Practical Privacy-Preserving Benchmarking

Florian Kerschbaum

Abstract Benchmarking is an important process for companies to stay competitive in today's markets. The basis for benchmarking are statistics of performance measures of a group of companies. The companies need to collaborate in order to compute these statistics.

Protocols for privately computing statistics have been proposed in the literature. This paper designs, implements and evaluates a privacy-preserving benchmarking platform which is a central entity that offers a database of benchmark statistics to its customers. This is the first attempt at building a practical privacy-preserving benchmarking system and the first attempt at addressing all necessary trade-offs.

The paper starts by designing a protocol that efficiently computes the statistics with constant cost per participant. The protocol uses central communication where customers only communicate with the central platform which facilitates a simple practical orchestration of the protocol. The protocols scale to realistic problem sizes due to the constant communication (and computation) cost per participant of the protocol.

1 Introduction

Benchmarking is the comparison of one company's key performance indicators (KPI) to the statistics of the same KPIs of its peer group. A key performance indicator (KPI) is a statistical quantity measuring the performance of a business process. Examples from different company operations are make cycle time (manufacturing), cash flow (financial) and employee fluctuation rate (human resources). A peer group is a group of (usually competing) companies that are interested in comparing their KPIs based on some similarity of the companies. Examples formed along different

Florian Kerschbaum
SAP Research, Karlsruhe, Germany, e-mail: florian.kerschbaum@sap.com

characteristics include car manufacturers (industry sector), Fortune 500 companies in the United States (revenue and location), or airline vs. railway vs. haulage (sales market).

Privacy is of the utmost importance in benchmarking. Companies are reluctant to share their business performance data due to the risk of losing a competitive advantage or being embarrassed. Several privacy-preserving protocols that can be used for benchmarking that keep the KPIs confidential within one company have been proposed in the literature [2, 5, 9, 10, 13, 20]. None of those matches the requirements of a large service provider offering a benchmarking service entirely. Instead this paper proposes a new practical, constant cost, centralized privacy-preserving benchmarking protocol and evaluates it.

A benchmarking platform is a central service provider that offers a database of statistics of peer groups and KPIs to its customers. Customers, i.e. companies, would first subscribe with the service provider and then would be able to retrieve relevant statistics. On the service provider's request the subscribed companies would engage in a protocol to recompute the statistics.

The benchmarking platform is not supposed to acquire the plain text KPIs from the companies acting as a trusted third party, but rather the KPIs are to be kept entirely private to the companies. In the privacy-preserving protocol the benchmarking platform is a regular participant without any input. While the privacy protects the confidentiality of the KPIs for the companies, it alleviates the benchmarking platform from the burden of storing and handling them and protects it from the potential embarrassment due to accidental revealation.

Another important aspect of the service provider model is that the subscribed companies only communicate with the service provider, but never amongst each other. Anonymity among the subscribed companies is a required feature and can only be achieved, if they do not need to address messages to others. The precise requirement for anonymity is that subscribers do not know or refer to any static identifier of other customers (e.g. IP addresses, public keys, etc.). Any static identifier will reveal changes in the composition of the peer group to the subscribers in subsequent executions of the protocol which is undesirable and may break the privacy of the entire system. In many cases, the service provider wants to know the identity of the subscribers for billing purposes, which simplifies communication.

In order to keep the proposed protocols practical, they need to be optimized in computation and communication cost. One measure is the number of rounds that are needed to complete the protocol. A round in the service provider model is a step in the protocol that all subscribers need to complete before any subscriber can initiate the next step. The proposed protocols have a constant number of rounds. Another measure is the communication complexity of the protocol. Our protocol has a constant (i.e. linear in the length of the security parameter) communication complexity for each subscriber independent of the number of subscribed companies.

This paper presents from our view the first practical implementation of privacy-preserving benchmarking. It addresses a number of trade-offs in its distributed systems (single central platform) and security (key distribution and security assumptions) architecture that are tuned for practical performance and economic benefit.

E.g. one central platform is crucial for economic acceptance. Alternatives using multiple mutually distrustful server are economically inacceptable due to the necessarily different business model. Nevertheless for the central communication model no linear cost protocols are known, so we had to sacrifice some security in the key distribution model. Furthermore, not only linear performance is required, but also low constant factors. We know from [20] that a two-party variation of a sub-protocol has superb practical performance. Therefore we are willing to accept the multiplicative hiding assumption (Section 5.1.2) resulting in a significantly improved performance. Our performance evaluation results show that such a solution is at the borderline of what is currently practically feasible.

