-100 queries is much easier. For instance, for top-10 ports, a choice of $H = 316$ and $S = 2$ yields already 99.9% correct items (see [Burkhart and Dimitropoulos 2010] for details).

6. RUNNING TIME EVALUATION

In this section we evaluate the running time of our protocols in a variety of settings and show that they finish in near real time for typical tasks. As in Section 5, we summarize results for the network statistics and event correlation protocols (Section 6.1) and go more into details for the PPTKS protocol (Section 6.2). For more details on the performance of the first protocols, please refer to Burkhart et al. [2010], which also includes an evaluation for varying numbers of privacy peers (between 3 and 9) and an out-of-lab evaluation in an Internet-wide setting. Generally, the number of privacy peers m has a linear impact on the running time.

6.1. Network Statistics and Event Correlation

For evaluating these protocols, we performed a number of tasks listed below. We report here only running times for the largest of the evaluated settings, involving 25 input peers and 9 privacy peers. Each input and privacy peer was run on a separate host to ensure that all communication crossed the network. Link speed was 100Mbit/s.



(a) Mean total running time.

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

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

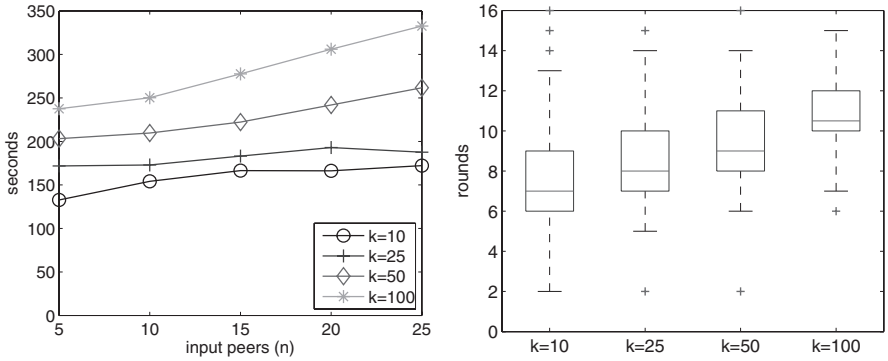
(1) *Volume Metrics.* Adding 21 volume metrics containing flow, packet, and byte counts, both total and separately filtered by protocol (TCP, UDP, ICMP) and direction (incoming, outgoing). Mean running time was below 1.6s.
(2) *Port Histogram.* Adding full port histograms. Input files contained 65,535 fields, each indicating the number of flows observed to the corresponding port. These local histograms were aggregated using the addition protocol. Mean running time was 50s.
(3) *Port Entropy.* Computing the Tsallis entropy of destination port distributions. The local input files contained the same information as for histogram aggregation. Mean running time was 67s.
(4) *Distinct count of AS numbers.* Aggregating the count of distinct source AS numbers in incoming traffic. The input files contained 65,535 columns, each denoting if the corresponding source AS number was observed. Mean running time was 88s.
(5) *Local top-30 events.* We correlated local top-30 events (750 events in total) to find the global top events. Events were reconstructed if a majority of input peers reported them. The key length was set to 24 bit. Mean running time was 209s.

6.2. Top-*k* Computation

The running times for PPTKS reported in Burkhart and Dimitropoulos [2010] are estimations based on benchmarks of the basic primitives. We have implemented PPTKS in SEPIA and here we evaluate the actual running time of PPTKS for different protocol parameters.

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 was again taken from the six biggest customers of the SWITCH network, as in Section 5.5. 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.

Figures 5 and 6 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



(a) Mean total running time.

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

Fig. 6. Running time statistics for top-k *IP address* reports ($m = 5$, $l = 33$ bits).

the smaller field size p. It is sufficient to use a p with 24 bits to represent the port values and intermediate aggregate counts for each port. For IP address reports, a p with 33 bits is necessary to represent IP addresses (the keys) as shared secrets. Recall from Section 3 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 5(b) and 6(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 preciser. Consequently, more rounds are needed for $k = 100$ than for $k = 10$. Nevertheless, variation is quite big. While for $k = 10$ in Figure 5(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.

