Elementary secure-multiparty computation for massive-scale collaborative network monitoring: A quantitative assessment

A. Iacovazzi a,*, A. D’Alconzo b, F. Ricciato c, M. Burkhart d

a DIET, Sapienza University of Rome, Via Eudossiana 18, 00184 Rome, Italy
b FTW – Forschungszentrum Telekommunikation Wien, Donau-City-St. 1, 1220 Vienna, Austria
c DII, University of Salento, Campus Ecotekne, Via per Monteroni, 73100 Lecce, Italy
d Department of Computer Science, ETH, Universitätstrasse 6, 8092 Zurich, Switzerland

1. Introduction and motivations

Since its inception the Internet has been exposed to global threats: spam, large-scale malware infections, DDoS attacks and botnets are all examples of global phenomena insensitive to any administrative network boundary. Besides threats, the popularity of global Over-The-Top (OTT) services and peer-to-peer applications has increased the risk of “global failures” that impact customers and networks of multiple ISPs, e.g., like the worldwide Skype outage in 2007 [1,2].

Despite the global nature of threats and failures, the operation and management of the network infrastructure remains almost entirely localized within each ISP’s domain, and so do the detection, prevention and reaction processes. The contrast between global problems and local response plays heavily in favor of the former. Most operators concede that some degree of coordination (and

* Corresponding author. Tel.: +39 0644585365; fax: +39 064744481.
E-mail addresses: alfonso.iacovazzi@uniroma1.it (A. Iacovazzi), a.dalconzo@ftw.at (A. D’Alconzo), fabio.ricciato@unisalento.it (F. Ricciato), burkhart@tik.ee.ethz.ch (M. Burkhart).
collaboration) across ISPs, at least in the stage of detecting and diagnosing the problem, would be highly beneficial. The simplest use-case would be to enable each ISP to complement the detailed view of its own “internal” network, obtained by the local monitoring process, with a condensed view of the “external” situation. The combination of the two views would improve the effectiveness of the alarming and troubleshooting process along several dimensions: lower rates of false positives, lower delay, lower cost. More advanced forms of inter-domain collaboration could involve sharing malware information (e.g., with a newly learned signature) or the coordinated activation of local countermeasures (e.g., new firewall rules).

In order to be accepted by ISPs any form of collaborative model must fulfill some fundamental requirements. First, ISPs will not share their raw data due to business sensitivity and/or user privacy regulations. Second, they will want to preserve their anonymity when it comes to disclosing information about critical events that have impacted their domain like failures and/or attacks.

Recently, Secure-Multiparty Computation (SMC) has been proposed as an approach to enable inter-domain network monitoring while protecting the data of individual ISPs [3,4]. With SMC the collaboration paradigm shifts from “local computation on shared data” to “shared computation on local data”. The SMC family includes many different techniques and variants, featuring different forms of “security”, i.e., against different types of attack (er) and with different levels of computation complexity and communication overhead. In the context of collaborative network monitoring, the rate and volume of network data to be (securely) processed is massive, and the number of participating players might be large, therefore scalability is a primary requirement. To preserve scalability one must sacrifice other requirements, like verifiability and computational completeness that, however, do not appear to be critical in this context. In fact, since SMC players map to ISPs, it is reasonable to exclude the presence of “active attackers” and assume that all players follow the “honest-but-curious” model. Therefore, we restrict the focus onto non-verifiable techniques that are much simpler and scalable than verifiable ones.

In a previous work [5] (see also the extended version [6]) we have shown that any “Elementary SMC” (E-SMC) scheme that supports only simple additions with private inputs and public output is sufficient to support a set of primitive operations that are likely relevant for inter-ISP collaboration, e.g., Conditional Counting, Voting, Histogramming, Set Union, Anonymous Publishing and even Anonymous Scheduling. The point made in [5,6] is that private addition can become very powerful when combined with local transformations of the inner data, e.g., involving probabilistic data structures like Bloom filters and bitmaps. Whenever intermediate results – which are necessarily public in E-SMC – are not regarded as sensitive, such primitives can be chained into structured “private workflows” that safeguard the privacy of the input data as well as the anonymity of each player. We claim that a large part, if not all, of the procedures needed to support collaborative inter-domain network monitoring can be reduced to elementary secure additions.

Given this framework, the central design problem reduces to finding the most scalable way to implement elementary secure additions. In this paper we consider two possible schemes: the Shamir’s Secret Sharing (SSS), based on polynomial interpolation on prime fields, and the Globally-Constrained Randomization (GCR) scheme based on simple blinding [5]. The goal of this paper is to address the system-level aspects and quantify the achievable performance of both schemes. An attractive system-level feature of GCR is the possibility of pushing all the communication and processing overhead into a preliminary offline preparation phase, leaving the online computation phase as fast and lightweight as a cleartext addition. In order to compare quantitatively the performance of the two schemes in a fair way, we have implemented a prototype version of GCR in SEPIA [4], an open-source platform that supports SSS natively, and then performed a number of controlled experiments in emulated scenarios.

The contributions of this work are:

1. We discuss a number of system-design features of GCR that enable massive-scale implementation. That is, how to split the computation into offline randomization and online aggregation phases, and how to efficiently handle joining/leaving of players.
2. We assess the sensitivity of GCR performance to a number of system design parameters, as well as to the network conditions.
3. We compare quantitatively the performance of a GCR-based implementation of additive E-SMC versus a SSS-based implementation.
4. We investigate the resilience of the GCR scheme to node failures by leveraging theoretical analysis and emulation results.

The rest of this paper is organized as follows. Section 2 describes the reference scenario and the assumed adversary model. We review the GCR scheme and its features in Section 3. Section 4 contrasts GCR and SSS from a theoretical point of view. Sections 5 and 6, illustrate the implementation of GCR within SEPIA and the emulation setup, respectively. In Section 7 we assess the dependency of the GCR performance from system parameters and network conditions, and we contrast it with the performance attained by SSS. In Section 8 we investigate the impact of players fault on the GCR performance. Finally, related work is discussed in Section 9, and in Section 10 we summarize our conclusions.