In summary this paper contributes and evaluates the first privacy-preserving benchmarking platform that combines the following three features:

- *practical*: It addresses all trade-offs from a distributed systems and security point of view for practical performance and economic benefit.
- *centralized*: Each participant communicates only with a central server (or set of servers).
- *constant-cost*: Cost per participant is constant:

  - constant number of rounds
  - constant communication cost

The remainder of the paper is structured as follows: After a short description of the economic motivation for using privacy in Section 2, we introduce some building blocks used and notation in Section 3. Then we describe the registration of companies with the service provider in Section 4. The protocols to recompute the statistics are described and analyzed in Section 5. The practical evaluation using a prototype is presented in Section 6. Related work is discussed in Section 7 before the conclusions are presented in Section 8.

2 Economic Motivation

The privacy requirement of the benchmarking platform is designed for the economic advantage of the service provider. Two advantages can be separated: customer acceptance and competitive advantage.

Privacy is anticipated to increase customer acceptance. The intuition is that customers are reluctant to share business critical data and private benchmarking can alleviate the risk. This in turn leads to more potential customers, a larger market size and in last consequence to larger revenue.

Privacy can also provide a competitive advantage. The risk and cost of sharing KPIs to engage in benchmarking can be lowered by privacy. Thereby offering a higher benefit to customers, justifying a higher price or increasing the market share.

Also given the realistic possibility of privacy-preserving benchmarking with similar results to and the same price as non-privacy-preserving benchmarking, there is

no reason to engage in non-privacy-preserving benchmarking. As mentioned above, the privacy feature also alleviates the service provider from handling and storing the individual KPIs and the embarrassment in case of an accidental disclosure, which reduces the operating costs of the service provider.

3 Preliminaries

3.1 Homomorphic Encryption

Our protocols are built using homomorphic encryption. In homomorphic encryption one operation on the cipher texts produces an encryption of the result of a homomorphic operation on the plain texts. In particular, we require the homomorphic operation to be addition (modulo a constant). Several such encryption systems exist [3, 8, 24, 27, 28]. We suggest Paillier's encryption system [28] which has been generalized in [8]. Let $E_X(x)$ denote the encryption of x with public key $\overline{K}_X$, then Paillier's encryption system has the following property:

$$E_X(x) \cdot E_X(y) = E_X(x+y)$$

From which the next property can be easily derived:

$$E_X(x)^y = E_X(x \cdot y)$$

3.2 Oblivious Transfer

Oblivious Transfer (OT) was introduced in [30] and generalized to 1-out-of-2 OT in [12]. In 1-out-of-2 OT the sender has two secrets x_0 and x_1 and the receiver has one bit b. After the execution of the OT protocol, the receiver has obtained x_b, but has learnt nothing about $x_{\neg b}$. The fastest known implementation of OT is described in [25]. It was proven secure under the (computational and decisional) Diffie-Hellman assumptions in the random oracle model. An OT protocol between a sender S and a receiver R (where the parameters are clear from the context) is denoted by

$$S \xrightarrow{\text{OT}} R$$

3.3 Message Authentication Codes

A message authentication code (MAC) is a function parameterized by a secret key k that is easy to compute and compresses an input x of finite arbitrary length to

an output $MAC(x,k)$ of fixed finite length [29]. More importantly, MACs provide computation-resistance, i.e. given any number of authenticated texts $\langle x_i, MAC(x_i,k)\rangle$ it is computationally infeasible to compute another authenticated text $\langle x, MAC(x,k)\rangle$ $(x \neq x_i)$ without knowing the key k. A successful attempt of producing an authenticated text is called MAC forgery.

3.4 Secret Sharing

A secret sharing scheme divides a secret s into n shares $S = \{s_1, \ldots, s_n\}$ such that any subset of S of size t can be used to recover s. Modular addition can be used for secret sharing where $t = n$, if $s = \sum_i^n s_i \mod m$ [19]. It can be replaced by exclusive-OR $s = s_1 \oplus \ldots \oplus s_n$ (where $\oplus$ denotes exclusive-OR). Both secret sharing schemes are perfect, i.e. less than t shares yield absolutely no information, in the information-theoretic sense, about s.

4 Registration

When companies subscribe to the benchmarking platform, a trusted third party is involved. This trusted third party only needs to be involved once during registration, but we assume that it does not maliciously collaborate with the service provider. A practical candidate for such a third party would be PKI certification authority. The issued certificates will be used to establish secure and authenticated channels between the subscribers and the service provider.

