Secure and Sustainable Benchmarking in Clouds

A Multi-Party Cloud Application with an Untrusted Service Provider

Cloud computing entails a novel security threat: The cloud service provider is entrusted with the data of all its customers. Encryption may provide the solution. Using the example of collaborative benchmarking the authors present and evaluate the exemplary design and implementation that operates on encrypted data.

DOI 10.1007/s12599-011-0153-9

The Author

Dr. Florian Kerschbaum (✉)
SAP AG
SAP Research
Vincenz-Prießnitz-Str. 1
76131 Karlsruhe
Germany
florian.kerschbaum@sap.com

Received: 2010-06-14
Accepted: 2011-02-04
Accepted after three revisions by Prof. Dr. Müller.
Published online: 2011-04-15

This article is also available in German in print and via http://www.wirtschaftsinformatik.de: Kerschbaum F (2011) Sicheres und nachhaltiges Benchmarking in der Cloud. Eine Mehrparteien-Cloud-Anwendung ohne vertrauenswürdigen Dienstanbieter. WIRTSCHAFTSINFORMATIK. doi: 10.1007/s11576-011-0267-1.

1 Introduction

Cloud computing entails a novel security threat: The cloud service provider is entrusted with all of the data of all its customers and may accidentally or maliciously disclose it to third parties. While the service provider may take the necessary precautions in order to protect the confidentiality of the data from outsiders or other customers, the service provider usually inadvertently learns the data and a new trust relationship between customer and service provider is inherent to cloud computing.

For some applications this trust may not be sustainable. Consider, for example, highly confidential data about a company's operation. In order for the customer to engage and sustain cloud computing for applications operating on such data confidentiality even against the service provider is necessary. A long-term and sustainable relationship between cloud service provider and customer should be based on minimal trust assumptions and this includes the trust of the customer in the service provider. Therefore, it would be advantageous for the sustainability of cloud computing, if the service provider could ensure the customer of the confidentiality of his data.

Assurance and preventive security measures are essential for confidentiality. Contracts and fines or other detective measures of protection require the ability to prove a confidentiality breach which can be difficult.

The technical means to provide assurance of data confidentiality without a reference monitor or policy enforcement point is encryption. Standard public-key or symmetric encryption as commonly used to secure data communications is inapplicable to sustainable cloud computing, since it cannot be modified once encrypted. Homomorphic encryption (Damgard and Jurik 2001; Gentry 2009; Paillier 1999) allows such modifications of encrypted data. Nevertheless it is too inefficient for large-scale cloud applications. Secure Multi-Party Computation (Ben-Or et al. 1988; Cramer et al. 2001; Goldreich et al. 1987), an alternative cryptographic technique, is computationally more efficient, but requires significant communication resources.

The research questions addressed in this paper are the design choices for a sustainable cloud information system based on these techniques. The designer has several options in the choice of encryption scheme, key distribution and security model as well as the application's functions and features. He has to balance the conflicting objectives of functionality, security and performance.

We will explore these design options using the case study of a confidentiality-preserving cloud application we have built. We have implemented a collaborative business application for benchmarking.

Benchmarking is the comparison of key performance indicators (KPI) to their statistics within a peer group. Our cloud application computes these statistics without disclosing the KPIs of any individual company.

Benchmarking is an important process for companies to stay competitive in today's markets. It allows them to evaluate their performance against the statistics of their peers and implement targeted improvement measures. Benchmarking services have been proposed and implemented before (Bogetoft and Nielsen 2005; Crotts et al. 2006), but none implements sustainable security against the service provider. The positive impact of confidentiality protection on the willingness of companies to share data has been established in related studies (Eurich et al. 2010).

We have designed, implemented and evaluated a prototype for collaborative benchmarking on encrypted data in the cloud. To the best of our knowledge this is the first cloud application that operates on encrypted data. We will use a combination of homomorphic encryption and secure multi-party computation in order to achieve the necessary performance and functionality. Furthermore we explore several design options in greater detail.

- *Security vs. functionality* (Sects. 3 and 4): First, we show that it is not feasible to securely implement all possible (benchmarking) functions (Sect. 3). While it is always possible to encrypt and compute on the encrypted data, the result of the computation may reveal the protected input. We give the necessary conditions for confidentiality-preserving benchmarking. Second, we show the implications of encrypted cloud computing on a rational player and the necessary selection of statistical functions (Sect. 4). We furthermore show how to efficiently prevent a novel attack on encrypted cloud computing.
- *Security vs. performance* (Sects. 5 and 6): First, we review the design options for key distribution and management and explain our choice (Sect. 5). Second, we present a novel technique for comparison using partially homomorphic encryption (Sect. 6). This novel technique is less secure than comparable cryptographic techniques, but significantly more efficient. It also simplifies the design of the entire application, since then partially homomorphic encryption suffices.
- *Functionality vs. performance* (Sect. 7): We present the software architecture and its component structure, such that confidentiality-preserving benchmarking can be performed despite its performance impact.

The remainder of the paper is structured as follows. In Sect. 2 we describe the problem of confidentiality-preserving cloud computing, our protection goals and compare the available approaches of homomorphic encryption and secure multi-party computation. We also explain collaborative benchmarking in detail. In Sect. 8 we report on the implementation and performance measurements. In Sect. 9 we review related work. Our conclusions are summarized in Sect. 10.