2. Reference scenario

In the collaborative inter-ISP scenario, a set of ISPs holds a set of monitored data collected locally, like e.g., traffic statistics, network logs, records of security incidents. Based on these data, each ISP performs statistical and behavioral analysis of the hosts interacting with its network and to identify possible threats such as spam campaigns, worms spread-out, and Distributed Denial of Service (DDoS) attacks. Unfortunately, each ISP holds only partial information corresponding to its particular viewpoint inside the global Internet. As pointed out already in [4], each ISP
would benefit from comparing its own local view of traffic conditions with the global view aggregated over all other ISPs, especially in case of anomalies and alarms, in order to understand whether the (unknown) root cause is local or global – a major discriminant for deciding the reaction strategy. Also, ISPs might be ready to share with other ISPs information about security incidents observed locally (e.g., malware signatures) provided that they can do so anonymously. Another possible use-case is the sharing of aggregated contact statistics for each DNS domain, from which domain-fluxing botnet servers and/or suspicious malware domains can be revealed. For example, it was noted in [7] that the combination of different datasets would improve the detection power of the bot-cluster identification algorithm proposed therein due to (i) larger scope of data and (ii) data diversity. For some of these use-cases, the collaborative computation system must sustain a high-rate of secure operations. If the output is used to trigger countermeasures, the computation delay and real-time response become also critical.

In the field of SMC two main adversary models are considered: malicious, that allows for active attacks, and semi-honest (also known as honest-but-curious) where only passive attacks are considered.

In the malicious model the adversaries have the ability to take full control of the corrupted parties. They can arbitrarily deviate from the correct behavior and carry out attacks inside the protocol, e.g., using erroneous inputs, force to output wrong and piloted answers and abort before the end of protocol. In the semi-honest model all parties run the protocol diligently and cooperate honestly to compute the final result, but a subset of them (possibly corrupted by an adversary) may combine information they see during the protocol execution, in order to infer private information of the other players. In other words, no malicious player will attempt to neither interrupt nor corrupt the computation process, e.g., by providing incorrect input data. Protocols robust to malicious adversaries are more complex and computationally expensive [8] and do not seem justified in the context of collaborative network monitoring where players map to ISPs, i.e., entities that do not have any clear incentive to boycott the computation process. In the context of cooperative network monitoring, where ISPs base their relationship on precise agreements, we can reasonably assume the semi-honest model. Nonetheless, a subset of ISP may decide to collude and exploit the results from the protocol executions in order to infer sensitive and private information belonging to another ISP. Given this framework, the adoption of an SMC technique assures that no unauthorized information about the input values can be learned by the parties – except for what they can already infer from their own input plus the public output – given that the number of colluding players is below a given threshold.

3. The GCR scheme

3.1. Notation

We consider a set of $N$ players $\{P_i, i = 1, \ldots, N\}$ with $N \geq 3$ (normally $N \gg 1$) each following the honest-but-curious model. Each player $P_i$ has a private input $a_i$ defined in some small-field (e.g., 32/64-bit scalar, binary string, array of $q$-bit counters) and the goal is to compute the public output $A = \sum a_i$ without disclosing the value of $a_i$ nor the identity of the player $P_i$. To achieve that, each player builds a random element (RE) $y$ defined in the same field as $a_i$, in a way that ensures the zero-sum condition $\sum_{i=1}^{N} y_i = 0$. The latter condition motivates the term Globally-Constrained Randomization to refer to this scheme [5]. The value of $y_i$ cannot be known by another player as far as the number of colluders remains below a threshold $\ell$ (with $\ell < N$). The colluding threshold $\ell$ is a system-design parameter that can be set independently from the system size $N$. The set of REs across all players is called Random Set (RS) and is denoted hereafter by $r = [r_i, i = 1, \ldots, N]$.

3.2. RS generation

The central aspect of GCR is that the RS is constructed in a way that guarantees the zero-sum condition, i.e., the composition of random elements across all players sums up to the null element. For this purpose each player $P_i$ ($i = 1, \ldots, N$) must construct its RE $y_i$ in cooperation with other players. In other words the RS $r$ is built collectively by all player. We remark that the RS generation procedure can be run in parallel by all players and is completely asynchronous. Each random element is initially set to the null element, i.e., $r_i = 0$. Each player $P_i$ extracts $\ell + 1$ random variables $x_{ij}$ ($j = 1, \ldots, \ell + 1$) and computes their sum $y_i = \sum x_{ij}$. It calculates the additive inverse\(^1\) of $y_i$, denoted by $\overline{y}_i$, and adds the latter to its own random element, i.e., $y_i \leftarrow r_i + \overline{y}_i$. At the same time, $P_i$ contacts $\ell + 1$ randomly selected other players and sends one variable $x_{ij}$ to each of them: each contacted player $P_j$ will then increment its random element by $x_{ij}$, i.e., $y_j \leftarrow y_j + x_{ij}$. This method is secure against collusion of up to $\ell$ players. Notably, the value of $\ell$ is a free parameter, independent from the system size $N$, which can be tuned to trade-off communication overhead with robustness to collusion, both scaling linearly in $\ell$.

3.3. Computation phase

To sum their private inputs, each player computes the public input element $v_i = a_i + r_i$ and sends it to the central collector. The RE $y_i$ protects the value of $a_i$, which cannot be derived from the public element $v_i$ (blinding). The private inputs $a_i$ are derived from the inner private data $x_i$ by some function $a_i = g(x_i)$. The function $g(\cdot)$ can involve probabilistic data structures (e.g. Bloom Filters, bitmap), encryption, randomization, etc. (see [6] for further details). More complex operations and workflows on private data can be performed by chaining multiple GCR computation (additions), where the results from one computation are taken as (public) arguments for the following one. No particular constraint applies to the aggregation method which can be centralized or distributed. For the sake of simplicity, we assume in the following a fully centralized scheme,

---

1 In modular arithmetic the additive inverse $y$ of $y$ is the element that satisfies $y + y = 0$. For real numbers in $(0, p), y = p - y$, while for binary strings $y = \overline{y}$. 