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. Overall, the actual running times of PPTKS are lower than previously estimated in Burkhart and Dimitropoulos [2010]. For example, in our previous work we estimated a running time of 8.8 min for $H = 1000$, $S = 2$, $k = 100$, $m = 5$, and 20 input peers. In practice, this configuration requires only 5.0 min for IP address and 2.0 min for port reports. The main reason for this improvement is that our estimation used a worst-case estimate for the number of binary search rounds. For port reports, the speedup also comes from using a smaller field size with only 24 bits.

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 [Burkhart and Dimitropoulos 2010].

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 3.4). 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 introduced in Burkhart et al. [2010]. This could further reduce running time of the binary search phase for small values of k and item distributions with rather low item counts.

7. DISTRIBUTED TROUBLESHOOTING IN PRACTICE

Developing fancy MPC protocols and even demonstrating that they run in near real-time is not useful for the network practitioner unless the protocols solve real-world network problems. Therefore, we go an important step further and apply our collaborative protocols to real traffic traces to demonstrate their utility in practice. In particular, we demonstrate how our protocols enable troubleshooting of distributed 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 [Rossi et al. 2009]. 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 timeseries of the UDP flow count metric and applied a simple detector to identify anomalies. For each timeseries, 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 anomalous time intervals. In Figure 7 we illustrate the local timeseries for the six largest organizations and the aggregate timeseries. 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.

7.1. 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 troubleshooting 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 8 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

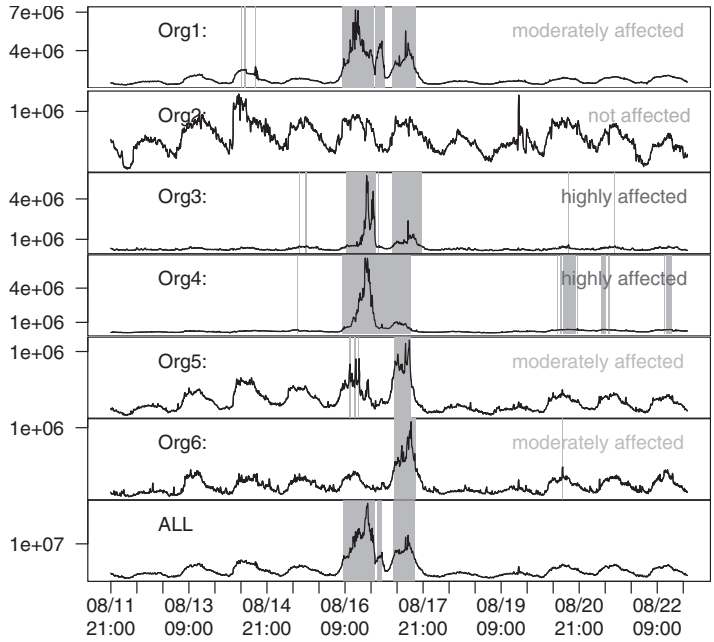


Fig. 7. Flow count in 5' windows with anomalies for the biggest organizations and aggregate view (ALL). Each organization sees its local and the aggregate traffic. The 2007 Skype anomaly is visible in the middle.

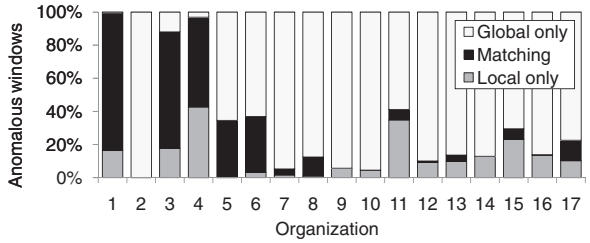


Fig. 8. Correlation of local and global anomalies for organizations ordered by size (1=biggest).

more to the aggregate view. While some organizations are highly correlated with the global view, for instance, organization 3 that notably contributes only 7.4% of the total traffic; other organizations are barely correlated, for instance, organizations 9 and 12; and organization 2 has no local anomalies at all.