2 Problem and Approach

Confidentiality-preserving cloud computing attempts to reduce the trust assumption in the service provider by encrypting the data by the customer before transmission to the cloud. There are two cryptographic techniques to achieve this: homomorphic encryption and secure multi-party computation.

2.1 Homomorphic Encryption

Homomorphic encryption is a modern encryption technique where one operation on the ciphertexts produces an encryption of the result of a homomorphic operation on the plaintexts. Its application to confidentiality-preserving cloud computing is straight-forward. The customer encrypts the data, retains the key and sends the ciphertext to the service provider. The service provider performs operations on the ciphertexts that map to the homomorphic operations on the plaintext.

Recently a fully homomorphic encryption scheme where the homomorphic operation is "logical not-and" (NAND) has been developed (Gentry 2009). It can be used to implement any function on the plaintext. Nevertheless this scheme is still considered too inefficient to implement secure functions. Instead we use more efficient encryption schemes where the homomorphic operation is limited to addition of integers (modulo a key-dependent constant) where not all operations can be implemented on the plaintext. We will later confirm that performance remains a critical aspect. We used Paillier's encryption scheme (Paillier 1999) and its threshold variant by Damgard and Jurik (2001). Let $E_X(x)$ denote the encryption of x with X's public key and $D_X()$ the corresponding decryption with X's private key, then the following property holds:

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

With simple arithmetic the following property can be derived

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

2.2 Secure Multi-Party Computation

Homomorphic encryption cleanly separates computation from input, but is limited to certain operations for performance reasons. An alternative technique is secure multi-party computation. In secure multi-party computation the function is not computed by only one party on the ciphertexts, but as a joint protocol by all. At the end of the protocol a defined subset of the parties will be able to reconstruct the result, but all parties know how the computation is performed regardless whether they receive a result or not.

Protocols that can implement any function have been proposed based on secret sharing (Ben-Or et al. 1988), oblivious transfer (Goldreich et al. 1987) and homomorphic encryption (Cramer et al. 2001). Generally speaking, one can implement additional functionality to the homomorphic operation by using interactive protocols between the players.

Its application to confidentiality-preserving cloud computing is less obvious. The customers and the cloud service provider engage in a secure computation protocol. The service provider will be treated just as another party. The clients provide input and every party—including the service provider—will learn nothing except what can be inferred from their input and output. Care must be taken in order to balance the computational load in favor of the customers (Kerschbaum 2009).

For secure multi-party computation a formal security model has been defined (Goldreich 2002). Recall that a party may not learn anything that cannot be inferred by its input and output. A proof for security may define a simulator which is given input and output and then show that the statistical difference between the simulator and a real protocol run is a negligible function of a security parameter. The two models defined by Goldreich (2002) are semi-honest and malicious. In the semi-honest model the adversary follows the protocol, but keeps a record of all messages to infer additional information. In the malicious the adversary may behave arbitrarily.

There exists a compiler from protocols secure in the semi-honest model to protocol secure in the malicious model (Goldreich 2002), but the resulting protocols are very inefficient. Furthermore the malicious protocol does not prevent providing false input, such that we have to assume the parties provide the correct input in order to obtain the correct result. We will revisit this problem in Sect. 4.

2.3 Collaborative Benchmarking in the Cloud

Benchmarking helps companies to stay competitive in today's markets. A KPI is a statistical quantity measuring the performance of a business process. Examples from different company functions are make cycle time (manufacturing), cash flow (financial) and employee fluctuation rate (human resources). A peer group is a group of (usually competing)

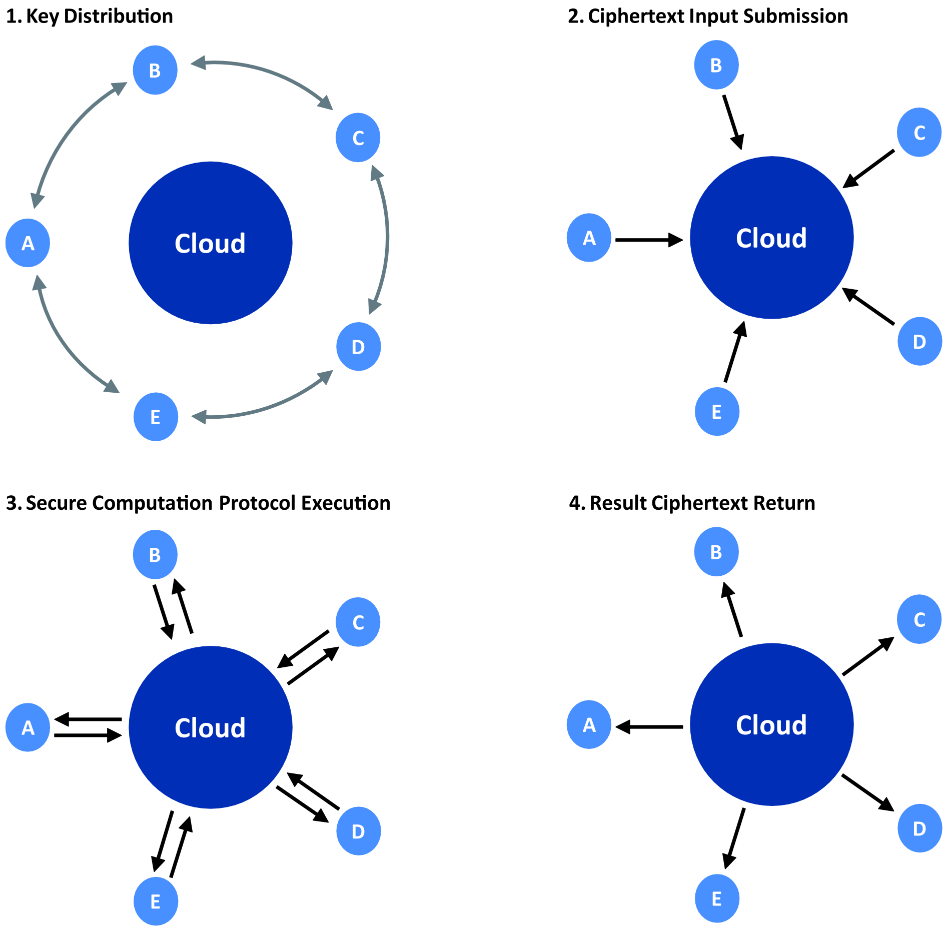
1. Key Distribution
B
C
A
Cloud
D
E
2. Ciphertext Input Submission
B
C
A
Cloud
D
E
3. Secure Computation Protocol Execution
B
C
A
Cloud
D
E
4. Result Ciphertext Return
B
C
A
Cloud
D
E