3730 A. Iacovazzi et al. / Computer Networks 57 (2013) 3728–3742
with a single master – not necessarily a player – that is in charge of collecting the $N$ public inputs, computing the result and finally publishing it to all players. Note however that the central collector does not have special trust endowment: it is as honest and curious as any other player.

In the following we address some system-level aspects of GCR.

### 3.4. Offline generation of random sets

One key advantage of GCR is that the process of generating the RS is decoupled from, and can be run in parallel to, the actual computation round. This has important implications for the design of a massive-scale system, enabling efficient management of the communication load and minimal response delay. Therefore we designed the system such that the RSs are generated offline and stored for later use. At any time, each player $P_i$ has available a collection of random elements $r_i[u]$, indexed in $u$, which can be readily used for future computation rounds. The communication protocol ensures that the RSs indexing is universal and synchronized across all players, and that during the online computation phase the same RS index is used by all players.

Performing RS generation offline brings several advantages. First, it minimizes GCR’s addition times down to the same value of an equivalent cleartext summation. Second, it allows to reduce the impact of communication overhead onto the network load by scheduling the RS generation process in periods when the online computation is idle and network load is low (e.g., at night or week-end).

### 3.5. Batching

Generation of multiple RSs can be made more efficient by using batching: in a single secure connection (typically SSL over TCP) players can exchange multiple (variable, index) pairs $\{x_j[u], u\}$ that collectively build a collection of RSs $\{r[u]\}$. This greatly reduces the communication overhead associated to connection establishment: hand-shaking, authentication, key exchange, etc. On the other hand, if the subset of players receiving the batch of random elements colludes, they may be able to reveal an entire batch of private inputs. Therefore, the batch size is a design parameter set trading-off robustness (to collusion) for communication overhead.

For similar reasons, also the online computation additions will not be performed in isolation, but in groups. We will use the term “round size” to denote the number of parallel additions performed in a single computation round, keeping the term “batch size” reserved for the offline computation phase.

### 3.6. Joining and leaving

In the GCR scheme, the set of players participating in the computation round must match exactly the set of players that have previously built the RS: the final result will not be reconstructed if the two sets differ by even a single element. If RSs are generated offline, the set of players might have changed during the interval between the generation of $r[u]$ and its consumption in a query. It would be very impractical to trash all pre-computed RSs upon every new player joining or leaving – an event not infrequent in large systems with many players. Fortunately this is not necessary and each legacy RS can be incrementally adjusted upon new join or leave with only $\ell + 1$ operations.

When a new player $P_j$ joins the system, it learns the index range currently in use $\{u_1, \ldots, u_2\}$ – note that this information is public – and computes a set of random variables $x_j[u]$ for $j = 1, \ldots, \ell + 1$ and $u \in \{u_1, \ldots, u_2\}$. Then it sets its local random elements as $r_j[u] = y_j[u]$ (recall that $y_i = \sum_{j=1}^{\ell+1} x_j[u]$). Then for each index value $k$ it selects $\ell + 1$ other players to which it sends the individual variables $x_j[u]$. Similarly, when an existing player $P_i$ wants to leave the system, it must first “release” its random elements $r_j[u]$. The simplest way to accomplish that is to simply pass the value of $r_j[u]$ to another randomly selected player $P_j$ and let the latter update its local random element as $r_j[u] \leftarrow r_j[u] + r_i[u]$.

### 4. GCR versus Shamir’s

#### 4.1. The SSS scheme

In SSS the secret input $a_i$ of the $i$th player is shared among a set of $M$ players by generating a random polynomial $f$ of degree $t < M$ over a prime field $\mathbb{Z}_p$, with $p > a_i$, such that $f(0) = a_i$. Each player $j = 1, \ldots, M$ then receives an evaluation point $s_j = f(s_j)$, called the share of player $i$. The secret $a_i$ can be reconstructed from any $t + 1$ shares using Lagrange interpolation but is completely undefined for $t$ or less shares. Because SSS is linear, addition of two shared secrets can be computed by having each player locally add his shares of the two values. Multiplication of two shares requires an extra round of communication among the $M$ players. Finally, to actually reconstruct a secret, each of the $M$ players sends his shares to all other players. Each player then locally interpolates the secret and finally returns the computation result to the input players.

#### 4.2. Advantages of SSS over GCR

There are two fundamental advantages of SSS over GCR. First, the basic operations accept public, private, and also secret input data and output secret data.2 That is, even without reconstructing intermediate values, it is possible to arbitrarily compose secret operations.

The second advantage of SSS is that it realizes a $(t + 1)$-out-of-$N$ threshold sharing scheme. That is, any set of $t + 1$ players can reconstruct a secret, being robust against up to $N - t - 1$ “missing” players. In GCR instead, a single non-responsive player renders the reconstruction of secret information impossible, i.e. GCR realizes only a $N$-out-of-$N$ scheme. For this reason player failures occurring between the RS generation and computation phases are

---

2 The notions of secret and private are distinct: private data is known in cleartext to at least one player (and usually only to one), while secret data remains unknown by all players and cannot be reconstructed unless a minimum number of players agree to do so.
critical in GCR. This problem is discussed more in detail later in Section 8 where also a quantitative analysis is given.

### 4.3. Advantages of GCR over SSS

GCR is highly optimized for online processing of queries, since all the communication and processing overhead can be pushed to a separate offline phase. When processing the query (online computation) GCR involves minimal communication overhead, since the players just send their randomized values instead of the original value to the aggregation node(s). In SSS, when N players want to sum their values, each of them generates N shares ad hoc and distributes them to the others. In principle, the players could pre-generate t random shares and distribute them in a pre-processing phase. In the online phase, they would calculate the remaining N – t shares using Lagrange interpolation, such that the interpolated polynomials represent their actual secrets. However, after distributing the last shares, each player still needs to perform N – 1 additions locally, and for the final reconstruction send their shares of the sum to the aggregation node(s), which eventually interpolates the final polynomial. It is not obvious how to further split this process into an offline pre-processing and an online phase similar to GCR, where a single message and addition operation is enough.