Furthermore, we extend the third party's functionality to a dealer. The trusted third party will distribute secrets to the subscribers. In particular, the trusted third party distributes to all subscribers:

- K_{common}: A private key in the homomorphic encryption system of Section 3.1. This private key is shared among the subscribers, but unknown to the service provider. The corresponding public key $\overline{K}_{common}$ is known to the service provider.
- s_{common}: A key for the MAC algorithm, also shared among the subscribers and unknown to the service provider.

5 Protocols

This section will present protocols for computing the following statistics:

- mean
- variance
- maximum

These protocols will be executed for each peer group and each KPI. Let x_i denote the input (KPI) of the i-th subscriber X_i. For reason explained later the subscribers know the size n of the peer group and therefore computation of the mean ($\mu = \frac{1}{n}\sum_i^n x_i$) is equivalent to summation. Furthermore, for practical reasons, we compute mean and variance in different rounds, such that the mean has been revealed before the variance is being computed. In this case computation of the variance is also equivalent to summation (of $(x_i - \mu)^2$). Note that, the summation of the variance should be done using a different shared key pair $\langle K_{common'}, \overline{K}_{common'} \rangle$, if the cost of distributing it to all subscribers is affordable.

Summation using homomorphic encryption is quite natural: Each subscriber submits $E_{common}(x_i)$ to the service provider which, after receiving all inputs, computes the encrypted sum as $\prod_i^n E_{common}(x_i) = E_{common}(\sum_i^n x_i)$.

Maximum computation for two parties was first presented and evaluated in [20] and the following protocol builds upon the successful experiences. The subscriber sends $E_{common}(x_i)$ to the service provider who maintains an encrypted (intermediate) maximum $E_{common}(max)$. The service provider chooses two large random numbers r and r' (e.g. size $O(m^2)$ where m is the maximum KPI possible), such that $r' < r$. He sends to the subscriber $E_{common}(c) = E_{common}(r \cdot (x_i - max) + r')$. It holds that $c < 0 \Leftrightarrow x_i < max$ where "< 0" is interpreted in a mapping of integers $[-d, d-1]$ to $[0, 2d-1]$ according to congruence in modular arithmetic.

Furthermore, the service provider flips a coin $r'' \in \{0, 1\}$ and if r'' is 1, then he negates c resulting in c'. The subscriber and the service provider share c as $c = c' \oplus r''$. They then engage in an OT where the service provider sends one of $\langle E_{common}(x_i + r'''), E_{common}(max + r''') \rangle$ where r''' is a random number chosen by the service provider to hide the encrypted value. The service provider switches the values, if he negated c. Then the subscriber chooses according to c', rerandomizes the value (by homomorphically adding $E_{common}(0)$) and returns it to the service provider which can then obtain a new $E_{common}(max)$ by subtracting r'''.

After executing the protocols the service provider has the encrypted values $E_{common}(sum)$ for the mean and variance and $E_{common}(max)$ for the maximum.

The intention of the protocol is that the service provider obtains the result to store it in the database for all subscribers and future subscribers until the statistics are recomputed. For software engineering reason the subscribers are not supposed to keep a record of the protocols, but rather obtain the results via a database query. The service provider submits therefore the encrypted results $E_{common}(result)$ to all subscribers and they respond with the decrypted $result$. We chose to submit to all subscribers, such that no single subscriber may obtain the correct result, but return an incorrect result to the service provider (and the other subscribers). The service provider can compare all $result$ values and detect the presence of malicious subscribers if there is at least one honest subscriber. He can identify the malicious subscribers, if there is a majority of honest subscribers.

Round 1:

$$X_i \longrightarrow SP \ E_{common}(x_i)$$
$$SP \longrightarrow X_i \ E_{common}(c) = E_{common}(-1^{r_3} \cdot (r_1 \cdot (x_i - max) + r_2))$$
$$SP \xrightarrow{\text{OT}} X_i \ E^c = \begin{cases} E_{common}(x_i + r_4) & \text{if } c \geq 0 \oplus (r_3 = 0) \\ E_{common}(max + r_4) & \text{if } c < 0 \oplus (r_3 = 0) \end{cases}$$
$$X_i \longrightarrow SP \ E_{common}(max') = E^c \cdot E_{common}(0)$$
$$SP \qquad E_{common}(max) = E_{common}(max' - r_4)$$