Fig. 1 Overview of confidentiality-preserving cloud computation

companies that are interested in comparing their KPIs. Examples using different characteristics include car manufacturers (industry sector), Fortune 500 companies in the United States (revenue and location), or airline vs. railway vs. haulage (sales market).

Let there be n customers X_i and one cloud service provider SP. Each customer X_i has one KPI value x_i—we consider each KPI separately. They want to compute several statistics about $x_1, \ldots, x_n$ (we also treat every peer group separately). The following functions are candidates commonly chosen in individual benchmarking: average, variance, maximum, median and top quartile.

The cloud computing paradigm entails that each customer X_i only communicates with the service provider SP, i.e. we prohibit any direct communication between the customers. Assuming a malicious service provider this central communication pattern corresponds to the attacker controlling the network and we inherit the security challenges from this model, such as no key agreement without a trusted third party.

Our protection goal is the confidentiality of the customer X_i's KPI value x_i, i.e. neither any other customer X_j $(j \neq i)$ nor the service provider SP should learn x_i.

2.4 Solution Overview

In the next sections we review several design choices. In order to relate them to the overall design we present **Fig.** 1 of the confidentiality-preserving cloud computation. In this life cycle of the computation we first observe a key distribution (Sect. 5). Then the parties submit their encrypted input (Sect. 6) before the secure computation takes place where we discuss the functions to be computed (Sect. 3). Then the (encrypted) result is returned where special pre-cautions need to be in place (Sect. 4).

3 Confidentiality

Secure multi-party computation can ensure the confidentiality of a single KPI, single peer-group computation, but our cloud application needs to handle multiple peer groups and multiple KPIs. Computation of multiple KPIs do not interfere with each other, since they are independent, but multiple peer groups may, since they may share common parties.

Let $\lambda_{i,j} = 1$ if X_i participates in the jth peer group and $\lambda_{i,j} = 0$. Since the number of customers in a peer group is known, the computation of the averages $s_1, \ldots, s_j$ can be written as

$$\vec{s} = \Lambda\vec{x}.$$

In case Λ is invertible or pseudo-invertible any party can compute the input from the output. In this case any secure computation (or homomorphic encryption) is unnecessary. A security proof for the plaintext protocol would be possible, but irrelevant. There is no security mechanism that may prevent learning the input from the output in this case and therefore there is infeasible to construct a confidentiality-preserving cloud application for this function. This problem arises if any subset of peer groups consists of linearly dependent sub-vectors except for the element which corresponds to the party with the revealed KPI value.

There are two possible countermeasures. First, the cloud service provider can make only peer group assignments that do not contain any such case. Second, the customer only participates in safe peer group assignments. The second case has the advantage that the customer can verify the confidentiality of his KPI and there is no explanation necessary in case participation is denied by the cloud service provider. Furthermore, the choice of participation by a party may make the assignment unsafe for another party. The only assignment scheme that prevents all these problems is to assign each party's KPI to at most one peer group, since already in case of two peer groups participation of one party can lead to linearly dependent sub-vectors.

4 Incentive-Compatible Mechanism

Any collaborative application implements a mechanism from game theory. Given a utility function rational players will behave in a way that leads to an equilibrium. There are several seminal results that we will revisit for our benchmarking application.

The fundamental problem of confidentiality-preserving computation is that the input data can no longer be audited by an external party. The cloud service provider and other customers have no assurance that a party submitted correct data. Even in the malicious security model there is no security mechanism for enforcing truth telling. A natural solution is to investigate the incentives of the players and whether it is in their best interest to provide correct data. Game theoretic analysis can assist when assuming rational players.

The first observation is that we need to assume that all players are interested in obtaining the correct result of the computation. Would just some players be not interested in obtaining the correct result, they would not provide the correct input leading to an equilibrium where no player is inclined to provide correct input.