Another advantage of GCR is that the additive scheme is not restricted to prime fields. This allows to set the field size to $2^{32}$ or $2^{54}$ and therefore to use implicit 32 (64) bit register wrap-arounds of CPU operations instead of performing an explicit modulo operation.\(^3\) Also, SSS requires linear storage overhead (N shares to be stored for each secret value), whereas GCR has constant storage overhead (one random value per private input).

In summary, provided that intermediate results are not sensitive, GCR allows for a much smaller storage and by reducing the overhead, to attain a higher computation rate during the online processing phase. In Section 7 we quantify this gain via experiments in an emulated testbed.

### 5. Implementation of GCR in SEPIA

GCR we have implemented in Java, as an extension to the SEPIA, a software platform implementing natively SSS. SEPIA provides a set of functionalities for efficient execution of several primitive operations, as well as for the development of entire protocols in the secure space [4]. In particular, thanks to grouping of operations in rounds, SEPIA attains significantly higher performance than other general-purpose SMC tools such as e.g., FairplayMP [9], and VIFF v0.7.1 [10].

The benefit of implementing GCR as an extension of SEPIA is twofold. First, it allows a fair comparison between the performance of the two schemes. In fact, the adoption of the same software platform guarantees that the mechanisms for handling communication and message passing between entities are the same, and therefore they have the same impact on the overall performance. Second, it allows to use SSS and GCR in combination, opting for one or the other scheme depending on the particular operation and use case.

In SEPIA two distinct types of entities are considered: Input Peers (IPs) and Privacy Peers (PPs). In a general scenario, N IPs own the private data (input for computation) and a group of M PPs performs the secure computation of the shares received by the IPs. Notice that PPs can be run by a subset of IP players, as well as by third parties. The logical topology of SEPIA-SSS is reported in Fig. 1(a); each IP sends M shares to the PP.

Similarly, in the SEPIA-GCR implementation we have two entities. Each player participating to the computation runs an Input Node (IN), whereas a distinct entity referred to as Collector Node (CN) computes the additions of the public inputs received by the INs. Also in this case the CN can be run by one of the players already running the IN, as well as by third party. The logical topology of GCR is reported in Fig. 1(b).

The communication channels between all the nodes (IN, CN, IP, PP) are encrypted and certificates provided by SEPIA are used to establish SSL channels over TCP. Information about the computation status and the final result are written in log and output files, respectively. The items to be processed in a single round are read at once for the same file, they must be of the same data type (i.e., integers, reals, or binary strings, coded by 64 bits) and are formatted as comma-separated values.

The SEPIA-GCR implementation consists of two distinct processes for (offline) RS generation and (online) computation. The production process generates offline the RSs and is run only on INs. The consumption process uses the available RSs for the online computation, and involves the INs plus the CN. The generation process is run at lower priority than the consumption process, in order not to limit the IN processing speed during the online phase. When an IN is started, its consumption process checks the availability of RSs generated with the same set of active players. Also, it checks whether the number of accumulated RSs is sufficient to cover an entire computation round. If both these conditions are fulfilled, then the consumption process starts, otherwise it is put on hold until the production process generates a sufficient number of RSs. The same mechanism is used to coordinate the two processes in case of a player fault which invalidates all stored RSs. The production process enters the idle state as soon as the number of buffered RSs reaches a maximum size configured according to the available memory resources.

### 6. Testbed and emulation scenario

The testbed used for running the emulations consists of four workstations connected by a dedicated Gigabit-Ethernet switch. Three workstations were equipped with a 4 x core i5 CPU @2.8 GHz, with 4 GB memory, and the fourth is an Intel Xeon server @3.2 GHz, with 10 GB memory. Depending on the emulation purpose, we used either only the server or all the four machines. In particular, due to memory and processing power limitations, we used

\(^3\) In general, $\text{mod}(a, N) = a - N \cdot \text{floor}(a/N)$, which uses an additional division, multiplication, and subtraction operation.
the distributed emulation testbed when investigating how the system scales with the number of players and the number of privacy peers. On the other hand, we resorted to single machine emulation when investigating performance against network bandwidth.

We used the Common Open Research Emulator (CORE) tool for emulating virtual networks and hosts in our testbed. CORE builds a “lightweight” representation of a computer network that runs in real time and allows to connect emulated to real networks. Furthermore, it is possible to run real applications (such as SEPIA) and protocols on each emulated host by exploiting virtualization provided by Linux or FreeBSD operating systems. This allows replicating only the network stack and the functions strictly necessary for emulation, avoiding replication of the entire OS image with considerable saving of memory space. This feature makes CORE particularly attractive for emulating large scale networks on commodity hardware. For further details about CORE’s features we refer the reader to [11, 12].

Fig. 1 depicts the mapping of the logical schemes of Fig. 1 to the emulated hosts, and the mapping of the emulated network to the physical machines, when the four workstations are used. The core of the emulated network consists of four fully-meshed virtual routers, plus four border routers with attached emulated hosts. All the emulated routers have sending/receiving buffers of infinite size. For SEPIA-SSS the IPs – evenly distributed among three workstations – are attached to the uppermost router of the core-network via three border routers one for each physical machine (as depicted in Fig. 2(a)). The PPs are connected to the three remaining core routers via three border routers. Similarly, Fig. 2(b) shows the topology used for the GCR emulations. It is evident that, from a topological point of view, GCR topology can be considered as an extreme case of SSS topology with only one PP. Note that we have been running a single SEPIA instance per each emulated host.

The reason for choosing such a network topology is that it allows to adjust independently the Round-Trip Times (RTT) between different types of nodes: IP–PP, PP–PP (for SSS), IN–IN and IN–CN (for GCR). Furthermore, the network topology is mapped to the physical machines in a way to minimize the number of virtual links connecting the different physical machines (dotted lines in the Fig. 2). In fact, packets sent over these links are actually transmitted through the Ethernet interfaces – hence they consume bandwidth resources of the physical LAN used in the testbed – whereas packets transmitted over links connecting virtual hosts emulated on the same machine are just handled in the system memory. This aspect is critical when all the machines are used, as for example in the scenarios with a large number of players. For the same reason, when investigating the performance in scenarios requiring more than 1 GB of bandwidth, we had necessarily to resort to single-machine emulation. In this case the topologies of Figs. 2 had been entirely mapped to one physical machine (i.e., the Xeon server), scaling-down the number of IPs/INs to cope with the more stringent memory and processing power constraints.