7.2. Relative Anomaly Size

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

Table II.
Organizations profiting from an early anomaly warning by aggregation

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

7.3. 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 II we list the organizations that had considerable lag, that is, 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 8, 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 [Burkhart et al. 2010] is useful to decide whether similar anomaly signatures are observed in the participating networks. Anomaly signatures can be extracted automatically using actively researched techniques [Brauckhoff et al. 2009a; Ranjan et al. 2007].

7.4. 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 customer's UDP traffic and show top-k statistics for incoming destination ports in Figure 9, and outgoing destination IP addresses in Figure 10. 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 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 propagate. 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, for instance, destination port 19690 accounts for 93% of all flows of organization #4, at the peak rate (see Figure 9). 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 9, 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 7, they are able to calculate the absolute flows for each port

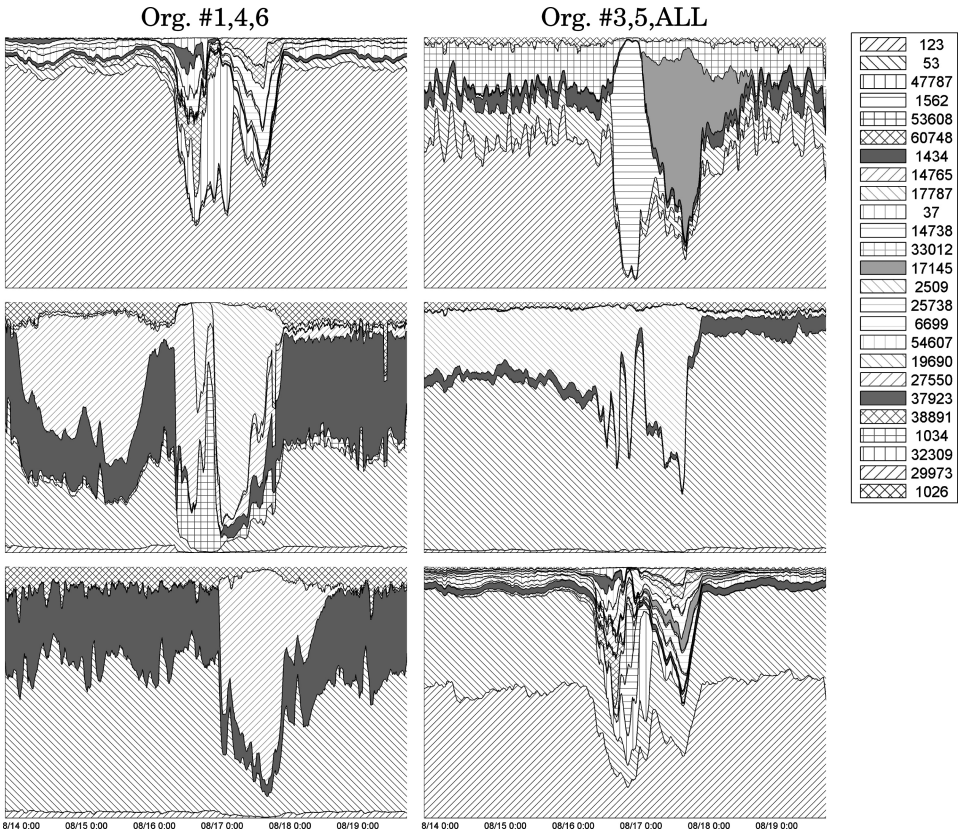


Fig. 9. Global top-25 incoming UDP destination ports and their local visibility 6 days around the 2007 Skype anomaly.

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 10 reveals that there is a subset of anomalous external IP addresses 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.

8. RELATED WORK

The similarities and differences between our work and MPC frameworks like FairplayMP [Ben-David et al. 2008], VIFF [Damgård et al. 2009], and Sharemind [Bogdanov et al. 2008] are discussed in Section 3.6.