Problems arise when we assume that players are also interested in obtaining information about the other players' input or withholding the result from them, i.e. a mixed utility function.

The first seminal result is a specification of non-cooperatively computable functions (Shoham and Tennenholtz 2005). Loosely speaking, a non-cooperatively computable functions is one that is neither dominated by one's input (i.e. can be computed from one's input) nor invertible. In case of invertible functions players would be inclined to provide false input, then reverse the function and recomputed it with correct input. The conclusion of Shoham and Tennenholtz (2005) is that sum, average and maximum are not non-cooperatively computable, while median and top quartile are.

The second seminal result is that the equilibrium in secure multi-party computation protocols is to not compute, since the players are not inclined to exchange the result shares (Halpern and Teague 2004; Abraham et al. 2006). Every party that sends its result shares to other parties has only those parties profiting, i.e. it does not act in its best interest, since it only needs the shares from the other parties to obtain the result. Halpern and Teague (2004) presented the first protocol for overcoming this problem. Later the result was improved by Abraham et al. (2006).

Fortunately this second dilemma does not arise in cloud computing. In cloud computing we can assume that the service provider has an interest in providing the correct result to its customers in order to maintain a sustainable business (Kerschbaum 2009). The service provider can then act as an intermediary forwarding the results to all parties, since it profits from forwarding and not only from the result.

In confidentiality-preserving cloud computing on encrypted data another dilemma arises. Assume the service provider has computed the result, but it is still encrypted under the homomorphic encryption scheme where only the customers hold the key. In a collaborative application there is usually only one result, but multiple customers that all want to obtain the same result. Also decryption in the computation phase and result publishing are usually decoupled. The following simple attack is rational for the service provider: He submits to all parties their input ciphertext, obtains the plaintext and computes the result on the plaintext which he publishes. In this case he has delivered the correct result (i.e. maximized its utility function) and obtained all input information.

There are two options for preventing this attack. First, the service provider may only ask one customer to decrypt the result, but the customers are not supposed to communicate among each other. So the chosen one cannot determine it is the only one. Second, the service provider submits the same ciphertext to all customers. This can be verified using signatures from the customers. This prevents any attack violating confidentiality as long as the correct result is delivered (Kerschbaum 2008).

5 Key Distribution and Management

Homomorphic encryption requires a key to protect the information. In order to process the ciphertexts each ciphertext needs to be encrypted under the same key. This key needs to be protected, since if it is leaked, confidentiality of the data is at risk.

The architect has the following design choices

- single or threshold key,
- static or dynamic key,
- key agreement secure against passive or active adversaries.

The key agreement should be chosen secure against the same type of adversary as the secure computation protocol, since the composition will be secure only against the weakest one. In Sect. 4 we have designed our protocol to be secure against an economically motivated active adversary and therefore key agreement must be secure against a (stronger) active adversary.

Key agreement protocols secure against active adversaries require a trusted third, e.g. the certificate authority in public key infrastructures (PKI). Our key agreement protocol therefore requires one as well and the service provider cannot be this trusted party, since this would reduce security to security against passive adversaries (i.e. the trusted third party is assumed to behave as expected).

If a third party needs to be involved in key agreement, two service providers would be required. It can therefore be advisable to choose static keys and limit the third party to a one-time interaction. Dynamic keys require a key agreement protocol for each round of computation and should only be used in case the key agreement can be performed without the interaction of the trusted third party, e.g. using PKI. Note that, public keys in PKI, as any static, break anonymity, since they can be used a static pseudonym.

The impact of the choice between single and threshold key is not always clear cut. On the one hand a single key if leaked compromises the security of the entire system, but this is more critical for static keys which have to be stored on disk. Nevertheless any collusion between service provider and any other party breaks the security of the system, even in case of dynamic keys. On the other hand a threshold key does not automatically imply security against collusions between service provider and other parties, because intermediate results are often necessary to enable the functionality using partially homomorphic encryption. In the case of benchmarking securing the protocol against collusions between service provider and other parties has proven to make the protocol too inefficient, even when using a threshold key.

In summary, we chose a single, dynamic session key for the homomorphic encryption which is established using a key agreement protocol secure against active adversaries without the active involvement of the PKI certificate authority as a trusted third party.

6 "Homomorphic" Comparison

When computing statistics, a key problem next to addition is comparison. Comparison is essential for statistical quantities, such as the median. In case of the benchmarking platform the inputs are encrypted using homomorphic encryption.

Let $E(x)$ denote the encryption of $x \in Z_n$ in the homomorphic encryption system. The problem is to construct an algorithm or protocol, such that given $E(x)$ and $E(y)$ the service provider can compute $E(c)$ where $c = x < y$ (represented by elements from Z_n). Clearly fully homomorphic encryption can perform this task, but also Fischlin (2001) described how this can be performed using the homomorphic encryption system by Sander et al. (1999). In this encryption scheme each bit of the input is encrypted as its own ciphertext, such that additions of integers are no longer possible. We therefore propose a solution using additive homomorphic encryption.