7. Performance evaluation

In this section we first report about the performance of the offline RS generation of the SEPIA-GCR implementation. We also show the performance of the online computation phase, and finally we compare it with SEPIA-SSS.

For GCR we have varied the number of INs in the range $[5, 90]$, whereas the batch size $n_b$ and the round size $n_r$ were varied within the ranges reported in Table 1. We investigated the effect of several network conditions by changing link bandwidth and delay so as to obtain different Round-Trip Times (RTTs). Hereafter for GCR we indicate by $RTT_{IN\rightarrow IN}$ and $RTT_{IN\rightarrow CN}$ the maximum RTT between INs and CN and between the INs, respectively. Also, we indicate by $BW_{IN\rightarrow CN}$ and $BW_{IN\rightarrow IN}$ the available bandwidth between INs and the CN and between the INs, respectively. For SEPIA-SSS we indicate by $RTT_{IP\rightarrow PP}$ and $RTT_{PP\rightarrow PP}$ the maximum RTT between IPs and PPs and between the PPs, respectively, and by $BW_{IP\rightarrow PP}$ the available bandwidth between the IPs and the PPs. The range of variability for each parameter is reported in Table 1.

For SEPIA-SSS experiments have been performed by varying the number of PPs within the range $[5, 30]$. Since PPs must be operated by distinct administrative domains for guaranteeing protocol security, in practice it is reasonable to expect that they will be located in geographically distant sites. This condition has been investigated by varying the $RTT_{PP\rightarrow PP}$ values.
The following results refer to the execution of the private addition of integers. The performance for the online phase are measured in terms of average rate, calculated as the ratio of the number of items in the round over the total time required for the execution of the round. Similarly, for the offline phase performance are calculated as the ratio of the number of batch items over the total time required for generating a batch. For each point in the plots we report the average over 10 iterations and the error bar representing the minimum and maximum observed values.

### 7.1. Speed of the random-set generation (offline phase, production process)

The performance of the offline generation phase depends mainly on the interplay of three factors: (i) on the number of random elements $x_{ij}$ to be generated from each IN, which in turn depends on the collusion threshold $\ell$, (ii) on the batch size, which determines the amount of REs exchanged in a single message with the same set of randomly selected INs (see Section 3.4, for details), (iii) on the maximum $RTT_{IN-IN}$ and on the available bandwidth between each pair of INs.
In addition to that, when emulating the offline generation phase, we have to take into account the constraints imposed by the testbed. In this regard, we can distinguish three aspects: the processing power available for each emulated host, the bandwidth of the links connecting the testbed machines, and the overall system memory. 

In fact, each IN establishes connections with \( i + 1 \) INs to send the locally generated random elements; the batch size determines the number of elements exchanged in a single message. Hence, the overall load on the network is proportional to \( \ell \cdot n_g \cdot N \). By scaling-up one of these factors, we can saturate the link bandwidth of the testbed (i.e., 1 Gb per link). This is indeed the case when investigating the dependency of the offline RS generation on the batch size. In order to avoid that, we have to resort to single-machine emulations. On the other hand, the performance of the offline generation depends also on the processing power dedicated to each emulated host. Hence, we had to keep the number of INs small to avoid exhaustion of processing resources.

In order to support the interpretation of the experimental results it is convenient modelling the components contributing to the generation time \( T_B \) of a batch of \( n_g \) RSs. A first component is due to the computation time \( \tau_c \) needed on the IN for extracting the random numbers used for the calculation of one RE. The second component comes from the exchange of the locally generated random numbers with the others INs. Since in our emulation setup there is no queuing latency, the communication time consists of a propagation component proportional to the round trip time \( RTT_{IN-IN} \) plus a transmission component proportional to the message size (i.e., \( x \cdot n_g \)) and inversely proportional to the link bandwidth. Finally, the last component is the time \( \tau_s \) spent by the IN for calculating the RE. Therefore, the total batch generation time can be expressed as:

\[
T_B = n_g \cdot \tau_c + \frac{x \cdot n_g}{BW_{IN-IN}} + \beta \cdot RTT_{IN-IN} + n_g \cdot \tau_s,
\]

where \( x \) and \( \beta \) are proportionality factors. By definition, the RS generation rate is:

\[
R_g \overset{\text{def}}{=} \frac{n_g}{T_B} = \frac{1}{(\tau_c + \tau_s + \frac{x}{BW_{IN-IN}}) + \frac{\beta \cdot RTT_{IN-IN}}{n_g}}.
\]  

Fig. 3(a) shows the average RS generation rate as function of the batch size \( n_g = [5, 10, 20, 50, 100] \times 10^2 \), for different values of \( RTT_{IN-IN} \). The results refer to a single-machine scenario where five emulated INs are connected by virtual links with unlimited bandwidth, and the collusion threshold has been set to \( \ell = 4 \). The limit for Eq. (2) for \( BW_{IN-IN} \to \infty \) is:

\[
\lim_{BW_{IN-IN} \to \infty} R_g = \frac{1}{(\tau_c + \tau_s) + \frac{\beta \cdot RTT_{IN-IN}}{n_g}}.
\]

When also the propagation delay is negligible (i.e., \( RTT_{IN-IN} \to 0 \)) already for moderately small \( n_g \) values the generation rate approaches the value \( 1/(\tau_c + \tau_s) \), which only depends on the computational speed of the IN. This is the situation depicted by the solid line in Fig. 3(a) where \( R_g \) approaches \( 10^5 \) for \( n_g > 5000 \). In other words, when the communication time is negligible, the gain from batching more than 5000 RSs is marginal. The dashed curves in Fig. 3(a) shows that when the RTTs increases the generation rate reduces, independently from the \( n_g \) value. This is easily explained by the contribution of the term proportional to the RTT at the denominator of the Eq. (3). Note also that these two curves asymptotically tend to \( 1/(\tau_c + \tau_s) \). However, they approach this upper bound for values of \( n_g \) too large to be experimented in our testbed because of the memory limitation of the used machine. Finally, the values of \( R_g \) depend on the processing speed of each emulated host (i.e., \( \tau_c \) and \( \tau_s \)), which in turn depend on the speed of the physical machine used in the emulation. Thus, in a real setup the performance can be further scaled-up by increasing the computational resources allocated to the offline generation phase.