Round 2:

$$SP \longrightarrow X_i \ E_{common}(sum) = E_{common}(\textstyle\sum_{i=1}^n x_i)$$
$$E_{common}(max)$$
$$X_i \longrightarrow SP \ sum$$
$$MAC(sum|i, s_{common})$$
$$max$$
$$MAC(max|i, s_{common})$$
$$E_{common'}((x_i - \tfrac{sum}{n})^2)$$

Round 3:

$$SP \longrightarrow X_i \ E_{common'}(sum') = E_{common'}(\textstyle\sum_{i=1}^n (x_i - \tfrac{sum}{n})^2)$$
$$H(MAC(sum|1, s_{common}), \dots, MAC(sum|n, s_{common}))$$
$$H(MAC(max|1, s_{common}), \dots, MAC(max|n, s_{common}))$$
$$X_i \longrightarrow SP \ sum'$$
$$MAC(sum'|i, s_{common})$$

Round 4:

$$SP \longrightarrow X_i \ H(MAC(sum'|1, s_{common}), \dots, MAC(sum'|n, s_{common}))$$

Fig. 1 Benchmarking Protocol

5.1 Security Assumptions

5.1.1 Semantic Security

Semantic security means it must be infeasible for a probabilistic polynomial-time adversary to derive information about a plaintext when given its cipher text and the public key. A more formal definition can be found in [15]. The property of semantic security has been proven equivalent to cipher text indistinguishability. If an encryption scheme possesses the property of indistinguishability, then an adversary will be unable to distinguish pairs of ciphertexts based on the message they encrypt. Paillier's encryption system has been proven semantically secure against chosen plain-text attacks under the Decisional Composite Residuosity Assumption [28].

5.1.2 Multiplicative Hiding

We assume that a number x is effectively hidden by multiplying it with a random number r in a large domain (e.g. $O(x^2)$) and adding another random number $r' < r$. Let $\mathbb{D}$ denote the domain of numbers hidden in this form. Then we assume that $d \in \mathbb{D}$ reveals no relevant information about the hidden x.

5.2 Security in the Semi-Honest Model

Following Goldreich's definitions [14] we define the view $VIEW_i$ of the i-th party as

Definition 1 *The view of the i-th party during an execution of our protocols on* $(x_1, \ldots, x_m)$, *denoted* $VIEW_i(x_i, \ldots, x_m)$ *is* $\{x_i, r_i, m_1, \ldots, m_\phi\}$, *where* r_i *represents the outcome of the i-th party's internal coin tosses, and* m_i *represents the i-th message it has received.*

The result is implicit in the view. Further following Goldreich's definitions of semi-honest security (for deterministic functions), we define security against a semi-honest attacker:

Definition 2 *Let* $f : (\{0,1\}^*)^m \mapsto (\{0,1\}^*)^m$ *be an m-ary functionality, where* $f_i(x_1, \ldots, x_m)$ *denotes the i-th element of* $f(x_1, \ldots, x_m)$. *For* $I = \{i_1, \ldots, i_t\} \in \{1, \ldots, m\}$, *we let* $f_I(x_1, \ldots, x_m)$ *denote the (sub-)sequence* $f_{i_1}(x_1, \ldots, x_m), \ldots, f_{i_t}(x_1, \ldots, x_m)$ *and let* $VIEW_I(x_1, \ldots, x_m) = (I, VIEW_{i_1}(x), \ldots, VIEW_{i_t}(x))$. *We say that our protocols privately compute* f *if there exists a polynomial-time simulator, denoted S, such that for every I of size t, such that* $S(I, (x_{i_1}, \ldots, x_{i_t}), f_I(x_1, \ldots, x_m))$ *is computationally indistinguishable from* $VIEW_I(x_1, \ldots, x_m)$.

We also use the composition theorem from [14]. It states that if a function g is privately reducible to f and there exists a protocol to privately compute f, then there exist a protocol for privately computing g. A protocol privately reduces g to f, if it privately computes g when f is replaced by an oracle (according to the ideal model). We use this to replace the use of OT in our protocols.

We show security against a semi-honest subscriber by giving a simulator of his view. In the first round, he receives a random value $d \in \mathbb{D}$ and an encrypted random share $E_{common}(r)$. In the second round he receives encryptions of $E_{common}(sum)$ and $E_{common}(max)$. The third round repeats $E_{common}(sum')$ for the variance.

We show security against a semi-honest service provider by giving a simulator of his view. In the first round he receives $E_{common}(x_i), E_{common}(max)_i$ from each participant. Due to the semantic security these encryptions appear as random numbers. In the second round he receives the results sum (mean) and max, as well as another encryption $E_{common}((x_i - \mu)^2)$ and in the third round he receives sum' (variance) (again from each participant).