Our solution is based on multiplicative blinding. We map the integers from $[-n/2, n/2]$ to the integers $[0, n-1]$ similar to two-complement notation, i.e. $0 \to 0$, $n/2 \to n/2$, $-n/2 \to n/2+1$ and $-1 \to n-1$. We can then encrypt the integers $[-n/2, n/2]$ using homomorphic encryption (*modulo n*).

Given $E(x)$ and $E(y)$ the service provider will choose random numbers $0 < r$ and $0 \leq r' < r$ and compute

$$\begin{aligned} E(c) &= (E(x)E(y)^{-1})^r E(r') \\ &= E(rx - ry + r'). \end{aligned}$$

It holds that

$$x < y \Leftrightarrow c < 0.$$

6.1 Information-Theoretic Confidentiality

Correctness of the homomorphic comparison is easy to verify, but security is difficult. We hide the difference $a = x - y$ of the plaintexts using multiplication with a random number r.

$$b = ra + r'.$$

Our security model is information-theoretic and not cryptographic. Consequently we can ask the question what does b reveal about a. First, note that if the bit length of r is likely to be known, then the bit length of b reveals the bit length of a, since

$$\begin{aligned} \lceil lgr \rceil + \lceil lga \rceil - 1 &\leq \lceil lgb \rceil \\ &\leq \lceil lgr \rceil + \lceil lga \rceil. \end{aligned}$$

We therefore choose the bit length of the number r first and then the remaining bits uniformly.

One can now assess the security of the homomorphic comparison under a number of attacks. In the simplest case a ciphertext-only attack is performed on a given b. The information theoretic measure for leakage of b about a is the conditional entropy

$$\begin{aligned} H(A|B) = -\Sigma p(B=b)\Sigma p(A=a|B=b) \\ \times \log p(A=a|B=b). \end{aligned}$$

Unfortunately we cannot assess it analytically, but we can sample it. Kerschbaum (2010) has conducted a large scale experiment and estimates a leakage of *0.11* bit for a single sample with *16* bit a and *512* bit r chosen using normal distribution.

7 System Architecture

7.1 Components and Use Cases

So far we have only considered the cryptographic aspects of the benchmarking protocol, but the entire system consists of several components and use cases (Kerschbaum 2007). We briefly describe here the use cases of our benchmarking platform.

- **Registration**: A party approaches the service provider to participate in the benchmarking process. This includes creation of account, payment, in case of a static key its distribution and triggers peer group formation. The participants of registration are the party, the service provider and the certificate authority. The only pre-condition for registration is a system setup between the trusted third party (certificate authority) for key agreement and the service provider. In case of PKI this is the installation of the root certificate at the service provider. The post-condition of registration is that the peer group formation use case can afterwards be performed.
- **Peer Group Formation**: Registration triggers peer group formation where the party is assigned a peer group to benchmark against. We consider this a separate use case due to the complexity of the function and different implementation alternatives involving different parties. We describe an algorithm to assign each party a peer group. Recall that for confidentiality reasons each party can be in at most one peer group. The peer group formation may affect the peer group assignment for parties other than the one that just joined, but we can perform formation completely automatic, such that the only participant of this use case is the service provider. The pre-condition is that sufficiently many parties have registered at the service provider. The post-condition is that the remaining two use cases of statistics computation and statistics retrieval can be performed.
- **Statistics Retrieval**: Once a party has registered and been assigned a peer group, it can start to benchmark. Benchmarking involves an analysis of one's KPIs in comparison to the peer group for which the statistics need to be retrieved from the service provider. Note that a party can start to benchmark right after peer group formation and does not have to wait for statistics computation, i.e. it can retrieve the statistics even if its own KPIs have been involved in the computation. This increases the benefit for the customer by eliminating waiting time. The service

provider maintains a database of the statistics which the party may access. The pre-condition is that peer group formation has been completed. There is no post-condition.

- **Statistics Computation**: Triggered by the service provider the benchmarking protocol starts to compute the statistics. The challenge is, of course, to perform the computation on encrypted data. The computation is decoupled from the retrieval, i.e. off-line. The participants of the protocol are all members of a peer group and the service provider. Pre-condition is that peer group formation has been completed and there is no post-condition.

The principle to decouple user interaction (benchmarking) from secure computation is very vital in handling the performance constraints of computation on encrypted data. It is a deviation from most cloud computing approaches, but has been adopted in the other deployed secure computation as well (Bogetoft et al. 2009).

7.2 Peer Group Formation

Peer group formation is triggered by the registration of a new party, but performed non-interactively by the service provider. The goal of peer group formation is to identify the groups that are likely to benefit most from benchmarking against each other. The assumption is that the more similar peer group members are, the better the benchmarking result.

We view the peer group formation problem as a data classification problem. We try to assign each party a classifier, its peer group. First during registration each party publishes non-sensitive characteristics about itself to the service provider. These characteristics should be rather stable for a company but vary between companies. They should be relevant to the business model of the company and their combination should be almost identifying, i.e. unique.

These characteristics form an m-dimensional space in which each party represents a point. The data classification problem is to divide the space, such that the data points in the resulting subspaces have low spatial extension. This is commonly called data clustering.

The fastest partitional data clustering algorithm is k-means clustering. Given a parameter k, it identifies k clusters minimizing the average distance to the cluster center. K-means clustering is a randomized algorithm, i.e. it does not necessarily find the optimum, but its output ranges over a number of near-optimal solutions depending on a random input parameter.