In Figs. 4(a) we show the results for the same experiment in a scenario with 30 emulated INs on three machines, where each IN is provided with 10 Mb/s network bandwidth. Even though results show the same qualitative dependency of the generation rate on the batch size, the absolute values are rescaled by one order of magnitude because of the bandwidth limitation. This is easily explained by the contribution of the term inversely proportional to the bandwidth at the denominator of Eq. (2).

For the experiment reported in Fig. 4(b) we set \( n_g = 10^4 \) while changing the collusion threshold, which is a design parameter controlling the GCR scheme robustness-to-collusion. The experimental results reveal that the dependency of the RS generation rate from the set collusion threshold is moderate.

### 7.2. Online computation

Similarly to the offline generation phase, for the online computation the physical bandwidth of the emulation testbed may become a limiting factor when investigating the relationship between computation rate and round size. Therefore, for investigating the maximum achievable performance, also in this case we used a single machine with five emulated INs and unlimited virtual bandwidth. Fig. 3(b) shows the trend of the GCR computation rate as function of the round size \( n_r \), for different values of \( RTT_{IN-C} \). Here we can note that when \( RTT_{IN-C} = 5 \) ms and the round size varies from \( 10^4 \) to \( 4 \times 10^4 \), the computation rate increases from \( 2 \times 10^3 \) to \( 12 \times 10^5 \) operations per second. Fig. 3(b) shows also that the computation rate decreases considerably for larger RTTs: for example it reduces to about \( 2.2 \times 10^4 \) operations per second, for rounds of \( 10^5 \) items, and \( RTT_{IN-C} = 200 \) ms. The explanation of such a behavior follows the same line of reasoning as for the offline generation phase, and an expression similar to Eq. (2) can be derived by considering the opportune variable changes (i.e., \( RTT_{IN-C} \) and \( BW_{IN-C} \)). Finally, Fig. 3(b) shows that also for the online computation phase the overhead due to the communication time can be reduced by increasing the number of items per round.

By comparing the Fig. 3(a) and (b) it is worth noting that, when the network conditions (i.e., RTT and bandwidth) between the INs and between the INs and the CN are similar, the RS generation and the computation phases attain similar rates. That is, the online computation phase...
consumes the RSs at a speed comparable with the generation one. Hence, the online computation can run at the maximum speed without getting blocked by the RSs generation process, and the overall time for a secure summation reduces to the same value as for a clear-text summation.

Fig. 5(a) shows how computation rates change with the round size, for different values of the bandwidth between the INs and the CN, when \( RTT_{IN-CN} = 5 \text{ ms} \). The Figure tells that it is worth grouping operations in round larger than \( 10^5 \) items only if the speed of the links between the INs and the CN is larger than 10 Mb/s, otherwise batching does not result in any significant performance gain. Similarly, Fig. 5(b) investigates the relationship between the computation rate and the available bandwidth, for different values of \( RTT_{IN-CN} \), and for \( n_r = 4 \times 10^5 \) elements. From this figure we can conclude that providing larger bandwidth results in higher computation rates only if \( RTT_{IN-CN} \) is below 100 ms.

In Fig. 6(a) we have repeated the same experiments as in Fig. 3(b), but on the distributed platform of four machines depicted in Fig. 2(b). In this setup we have considered 30 INs, and the bandwidth between each IN and the CN has been limited to 10 Mb/s. As expected the computation rate reduces by one order of magnitude because of the bandwidth limitation. Notably, also in the distributed case, the rate attained by the online computation phase is comparable with the rate of the offline RS generation (cf. Fig. 4(a)). Therefore, the same conclusions as for the single machine experiments reported in Fig. 3 hold.

Finally, in Fig. 6(b) we investigate the trend of the computation rate versus the number of INs, with rounds of \( 10^5 \) items, and for several \( RTT_{IN-CN} \) values: it can be observed that even with a large number of INs performance remains practically unaffected. Therefore, the GCR scheme scales well with the number of players participating to the computation system.

We can conclude that the computation rate achieved by GCR in the online phase is mostly conditioned by the communication between the INs and the CN, and can be optimized by opportunistically choosing the round size, by controlling the \( RTT_{IN-CN} \) and by allocating sufficient bandwidth resources on each IN-to-CN path.
7.3. SEPIA-GCR versus SEPIA-SSS

In this Section we contrast the performance achieved by SEPIA-GCR and SEPIA-SSS when executed in the same network conditions. We recall that for SSS security of the protocol is guaranteed by the fact that PPs are operated by distinct administrative domains. Thus, we have been considering different \( \text{RTT}_{\text{PP}} \) values in order to investigate the impact of geographically distributed PPs on the SSS performance.

In Fig. 7(a) we report the computation rate of GCR and SSS versus the round size for different values of the \( \text{RTT}_{\text{PP}} \). The results reported refer to 5 players emulated on a single machine with unlimited bandwidth, \( \text{RTT}_{\text{IN-C}} = \text{RTT}_{\text{PP}} = 100 \text{ ms} \), collusion threshold \( \ell = 4 \) for GCR, and 5 PPs for SSS. Notice that also for SSS the same qualitative dependency on the \( n_r \) and on the \( \text{RTT}_{\text{PP}} \) holds as for the GCR online phase, and an expression similar to Eq. (2) can be derived by changing the computation times on the INs with those on the PPs. This explains the trend of the SSS computation rate depicted in Fig. 7(a), both as a function of \( n_r \) and \( \text{RTT}_{\text{PP}} \), that was already observed in Fig. 3(a). Furthermore, Fig. 7(a) shows that GCR consistently outperform SSS for whatever round size and RTT value. These results can be easily explained by looking at Fig. 8, which reports the round time break-down for the two schemes. In SSS the communication time between the PPs – needed at the end of each round for reconstructing the output – is the responsible for part of the performance degradation, especially for longer \( \text{RTT}_{\text{PP}} \). However, Fig. 7(a) shows that even when \( \text{RTT}_{\text{PP}} \) is negligible, the performance of SSS is lower because of the higher computation time on the PPs required for the Lagrange interpolation.