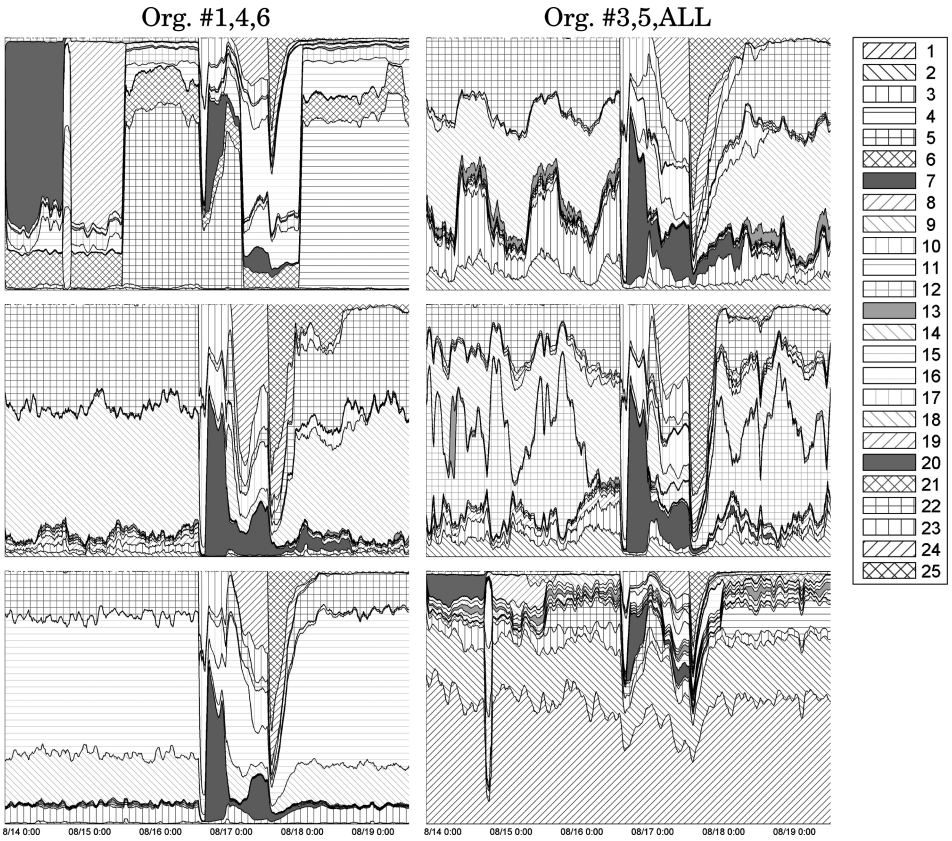


Fig. 10. Global top-25 outgoing UDP destination IP addresses (replaced by identifiers) and their local visibility 6 days around the 2007 Skype anomaly.

8.1. Top-k Queries

A lot of work [Fagin 1996; Fagin et al. 2001; Akbarinia et al. 2007; Chang and Hwang 2002; Marian et al. 2004; Vaidya and Clifton 2005] 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 [1996], 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 Babcock and Olston [2003], 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, Vaidya and Clifton [2005, 2009] 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 [Fagin 1996]. 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 Vaidya and

Clifton [2005] is that all items are reported by all sites. However, this is not the case with our type of data, that is, 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 depends on x.

Xiong et al. [2005] 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.

8.2. Privacy-Preserving Network Applications

Roughan and Zhang [2006b] first proposed the use of MPC techniques for a number of applications relating to traffic measurements, including the estimation of global traffic volume and performance measurements [Roughan and Zhang 2006a]. In addition, the authors identified that MPC techniques can be combined with commonly-used traffic analysis methods and tools, such as time-series algorithms and sketch data structures. Our work is similar in spirit, yet it extends their work by introducing new MPC protocols for event correlation, entropy, and distinct count computation and by implementing these protocols in a ready-to-use library.