K-means clustering, as any standard clustering algorithm, is not directly applicable to peer group formation. In order to preserve the confidentiality of the KPI values each peer group must be of minimum size. This constraint must be observed in the data clustering algorithm. Bennett et al. propose using linear programming (LP) in order to minimize the distance to the cluster centers while observing the constraints (Bennett et al. 2000). Unfortunately for the problem sizes considered in peer group formation LP does not scale and we need a solution that preserves the superior performance of k-means clustering.

Kerschbaum (2007) introduces k, l-means clustering which is a greedy variation of k-means clustering where each cluster has to have at least size l. The basic idea is as follows. In each round of the algorithm we iterate through the clusters until each cluster has size l. If a cluster has less than l members it "grabs" the nearest ones until it has l members. We keep track if a data point has been reassigned to a new cluster and each data point may be reassigned at most once in order to prevent infinite loops. At the end of this iteration each cluster has at least l members.

We performed an analysis of the quality of peer group formation. In collaboration with industry experts we defined a measure for quality of a peer group. The maximum distance in any characteristic is used a quality measure: The smaller this distance, the better the cluster. We evaluated k, l-means clustering using different distance norms and cluster sizes. Astonishingly the distance norms made little difference and we settled for the Euclidean distance, but the clustering performed better if the minimum cluster size l was increased if there were artificially introduced cluster in the population. This implies that k, l-means clustering performs better than k-means clustering, if the size l of the clusters is known.

As described in Sect. 7.1 peer group formation is triggered by a new party joining the service provider. It is therefore sufficient to assign this new party to an existing cluster instead of performing a full data clustering for reassignment. This also limits the number of peer groups where statistics need to be recomputed. Consequently we simply choose the best peer group for the new party and split this group into two if exceeds a threshold. For splitting we can use k, l-means clustering again. The difference between the quality of incrementally assigning peer groups and the quality of always performing a full peer group assignment is negligible.

8 Performance

Performance for computations on encrypted data remains critical and very few experimental evaluations exist. Our protocol has quadratic $O(n^2)$ computation and communication cost. We therefore implemented the benchmarking protocol for the statistics computation use case. Following current practice in industry we implemented the protocols over web services as a communication mechanism (Kerschbaum et al. 2009). Each client and server runs its own web application server with a web application for the protocol implementation. The implementation is entirely written in Java with the crypto modules taking advantage of the BigInteger arithmetic of Java's standard library.

We evaluated our implementation in an experimental study in the following setup. The service provider was deployed on a Pentium 4 3.2 GHz machine with 1.5 GB of memory. All clients were deployed on a Xeon Dual 3.6 GHz machine with 8 GB of memory. Between the client and server machine we deployed a WAN emulator as a router. The WAN emulation software was the dummynet package for FreeBSD (Rizzo 1997). All machines are physically connected via a non-dedicated Gigabit Ethernet switch.

In a first experiment we increased the number of clients from 5 to 45 in steps of 5 and we increased the latency on the network connection from 0 to 100 milliseconds in steps of 25. The latency or delay is used to simulate WAN conditions as over the Internet. A delay of 100 ms results in a round-trip time (RTT) of 200 ms, which roughly corresponds to the RTT between Germany and Japan over the Internet.

It became apparent that in this experiment the network performance plays a significant role. For 45 clients and a delay of 100 ms the time spent for communication is almost half of the overall running time.

For the next experiment we modified the server implementation. Instead of sequentially calling each client, we create a thread for each client that asynchronously handles the communication. This is made possible by our benchmarking protocol, since each round for each client only requires input of the previous round and all clients can run concurrently. The order of the clients does not matter for the protocol's semantics. The necessary synchronization between rounds is achieved using a barrier.

We conducted the same series of experiments for the concurrent implementation. We increased the number of clients and independently increased the network delay. The results are depicted in **Fig.** 2. The impact of the network performance has significantly decreased and is almost negligible compared to the impact of the computational effort. For 45 clients and a delay of 100 ms the time spent for communication is only 6% of the overall running time.

We conclude that computation on encrypted in the cloud can be implemented, such that its performance is almost independent of the network performance, i.e. for the overall performance it nearly does not matter whether the clients are located on the same LAN or half-way around the world over the Internet.

9 Related Work

Collaborative benchmarking service platforms have been first proposed and implemented by Bogetoft and Nielsen (2005). In addition to parametric (statistical) they implemented non-parametric benchmarking using data envelope analysis (DEA). DEA is based on linear programming which is currently out of reach of practical secure computation, although theoretical protocols exist (Li and Atallah 2006; Toft 2009).

Secure computation for benchmarking has been first proposed by Atallah et al. (2004). They implement a number of secure division protocols for implementing time series. In our case of statistical benchmarking time series can be computed on the local data and the computed statistics, both in plain text. Encryption is unnecessary, since the protection of the KPIs stems from the aggregation as statistical quantities.

There also exist a number of secure computation protocols for computing statistics. Most notably (Aggarwal et al. 2004) describes an efficient protocol for computing the median. As with any secure computation protocol, the model is based on distributed, mutually distrustful parties, and it is not clear how to apply it in the cloud service provider case. When applied straightforwardly it is no more efficient than our protocol.