In Fig. 7(b) we have investigated the performance as a function of the number of players in the computation, for both GCR and SSS with \( \ell = 4 \) and 5 PPs, respectively. Even though Fig. 7(a) suggests to set the \( n_r = 10^5 \), we had to set \( n_r = 10^4 \) because of the memory limitations of our testbed for the scenario with 90 players. This setting reduces the performance gain of GCR over SSS. However, Fig. 7(b) shows that even in this case GCR is at least three times
faster than SSS also when $RTT_{PP\rightarrow PP} = 5\text{ ms}$. Furthermore, the performance of SSS decreases with the number of players in the computation, whereas the GCR one is practically not affected by the number of players.

Finally, Fig. 9 shows the computation rate achieved by the two secure multiparty schemes as a function of the collusion threshold. For SSS the collusion threshold is $M - 1$ for $M$ PPs, and is varied by increasing the number of PPs (and the degree of the random polynomial). In GCR the collusion threshold $l$ is equal to the number of elements in each RS minus one. In this emulation scenario we set the number of input peers to 30, $n_r = 10^2$, and we considered that the CN and the PPs are experiencing the same network conditions as the other players, i.e., $RTT_{IN\rightarrow IN} = RTT_{IN\rightarrow CN} = RTT_{PP\rightarrow PP} = 100\text{ ms}$, while $RTT_{PP\rightarrow PP}$ has been varied within the range $[0, 400]$ ms. For GCR increasing the collusion threshold leads to longer RS generation times (as already shown in Fig. 4(b)), but has no influence on the online computation speed. On the contrary, with SSS it leads to an exponential decrease of the performance, especially when the contribution of the communication between the PPs is not negligible. By offloading to the offline phase the overhead due to the mechanism introduced to protect the data privacy (i.e., the computation of the random elements) the GCR attains the maximum possible computation speed, i.e., that one of a clear-text addition in a distributed system.

8. Resilience to faults

So far we have been assuming a “cooperative leaving” behavior: players release their unused random elements to the system before leaving. However, if a player shuts down without releasing its random elements – e.g., due to failure, power off or disconnection – all accumulated RSs in the system are invalidated and become useless. In large scale systems such events might not be infrequent and it is important to assess their impact on the overall GCR performance. In the following analysis we assume that each player can fail during a computation round with probability $p_f$. In practice, the value of $p_f$ can be controlled by proper redundancy techniques. The failure of the central collector is neglected.

Consider $N$ players accumulating data to be elaborated by the secure computation system in rounds of size $n_r$. As soon as $n_r$ data have been collected, a round can be launched only if a sufficient number of RSs is available (i.e., $A \geq n_r$ in Fig. 10), otherwise the computation round is put on hold for $t_{wu}$ until the remaining RSs are generated.

Let $t_d$ be the average time for generating a single RS, and $t_c$ be the average time for performing a single addition.
Given $N_f$ players at the first round, the number of active players at the $j$th round is $N_j = N_f - N_{fauts}^{(j-1)} - N_{fleaving}^{(j-1)} + N_{growing}^{(j-1)}$, where $N_{fauts}^{(j-1)}$, $N_{fleaving}^{(j-1)}$, and $N_{growing}^{(j-1)}$ are the total number of players who left, joined, and failed, respectively, from the first to the $j$th round. For simplicity of the analysis, we assume a stable system (i.e., the expected number of active players at the generic round $j$ is $N$), and that fault events across rounds are independent. In other words, the overall balance between players joining, leaving, and failing is such that (on average) the number of active players is $N$. Hence, we model $N_j$ as a random variable with distribution $P(N_j)$ in the interval $[N - \alpha, N + \beta]$. Thus Eq. (4) can be rewritten as:

$$E(T_r) = \sum_{N=0}^{N_f - \beta} \sum_{h=0}^{N_f - \alpha} T_r(h) \cdot \prod_{w=0}^{h-1} P_f(N_j - w).$$

Finally, we define the average GCR computation rate as $RGCR = n_t/E(T_r)$.

Fig. 11 shows the trend of $RGCR$ as function of the fault probability $p_f$ over a round. The number of active players at the beginning of a round is modelled as a random variable uniformly distributed in the interval $[N - \alpha, N + \beta]$ with $\alpha = 0.05N/2$. The other parameters, listed in Table 2, are derived from the simulation results reported in Sections 7.1 and 7.2.

Fig. 11 shows that the more the players participating to the system, the smaller should be the players’ fault probability so as to guarantee nominal performance. In particular, it is evident that $RGCR$ degrades very quickly for $p_f > 10^{-3}$. However, a fault probability of $10^{-3}$ is quite unrealistic. In fact, for a round lasting about 2 s (like in example of Fig. 11), it corresponds to a player failing (on average) once every 30 min. In a system with 90 players, it corresponds to an extremely short inter-failure time of about 25 s. In other words, for realistic fault probabilities, i.e., smaller than $10^{-4}$, Fig. 11 shows that GCR still guarantees average performance close to the nominal one.

Also in this case Eq. (5) can be further simplified. In fact, when $p_f \approx 10^{-4}$, it is $P_f(N_j) \approx N_f p_f$. Hence, the last factor in
Eq. (5) can be rewritten as

\[
E(T_r) \approx t_r + t_w + (2t_r + t_w + t_b + T_g) \cdot p_f \cdot N_{N_f} \cdot N_f
\]

which allows deriving a linear approximation of \(R_{GCR}\) for fault probabilities in \([10^{-4}, 10^{-3}]\).