5.3 Security in the Constrained Malicious Model

Security against a malicious attacker provides security against any deviations from the protocol, such that secrecy of the computation can be reduced to the semi-honest security. Security against a malicious attacker provides no security against protocol abortion (from the platform provider) or providing false inputs. In particular, a malicious subscriber can submit the maximum possible KPI value and thereby falsify the result of the maximum computation. Differently from an auction, where the maximum value or at least its submitter (Vickrey auctions) are revealed, this is not case for benchmarking. We therefore abandon security against a malicious attacker and its cost in favor of a lesser security definition.

We are particularly concerned with secrecy of the KPIs. We therefore assume a constrained malicious attacker that can deviate from the protocol in order to obtain additional information (except what can be inferred by the result and the local input). The constraint is that the attacker is to deliver the correct result to the other parties. Such behavior can be enforced for a service provider by contract obligations. It is also economically motivated, since we can assume that all subscribers and the service provider have a vested interest in obtaining the correct result.

Consider the following attacker impersonating a service provider: When obtaining the result, he simply resends an encrypted, originally submitted, KPI $E_{common}(x_i)$. Then he can compute the mean, variance and maximum locally and store the correct results in the database where the subscribers will retrieve them. He deviated from the protocols, obtained all individual KPIs and still delivered the correct result to all subscribers.

The following protocol should prevent any constrained malicious attacker. When obtaining the result the subscribers not only return the decrypted result *result*, but also send a MAC for the value received $MAC(result|i, s_{common})$ (where | denotes concatenation). In a further round the service provider proves to all subscribers that he submitted the same value for decryption to all of them by sending them $\theta = H(MAC(result.1, s_{common}) \mid \ldots \mid MAC(result.n, s_{common}))$ (where $H()$ denotes a cryptographic hash function).

Formally, we define an attacker in the constrained malicious model:

Definition 3 *Let $f : (\{0,1\}^*)^m \mapsto (\{0,1\}^*)^m$ be an m-ary functionality, $I = \{i_1, \ldots, i_t\} \subset \{1, \ldots, m\}$, $\neg I = \{1, \ldots, m\} \setminus I$ and $(x_1, \ldots, x_m)_I = (x_{i_1}, \ldots, x_{i_t})$. A pair (I, C), where C is a polynomial-size circuit family, represents an adversary A in the real model. The joint execution of our protocols under C and $\neg C$ where $\neg C$ coincides with the strategies defined by our protocols, denoted as REAL(x), is defined as the output resulting from the interaction between $C(x_I)$ and $\neg C(x_I)$. An adversary A is admissible (for the constrained malicious model) if $REAL_{\neg C(x)} = f_{\neg C(x)}$.*

Then we extend the output of subscriber X_i in the ideal model with a verification bit b_i. In the ideal model b_i is always true, in the real protocol it is computed by verifying $\theta \stackrel{?}{=} H(MAC(result.1, s_{common})| \ldots |MAC(result.n, s_{common}))$. Recall, that due to software engineering reasons of decoupling retrieval from computation the sub-

scribers are not to keep a record of the recomputation, but rather obtain the results via database query.

Finally we state the following theorem:

Theorem 1 *Our benchmarking protocols privately compute average, variance and maximum in the presence of a constrained malicious service provider.*

Proof Sketch: The view of the service provider offers no information (i.e. all received messages appear as random numbers) until he receives the first decrypted result. This implies that all deviations from the protocol until the first decrypted result reveal no information to him. After the first decrypted result he will obtain further decrypted result which provide information. All decryptions solely depend on the encryption sent by the service provider. Our verification bit protocol ensures that if every subscribers outputs "true" as his verification bit, the service provider has submitted the same encryption to all subscribers and obtains the exact same information from all of them (or he has successfully forged a MAC). By definition of the constrained malicious attacker, this is the correct result, i.e. the service provider only obtains the correct result and nothing else.

The final protocols with security against a constrained malicious service provider are depicted in Figure 1. The protocol uses a constant number of rounds (4) and constant message size in each round (i.e. linear in a fixed security parameter K of the homomorphic encryption scheme). It is independent of the number of participants, the number of KPIs or peer groups and the subscribers never need to exchange message even indirectly and therefore remain entirely anonymous amongst each other.