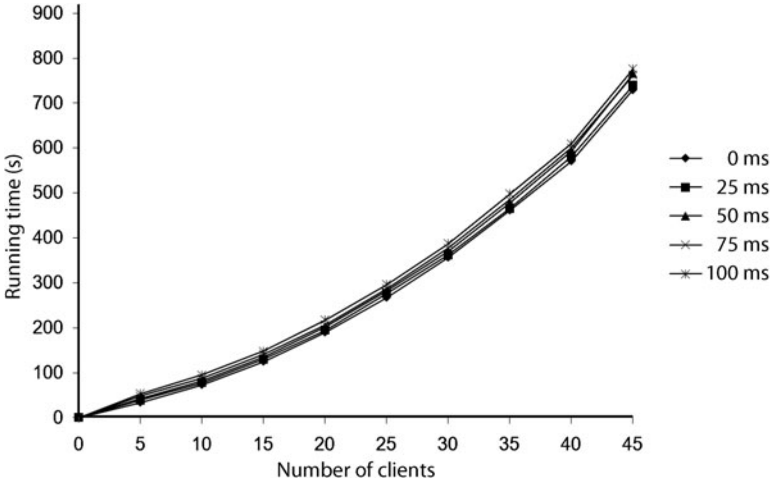

Fig. 2 Performance of benchmarking on encrypted data

There are also a few protocols for securely computing statistics using a central service provider. Di Crescenzo (2000) describes a protocol for the sum and Di Crescenzo (2001) describes a protocol for the maximum. Both use a semi-honest service provider and thereby avoid the problem of key agreement, but, of course, introducing a certificate authority does not make the protocols secure in a stronger security model.

One challenge is to combine the protocols for the different statistics and remain efficient. To the best of our knowledge the homomorphic comparison is the first and most efficient proposal for a comparison operation on homomorphically encrypted data (Kerschbaum and Terzidis 2006). A related proposal has been made later by Sakuma and Kobayashi (2007). Their security "proof" is questionable, since it does not compare the probability of the different cases, but only shows that there is at least one for each combination of numbers. Therefore the leakage as proposed by us provides a much better analysis.

Our homomorphic comparison competes with a huge number of secure inequality comparison protocols, to name a few (Damgard et al. 2008; Fischlin 2001; Yao 1986). These protocols implement secure "greater-than" comparison of two private values, also known as Yao's millionaires' problem after the initial protocol by Yao (1986). The currently fastest protocol by Damgard et al. uses one public value and we have shown that the fastest with two private values by Fischlin is significantly slower than our homomorphic comparison (Kerschbaum and Terzidis 2006).

The only other practically deployed secure computation has been described by Bogetoft et al. (2006; improved in 2009). It implements a secure auction protocol. They implement the same decoupling between user interaction and computation as we do and limit computation to a constant sized set of servers, i.e. they compute using several, but mutually distrustful service providers. While this model is not readily applicable to the cloud, it may give rise to a new type of service provider that obviously participates in a secure computation. Their reported computation times are on the same order as ours.

There exist a number of other frameworks for secure computation which are based on general protocols. The first for secure two-party computation was FairPlay (Malkhi et al. 2004). Later VIFF (2010), ShareMind (2010) and FairPlayMP (Ben-David et al. 2008) follow for secure multi-party computation. Since all of them are based on general secure computation protocols which can implement any functionality, but are not optimized for the functionality, it remains to be seen whether they will ever be practically used due to the high computational requirements.

To the best of our knowledge no computations on encrypted data in the cloud using any form of homomorphic encryption (Gentry 2009) have been reported so far. Our experiences building the benchmarking platform are therefore the first and we hope they provide sufficient insight in order to inspire future research in the topic.

10 Summary and Outlook

In this paper we examined confidentiality-preserving cloud computing, i.e. the computation on encrypted data in the cloud. We presented the available design choices using an example cloud application for collaborative benchmarking. It addresses the security threat of a confidentiality breach by the service provider by computing on encrypted data thereby reducing the necessary trust assumptions. We compare the different models of homomorphic encryption and secure computation and implemented a hybrid model for improved performance. Our experience from building this cloud application revealed a number of insights some contrary to common belief.

First, *encrypting data is not enough*. The information revealed by the result may imply the input by inverse computation. It is therefore necessary to carefully design and evaluate the functionality implemented on encrypted data. In our case it was necessary to restrict each party to participate in at most one peer group per KPI. We can show that if the size of the peer group is large enough, the KPIs remain private.

We note that there are no formal tools or methods yet for this type of analysis and all work needs to be performed manually by skilled researchers and engineers. This is a clear opportunity for future research in providing formal methods and tools for verifying the security of the application.

Second, *security should match the business objectives*. It is difficult, if not impossible, to prevent all attacks by malicious adversaries on encrypted cloud computing, since the input is encrypted opening wide the door to denial-of-service attacks. It is therefore necessary to align the objectives of the cloud application with the economic objectives of the participants.

In our benchmarking application we use incentive-compatible statistics and enhance the security against economically motivated adversaries. This implies that some simple protocols secure against only passive adversaries are no longer applicable, such as key exchange using the cloud service provider, and we need a trusted third party (certificate authority) for the key exchange. Nevertheless, to the best of our knowledge, our benchmarking protocol is the first instance of a protocol that is more efficient against a rational attacker than a malicious one. We believe that there will be many more such cases and look forward to this research and development.