Data correlation systems that provide strong privacy guarantees for the participants achieve data privacy by means of (partial) data sanitization based on bloom filters [Stolfo 2004] or cryptographic functions [Lincoln et al. 2004; Lee et al. 2006]. However, data sanitization is in general not a lossless process and therefore imposes an unavoidable trade-off between data privacy and data utility.

Chow et al. [2009] and Applebaum et al. [2010] avoid this trade-off 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 derandomized at the front end. Applebaum et al. present a semicentralized 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 nonblinded 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 nonblinded 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.

Two problems related to matching against private data sets are privacy-preserving set intersection (in which each party wants to learn the intersection of all private data sets) and privacy-preserving set matching (in which each party wants to learn whether its elements can be matched in any private set of the other parties). Efficient solutions to these problems have been proposed in Freedman et al. [2004] and Sang et al. [2006]. Both solutions are based on homomorphic encryption.

9. CONCLUSION

The area of secure multiparty computation has produced a tremendous body of theoretical work with strong feasibility results. Unfortunately, designing efficient MPC solutions is far from trivial, which is the main reason why MPC currently has a very small number of real-world applications. Using MPC for collaborative network monitoring and security applications appears as an ideal alternative to the intricate privacy-utility trade-off involved in obfuscation and anonymization of sensitive data. However, applying MPC to network monitoring and security problems introduces even more challenging efficiency constraints than previously considered in the literature, as input data is typically voluminous and responses are needed in near real time. In this work we make a number of contributions on a range of topics bridging the gap between MPC theory and network monitoring and security practices.

The widely used constant-round design paradigm for MPC protocols ignores the number of distributed multiplications, leading to bad overall performance when many parallel MPC operations are required. Instead, the protocol design should strive to optimize both the number of multiplications and synchronization rounds. Based on this insight, we introduce MPC comparison operations that need significantly less CPU and bandwidth resources than existing alternatives.

We implement our basic primitives along with four optimized MPC protocols inspired from specific collaborative network monitoring and security applications in the SEPIA library, which is made publicly available. We show that our protocols are sufficiently efficient for real-world network security and monitoring applications. For example, our top-k protocol PPTKS accurately aggregates counts for 180,000 distinct IP addresses distributed across six networks in only a few minutes. In addition, we show that SEPIA's basic operations are much faster than those of comparable state-of-the-art MPC frameworks.

Finally, we apply our MPC protocols to traffic traces from 17 networks and demonstrate how collaboration based on MPC can help address a number of different real-world problems faced by network security and monitoring practitioners.

ACKNOWLEDGMENTS

We thank Dilip Many and Mario Strasser for their contributions to the SEPIA library.

REFERENCES

AGGARVAL, G., MISHRA, N., AND PINKAS, B. 2004. Secure Computation of the kth-Ranked Element. In *Proceedings of the EUROCRYPT*.

AKBARINIA, R., PACITTI, E., AND VALDURIEZ, P. 2007. Best position algorithms for top-k queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*.

APPLEBAUM, B., RINGBERG, H., FREEDMAN, M. J., CAESAR, M., AND REXFORD, J. 2010. Collaborative, privacy-preserving data aggregation at scale. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*.

BABCOCK, B. AND OLSTON, C. 2003. Distributed top-k monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

BAR-ILAN, J. AND BEAVER, D. 1989. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*.

BEAVER, D., MICALI, S., AND ROGAWAY, P. 1990. The round complexity of secure protocols. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*.

BEN-DAVID, A., NISAN, N., AND PINKAS, B. 2008. FairplayMP: a system for secure multi-party computation. In *Proceedings of the Conference on Computer and Communications Security (CCS)*.

BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*.

BETHENCOURT, J., FRANKLIN, J., AND VERNON, M. 2005. Mapping internet sensors with probe response attacks. In *Proceedings of the 14th USENIX Security Symposium*.