9. Related work

SMC is a cryptographic framework introduced by Yao [13] and later generalized by Goldreich et al. [14]. SMC techniques have been widely used in the data mining community. For a comprehensive survey, please refer to [15]. Roughan and Zhang [3] first proposed the use of SMC techniques for a number of applications relating to traffic measurements, including the estimation of global traffic volume and performance measurements [16]. In addition, the authors identified that SMC techniques can be combined with commonly-used traffic analysis methods and tools, such as time-series algorithms [17] and sketch data structures.

However, for many years, SMC-based solutions have mainly been of theoretical interest due to impractical resource requirements. Only recently, generic SMC frameworks optimized for efficient processing of voluminous input data have been developed [4,18]. Today, it is possible to process hundreds of thousands of elements distributed across dozens of networks within few minutes, for instance to generate distributed top-k reports [19]. While these results are compelling, they are based on the completely secret evaluation scheme. Our work aims at boosting scalability even further by relaxing the secrecy constraint for intermediate results. As such, our approach can be applied only in cases where the disclosure of intermediate results is not regarded as critical – a quite frequent case in practical applications. Moreover, we aim at optimizing the sharing scheme for fast computation in the online phase.

When it comes to analyzing traffic data across multiple networks, various anonymization techniques have been proposed for obscuring sensitive local information (e.g., [20]). However, these methods are generally not lossless and introduce a delicate privacy-utility trade-off [21]. Moreover, the capability of anonymization to protect privacy has recently been called in question, both from a technical [22] and a legal perspective [23].

10. Conclusions

The use of SMC techniques has recently been proposed to overcome the inhibiting privacy concerns associated with inter-domain sharing of network traffic data. Although design and implementation of basic SMC primitives have recently been optimized (e.g., by the SEPIA protocol suite), processing time as well as communication overhead is still significant.

In the context of collaborative inter-ISP network monitoring there are several practical use cases for which perfect secrecy of intermediate results is not required, or that can be anyway mapped to simple computations. In such cases we advocate the use of “elementary” (as opposed to “complete”) secure multiparty computation (E-SMC) procedures. Indeed, E-SMC supports only simple computations with private input and public output, i.e., they cannot handle secret input nor secret (intermediate) output. The proposed GCR scheme is based on additive secret sharing and, besides the simplification of an E-SMC scheme, enables to divide the computation process into an offline and an online phase. Random secret shares can be generated during the offline phase, with constant storage overhead, whereas the actual queries are run in the online phase with no additional overhead compared to the equivalent plain-text operation.

In this paper we have addressed several system-design aspects relevant for the adoption of GCR in large-scale scenarios. In particular, we have addressed the problem of the natural churn in the number of participants (i.e., joining and leaving), by providing a simple mechanism that allows to save most of the speed-up deriving from the offline random set computation. We have also provided a theoretical analysis of GCR resilience to input nodes faults, as well as a quantitative assessment by using numerical results from emulations.
The GCR prototype has been implemented as extension of SEPIA, a multiparty computation protocol suite that already implements natively the Shamir’s secret sharing scheme. This allows to leverage the optimized implementation of SEPIA’s communication protocols, and at the same time enables the unbiased comparison of the performances achievable by the two SMC schemes.

For assessing GCR performance we have emulated a number of realistic network setups in a distributed testbed. Results show that additions via GCR are always faster than via SEPIA-SSS and scale better both in data volume and number of participants. Therefore, we conclude that GCR is amenable for massive-scale adoption in the context of collaborative network monitoring, whenever operations can be mapped to chains of not sensitive additions.

Still, we recognize that not all the network monitoring applications can be mapped to simple additions. In practical applications one could combine GCR and SSS into a hybrid approach, switching to one of the other scheme depending on the particular use-case, with the option of trading scalability versus functional completeness. The implementation of both GCR and SSS within the SEPIA package is a key enabler for further experimental work along this direction.

The source code of the GCR implementation is available at https://portal.ftw.at/public/GCR-source-code.

Acknowledgements

This work was supported by the DEMONS project funded by the EU 7th Framework Programme [G.A. No. 257315] (http://fp7-demons.eu). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DEMONS project or the European Commission. The authors thank Pasquale Lorusso for the prototype implementation of GCR within the SEPIA framework.

References


Alfonso Iacovazzi received his MSc Degree in Telecommunication Engineering from Sapienza University of Rome, Italy, in 2008, and his PhD degree in Information and Communications Engineering from the same University, in 2013. Since March 2013 he is a Postdoctoral Research Fellow at DIET Dept, Rome, Italy. He is part of the Networking Group. His main research interests are about communications security and privacy, traffic analysis and monitoring, traffic anonymization, cryptography (mathematical aspects and applications).

Alessandro D’Alconzo received the MSc Diploma in Electrical Engineering and the PhD degree from the Polytechnic of Bari, Italy, in 2003 and 2007, respectively. Since 2007 he is Senior Researcher at the Telecommunications Research Center Vienna (FTW), Austria. Since 2008 he is Management Committee representative of Austria for the COST Action IC0703 “Traffic Monitoring and Analysis”. Since September 2010 he is FTW’s scientific coordinator for the EU IP project DEMONS. His current research interests embrace the Network Measurements and Traffic Monitoring area, Quality of Experience evaluation, and application of secure multiparty computation techniques to inter-domain network monitoring.
Fabio Ricciato received the PhD in Information and Communications Engineering in 2003 from University La Sapienza, Italy. In 2004 he joined the Telecommunications Research Center Vienna (FTW) where he later acquired the leadership of the Networking Area. Since 2007 he is Assistant Professor (Ricercatore) in the Telecommunications Group at the University of Salento, where he teaches the course of “Telecommunication Systems”. His research interests cover various topics in the field of Telecommunication Networks, including Traffic Monitoring and Analysis, Network Measurements, security and privacy, routing and optimization, software-defined radio networks.

Martin Burkhart received an MSc and PhD degree from ETH Zurich, Switzerland, in 2003 and 2011, respectively. From 2003–2007 he worked as a software engineer for the banking and logistics industry. His research interests include Internet measurement, network anomaly detection, collaborative network security and applied cryptography. He developed the SEPIA library for secure multiparty computation and is currently working as a security consultant. He has served as a technical reviewer for several international journals and conferences and has an issued patent in network anomaly detection.