6 Performance Evaluation

The goal of the benchmarking protocols is to act as the basis of a real-world benchmarking platform. Therefore the protocols need to be evaluated under realistic conditions. We implemented a prototype version of the protocols (with partial anonymity among the subscribers only) based on web services. The web service stack consists of a Tomcat 5.5 [1] web application server using the Axis2 1.1 [2] web service engine. All our implementation was done in Java 1.5. The implementation includes our own implementation of Paillier's [28] encryption system.

The test bed includes a central server with a 3.2GHz 32-bit Intel processor with 2GB of RAM of which 256MB were available to the Tomcat Java Virtual Machine. The clients were emulated using VMware [3] as a virtual machine monitor each having 256MB of RAM of which 64MB were available to each Tomcat Java Virtual Machine. Each emulated client ran five Tomcat web application servers acting as
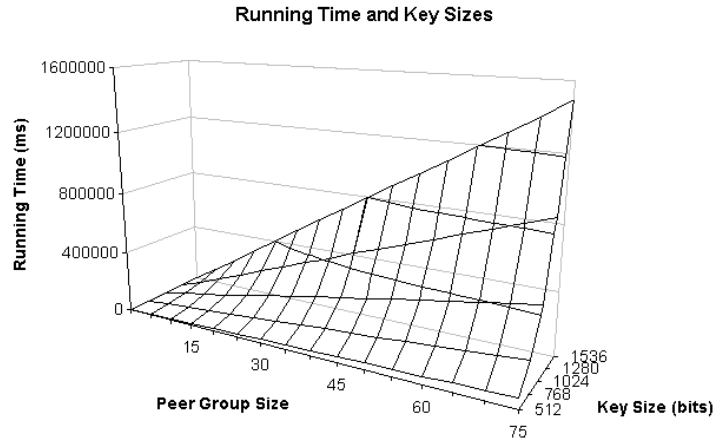
---

[1] http://tomcat.apache.org/

[2] http://ws.apache.org/axis2/

[3] http://www.vmware.com/

five subscribers in the protocol. Up to fifteen such clients were emulated on two servers. We expect realistic peer group sizes to be between 10 and 25 subscribers, such that 75 subscribers as the maximum peer group size should underpin the protocol's scalability.

Running Time and Key Sizes

Fig. 2 Computation performance over key size and peer group size

The first experiment was to measure the computational cost by increasing both the peer group size and the key size of Paillier's encryption system, but independently of each other. We computed one KPI repeatedly for varying peer group sizes and key lengths. The entire system including server and clients was emulated on one machine disabling network cost. In the results depicted in Figure 2 one can see the expected linear increase of running time with peer group size as expected from the constant cost per participant, but running time increases quickly with key size. While the computational cost for one participant is below 1 second for 512 bit key length, it is almost 20 seconds for 1536 key length. Paillier's encryption system uses RSA-type keys [31], such that 768 or 1024 bit key lengths can be expected to be the most common ones. The conclusion from this experiment is to be conservative when choosing the key length for large peer groups, but realistic key sizes, such 768 or 1024 bits are feasible even for large peer groups.

The second experiment was to measure network cost by increasing the delay of the network. We computed one KPI first over the shared local area network (LAN), then with dummynet [32] as a wide-area network (WAN) emulator. We increased the round trip time to 200 milliseconds which corresponds roughly to the round trip time in the Internet from Germany to Japan or Australia. From the results in Figure 3 we can see that the performance on LAN is dominated by the computation cost and that the performance on WAN is dominated by communication cost. As expected leads the constant communication cost to a linear increase in running time. The conclusion
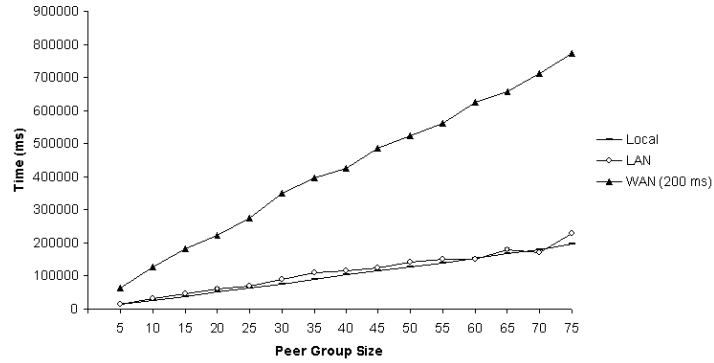
Fig. 3 Performance under different network conditions

	Centralized	No. Central Servers	Cost per Participant	Anonymizable	Implemented
[2]	n	n/a	$O(n^2)$[1]	n	n
[20]	n	n/a	$O(\log^2 n)$[1]	y	n
[9, 10]	y	1	$O(n)$	n	n
[13]	y	2	$O(1)$	y	y
[5]	y	3, 5, 7	$O(1)$	y	y
this paper	y	1	$O(1)$	y	y

Table 1 Overview of Related Work

from this experiment is that for a protocol to be practically realizable over current network conditions the focus should be on communication cost. A peer group of 75 subscribers computes one KPI in approximately 12 minutes. We expect up to 200 KPIs, but computations can be performed concurrently, such that an estimate of the combined running time is difficult.