BOGDANOV, D., LAUR, S., AND WILLEMSON, J. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

BOGETOFT, P., CHRISTENSEN, D., DAMGÅRD, I., GEISLER, M., JAKOBSEN, T., KRØIGAARD, M., NIELSEN, J., NIELSEN, J., NIELSEN, K., PAGTER, J., ET AL. 2009. Secure multiparty computation goes live. In *Proceedings of the Financial Cryptography Association*.

BRAUCKHOFF, D., DIMITROPOULOS, X., WAGNER, A., AND SALAMATIAN, K. 2009a. Anomaly extraction in backbone networks using association rules. In *Proceedings of the Internet Measurement Conference (IMC)*.

BRAUCKHOFF, D., SALAMATIAN, K., AND MAY, M. 2009b. Applying PCA for Traffic Anomaly Detection: Problems and Solutions. In *Proceedings of INFOCOM*.

BURKHART, M. AND DIMITROPOULOS, X. 2010. Fast privacy-preserving top-k queries using secret sharing. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*.

BURKHART, M., STRASSER, M., MANY, D., AND DIMITROPOULOS, X. 2010. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of the 19th USENIX Security Symposium*.

CANETTI, R. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*.

CHANG, K. AND HWANG, S. 2002. Minimal probing: Supporting expensive predicates for top-k queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

CHOW, S. S. M., LEE, J.-H., AND SUBRAMANIAN, L. 2009. Two-party computation model for privacy-preserving queries over distributed databases. In *Proceedings of the Network and Distributed Systems Society Symposium (NDSS)*. The Internet Society.

DAMGÅRD, I., FITZI, M., KILTZ, E., NIELSEN, J., AND TOFT, T. 2006. Unconditionally secure constant-rounds multiparty computation for equality, comparison, bits and exponentiation. In *Proceedings of the Theory of Cryptography Conference (TCC)*.

DAMGÅRD, I., GEISLER, M., KRØIGAARD, M., AND NIELSEN, J. 2009. Asynchronous multiparty computation: Theory and implementation. In *Proceedings of the Conference on Practice and Theory in Public Key Cryptography (PKC)*.

DAMGÅRD, I., MELDGAARD, S., AND NIELSEN, J. B. 2011. Perfectly Secure Oblivious RAM Without Random Oracles. In *Proceedings of the Theory of Cryptography Conference (TCC)*.

DUAN, Y. 2009. Differential privacy for sum queries without external noise. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*.

DWORK, C. 2008. Differential privacy: A survey of results. In *Proceedings of the Conference on Theory and Applications of Models of Computation (TAMC)*.

FAGIN, R. 1996. Combining fuzzy information from multiple systems. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

FAGIN, R., LOTEM, A., AND NAOR, M. 2001. Optimal aggregation algorithms for middleware. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*.

FREEDMAN, M. J., NISSIM, K., AND PINKAS, B. 2004. Efficient private matching and set intersection. In *Proceedings of the EUROCRYPT*. Lecture Notes in Computer Science, vol. 3027, Springer Berlin, 1–19.

GENNARO, R., ISHAI, Y., KUSHILEVITZ, E., AND RABIN, T. 2002. On 2-round secure multiparty computation. In *Proceedings of CRYPTO*.

GENNARO, R., RABIN, M., AND RABIN, T. 1998. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*.

GOLDREICH, O., MICALI, S., AND WIGDERSON, A. 1987. How to play any mental game. In *Proceedings of the ACM Symposium on the Theory of Computing (STOC)*.

LAKHINA, A., CROVELLA, M., AND DIOT, C. 2005. Mining anomalies using traffic feature distributions. In *Proceedings of the ACM SIGCOMM Data Communications Festival*.

LEE, A. J., TABRIZ, P., AND BORISOV, N. 2006. A privacy-preserving interdomain audit framework. In *Proceedings of the Workshop on Privacy in Electronic Society (WPES)*.

LI, X., BIAN, F., CROVELLA, M., DIOT, C., GOVINDAN, R., IANNACCONE, G., AND LAKHINA, A. 2006. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the Internet Measurement Conference (IMC)*.