7 Related Work

The application of private collaborative benchmarking has been first described in [2]. The authors present important applications of a secure division protocol, where one party performs the blinded division after the others have agreed on the blinding. Filtering outlying values before benchmarking has been considered in [20], where also the initial idea for the comparison (maximum) protocol was presented, but all communication was done in a peer-to-peer model. Although both papers are concerned with benchmarking as an application, they consider different computations than our statistics.

Other examples of privacy-preserving statistical computations include the computation of the mean [11, 21]. Here two party each have a mean as a fraction and

want to compute the combined mean. In [1] the median is being computed as a two or peer-to-peer multi-party protocol.

All these and many more protocols are special protocols of general secure (multi-party) computation (SMC) protocols. SMC was introduced in [34] and generalized to multi-party computations in [4, 16] with computational security [16] and information-theoretic security [4]. The draft by Goldreich [14] gives a general construction and very extensive background on the security models. The protocols by Yao were implemented in [23].

The motivation to build special protocols for important problems was realized soon [17]. Besides benchmarking a related application are auctions. For auctions similar models to our service provider model have been introduced. A model applicable to general multi-party computations is introduced in [26] which has been extended in [18, 22]: Two mutually distrusting servers execute a binary circuit with the input of multiple external parties. The separation of input clients and computation servers has been picked-up in [6] and extended in [7]. The protocols in [6] and the recent result by [7] for constant-round SMC for any function requires at least three servers. All implementations of SMC [5, 13] except ours use this model.

In [13] the model has been applied to online surveys which computes the same statistics as our protocols. Their implementation is based on [23] and uses the two server model of [26]. They do not report absolute performance figures, but indicate that the computation is practical. They even present an idea on how to verify the entries which unfortunately does not apply in our case.

In [5] the model is applied to secure auctions which can be extended to benchmarking. Their implementation is based on variations of [6] and they report performance for three, five and seven servers. They do not report figures for the entire application, but again indicate that the computation is practical.

The problem with both implementations and the model in general is that it requires multiple computation servers. These servers need to be mutually distrustful in order for the protocol to be secure which implies separate business entities in a practical realization. This is a major obstacle for a single service provider business model, since one has to organize and finance several collaborating, but mutually distrustful partners. Therefore we argue that different protocols are needed.

The single server model has been used in theory for maximum [9] and mean [10] computations. Both protocols have linear communication cost in the number of participants as opposed to our constant. Furthermore our protocols achieve full anonymity (no static identifier), even for the provider of the maximum (auction winner), which is not the case in [9, 10] where companies refer to each other by public keys. Static identifiers reveal changes in peer group composition to subscribers in subsequent executions of the protocol which is undesirable. We also strengthen the security against the central platform provider to the constrained malicious model, since this is our main economic motivator.

Table 1 provides an overview over the related papers for protocols and implementation. Although none of the previous work considers anonymity as a feature, we anticipate that anonymous versions can be built from the work as indicated in the table. Our protocols are not only the first constant-cost, anonymous, centralized

benchmarking protocols, but also the first implementation not based on the multiple server model.

One can view our problem also as a database privacy problem. We compute a non-sensitive database from sensitive distributed entries. A simple approach for secure query evaluation on a sensitive database, based on homomorphic encryption, is evaluated for performance in [33] and found lacking the necessary performance.

8 Conclusions

In this paper we have presented and evaluated a constant-cost, anonymous, centralized privacy-preserving benchmarking protocol. The secrecy of individual KPIs is maintained against the service provider in any case, if he delivers the correct result. It can be used to realize a (central) privacy-preserving benchmarking platform that computes the statistics of the key performance indicators of the subscribers.

Full anonymity, i.e. everybody except the central service provder remains anoynmous, can be achieved using an anonymous communication channel. Then participation in multiple peer groups is possible.

The practical evaluation shows that the effort for building a constant-cost (independent of the number of participants) protocol is fruitful and yields computation times on the order of minutes even over WAN network conditions. The protocols are among the first practically evaluated secure multi-party computation systems.

Based on our positive economic evaluation of privacy we intend to continue to build practical systems on top of the protocols.

References

1. G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the kth-ranked element. *Proceedings of EUROCRYPT*, 2004.
2. M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. *Proceedings of the ACM workshop on Privacy in the electronic society*, 2004.
3. J. Benaloh. Verifiable Secret-Ballot Elections. *PhD thesis, Yale University*, 1987.
4. M. Ben-Or, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th ACM symposium on theory of computing*, 1988.
5. P. Bogetoft, I. Damgard, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation. *Proceedings of Financial Cryptography*, 2006.
6. I. Damgard, R. Cramer, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. *Proceedings of EUROCRYPT*, 2001.
7. I. Damgard, and Y. Ishai Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. *Proceedings of CRYPTO*, 2005.

----

[1] Different computations than statistics

8. I. Damgard, and M. Jurik. A Generalisation, a Simplification and some Applications of Pailliers Probabilistic Public-Key System. *Proceedings of International Conference on Theory and Practice of Public-Key Cryptography*, 2001.
9. G. Di Crescenzo. Private Selective Payment Protocols. *Proceedings of Financial Cryptography*, 2000.
10. G. Di Crescenzo. Privacy for the Stock Market. *Proceedings of Financial Cryptography*, 2001.
11. W. Du, and M. Atallah. Privacy-preserving Cooperative Statistical Analysis. *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.
12. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM 28(6)*, 1985.
13. J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure Computation of Surveys. *Proceedings of the EU Workshop on Secure Multiparty Protocols*, 2004.
14. O. Goldreich. Secure Multi-party Computation. Available at *www.wisdom.weizmann.ac.il/˜oded/pp.html*, 2002.
15. O. Goldreich. The Foundations of Cryptography Vol. 2. *Cambridge University Press*, 2004.
16. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the 19th ACM conference on theory of computing*, 1987.
17. S. Goldwasser. Multi party computations: past and present. *Proceedings of the 16th ACM symposium on principles of distributed computing*, 1997.
18. A. Juels, and M. Szydlo. A two-server, sealed-bid auction protocol. *Proceedings of the 6th Conference on Financial Cryptography*, 2002.
19. E. Karnin, J. Green and M. Hellman. On Secret Sharing Systems. *IEEE Tranactions on Information Theory 29(1)*, 1983.
20. F. Kerschbaum, and O. Terzidis. Filtering for Private Collaborative Benchmarking. *Proceedings of the International Conference on Emerging Trends in Information and Communication Security*, 2006.
21. E. Kiltz, G. Leander, and J. Malone-Lee. Secure Computation of the Mean and Related Statistics. *Proceedings of Theory of Cryptography Conference*, 2005.
22. H. Lipmaa, N. Asokan, and V. Niemi. Secure Vickrey auctions without threshold trust. *Proceedings of the 6th Conference on Financial Cryptography*, 2002.
23. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-party Computation System. *Proceedings of the USENIX security symposium*, 2004.
24. D. Naccache, and J. Stern. A New Public-Key Cryptosystem Based on Higher Residues. *Proceedings of the ACM Conference on Computer and Communications Security*, 1998.
25. M. Naor, and B. Pinkas. Efficient Oblivious Transfer Protocols. *Proceedings of the symposium on data structures and algorithms*, 2001.
26. M. Naor, B. Pinkas and R. Sumner. Privacy Preserving Auctions and Mechanism Design. *Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999.
27. T. Okamoto, and S. Uchiyama. A new public-key cryptosystem as secure as factoring. *Proceedings of EUROCRYPT*, 1998.
28. P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Proceedings of EUROCRYPT*, 1999.
29. B. Preneel. Cryptographic hash functions. *European Transactions on Telecommunications 5(4)*, 1994.
30. M. Rabin. How to exchange secrets by oblivious transfer. *Technical Memo TR–81, Aiken Computation Laboratory*, 1981.
31. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21(2)*, 1978.
32. L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review 27(1)*, 1997.
33. H. Subramaniam, R. Wright, and Z. Yang. Experimental Analysis of Privacy-Preserving Statistics Computation. *Proceedings of the Workshop on Secure Data Management*, 2004.
34. A. Yao. Protocols for Secure Computations. *Proceedings of the IEEE Symposium on foundations of computer science 23*, 1982.