LINCOLN, P., PORRAS, P., AND SHMATIKOV, V. 2004. Privacy-preserving sharing and correlation of security alerts. In *Proceedings of the 13th USENIX Security Symposium*.

MACHIRAJU, S. AND KATZ, R. H. 2004. Verifying global invariants in multi-provider distributed systems. In *Proceedings of the SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. ACM.

MARIAN, A., BRUNO, N., AND GRAVANO, L. 2004. Evaluating top-k queries over web-accessible databases. *ACM Trans. Datab. Syst. 29,* 2, 319–362.

MCSHERRY, F. AND MAHAJAN, R. 2010. Differentially-private network trace analysis. In *Proceedings of the ACM SIGCOMM Data Communications Festival*.

NISHIDE, T. AND OHTA, K. 2007. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Proceedings of the Conference on Theory and Practice of Public Key Cryptography (PKC)*.

PAREKH, J. J., WANG, K., AND STOLFO, S. J. 2006. Privacy-preserving payload-based correlation for accurate malicious traffic detection. In *Proceedings of the ACM Workshop on Large-Scale Attack Defense (LSAD)*.

RANJAN, S., SHAH, S., NUCCI, A., MUNAFÒ, M. M., CRUZ, R. L., AND MUTHUKRISHNAN, S. M. 2007. Dowitcher: Effective worm detection and containment in the internet core. In *Proceedings of INFOCOM*.

RINGBERG, H. 2009. Privacy-preserving collaborative anomaly detection. Ph.D. thesis, Princeton University.

ROSSI, D., MELLIA, M., AND MEO, M. 2009. Understanding Skype signaling. *Comput. Netw. 53,* 2, 130–140.

ROUGHAN, M. AND ZHANG, Y. 2006a. Privacy-preserving performance measurements. In *Proceedings of the SIGCOMM Workshop on Mining Network Data (MineNet)*.

ROUGHAN, M. AND ZHANG, Y. 2006b. Secure distributed data-mining and its application to large-scale network measurements. *Comput. Comm. Rev. 36,* 1, 7–14.

SANG, Y., SHEN, H., TAN, Y., AND XIONG, N. 2006. Efficient protocols for privacy preserving matching against distributed datasets. In *Proceedings of the Conference on Information and Communications Security (ICICS)*.

SHAMIR, A. 1979. How to share a secret. *Comm. ACM 22,* 11, 612–613.

SHMATIKOV, V. AND WANG, M. 2007. Security against probe-response attacks in collaborative intrusion detection. In *Proceedings of the ACM Workshop on Large-scale Attack Defense (LSAD)*.

STOLFO, S. J. 2004. Worm and attack early warning. *IEEE Secur. Priv. 2,* 3, 73–75.

SWITCH. The Swiss education and research network. http://www.switch.ch.

TARIQ, M. B., MOTIWALA, M., FEAMSTER, N., AND AMMAR, M. 2009. Detecting network neutrality violations with causal inference. In *Proceedings of the Conference on Emerging Networking Experiments and Technologies (CoNEXT)*.

TELLENBACH, B., BURKHART, M., SCHATZMANN, D., GUGELMANN, D., AND SORNETTE, D. 2011. Accurate network anomaly classification with generalized entropy metrics. *Comput. Netw. 55,* 15, 3485–3502.

VAIDYA, J. AND CLIFTON, C. 2005. Privacy-preserving top-k queries. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*.

VAIDYA, J. AND CLIFTON, C. 2009. Privacy-preserving kth element score over vertically partitioned data. *IEEE Trans. Knowl. Data 21,* 2, 253–258.

XIONG, L., CHITTI, S., AND LIU, L. 2005. Topk queries across multiple private databases. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*.

YAO, A. 1982. Protocols for secure computations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*.

YEGNESWARAN, V., BARFORD, P., AND JHA, S. 2004. Global intrusion detection in the DOMINO overlay system. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Received October 2010; revised March 2011; accepted July 2011