Third, *inequality comparison of homomorphically encrypted integer plaintexts is possible*. Our result predates the development of fully homomorphic encryption and has applications outside of comparison. Our comparison method does not fit existing formal security models, but we devised a new one and hope to see many more research results in this area. In particular it seems challenging to extend the method to other computations beside comparison and general results are required.

Fourth, *computational performance will remain the bottleneck*. While we can show that performance for benchmarking is acceptable, more complicated computations are likely not. The benchmarking functionality can be computed locally in *15* ms. In comparison to our benchmarking protocol this is a factor of roughly *60.000*. This comparison is not entirely fair, since a secure cloud application always implies some overhead for secure transmission and storage of data, but we could show that network performance can be effectively reduced by parallelization. The main obstacle therefore remains computational performance.

In this respect our homomorphic comparison also significantly improves over other homomorphic encryption, such as Gentry's general construction (Gentry 2009) or Fischlin's bitwise comparison (Fischlin 2001), because our performance matches that of partially (additive) homomorphic encryption.

We anticipate that the improvement of performance will remain the major research challenge for computation on encrypted data. In fact it is a question of simple economics whether the application justifies the expenditure for the computational capacity and response wait time. In case cryptography does not provide sufficiently efficient solutions sufficiently fast, industry is likely to look for alternatives. In the current boom of cloud computing this may be a missed opportunity, since there seems to be no technical alternatives for preventive measures against confidentiality breaches by the service provider.

Abstract

Florian Kerschbaum

Secure and Sustainable Benchmarking in Clouds

A Multi-Party Cloud Application with an Untrusted Service Provider

Cloud computing entails a novel security threat: The cloud service provider is entrusted with the data of all its customers. This may not be sustainable for highly confidential data. Encryption, or more generally cryptography, may provide a solution by computing on data encrypted by the customers. While this solution is theoretically appealing, it raises a number of research questions in information system design.

Using the example of collaborative benchmarking the author presents and evaluates an exemplary design and implementation of a cloud application that operates only on encrypted data, thus protecting the confidentiality of the customer's data against the cloud service provider. The cloud application computes common statistics for benchmarking without disclosing the individual key performance indicators.

Benchmarking is an important process for companies to stay competitive in today's markets. It allows them to evaluate their performance against the statistics of their peers and implement targeted improvement measures.

Keywords: Cloud computing, Encryption, Cryptography, Homomorphic encryption, Secure multi-party computation, Benchmarking, Collaborative business applications

Fifth, *separating user interaction and computation may provide an alternative.* In applications, such as our collaborative benchmarking, that allow this separation it may provide an alternative way of designing the system that a user can accept. Furthermore, since the service provider can then schedule the computation this enables better load balancing. Already the first application appeared (Bogetoft et al. 2009) and we anticipate many more following this design choice.

References

Abraham I, Dolev D, Gonen R, Halpern JY (2006) Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In: Proc 25th ACM symposium on principles of distributed computing, pp 53–62

Aggarwal G, Mishra N, Pinkas B (2004) Secure computation of the *k*th-ranked element. In: Proc Eurocrypt, pp 40–55

Atallah M, Bykova M, Li J, Frikken K, Topkara M (2004) Private collaborative forecasting and benchmarking. In: Proc ACM workshop on privacy in an electronic society, pp 103–114

Ben-David A, Nisan N, Pinkas B (2008) FairplayMP: a system for secure multi-party computation. In: Proc 15th ACM conference on computer and communications security, pp 257–266

Bennett K, Bradley P, Demiriz A (2000) Constrained K-means clustering. Microsoft technical report

Ben-Or M, Goldwasser S, Wigderson A (1988) Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proc 20th ACM symposium on theory of computing, pp 1–10

Bogetoft P, Christensen D, Damgard I, Geisler M, Jakobsen T, Kroigaard M, Nielsen J, Nielsen J, Nielsen K, Pagter J, Schwartzbach M, Toft T (2009) Secure multiparty computation goes live. In: Proc 13th international conference on financial cryptography and data security, pp 325–343

Bogetoft P, Damgard I, Jakobsen T, Nielsen K, Pagter J, Toft T (2006) A practical implementation of secure auctions based on multiparty integer computation. In: Proc 10th international conference on financial cryptography and data security, pp 142–147

Bogetoft P, Nielsen K (2005) Internet based benchmarking. Group Decision and Negotiation 14(3):195–215

Cramer R, Damgard I, Nielsen J (2001) Multiparty computation from threshold homomorphic encryption. In: Proc Eurocrypt, pp 280–299

Crotts J, Pan B, Dimitry C (2006) Hospitality performance index: a case study of developing an internet-based competitive analysis and benchmarking tool for hospitality industry. In: Proc conference of travel and tourism research association

Damgard I, Geisler M, Kroigard M (2008) Homomorphic encryption and secure comparison. International Journal of Applied Cryptography 1(1):22–31

Damgard I, Jurik M (2001) A generalisation, a simplification and some applications of pailliers probabilistic public-key system. In: Proc international conference on theory and practice of public-key cryptography, pp 119–136

Di Crescenzo G (2000) Private selective payment protocols. In: Proc 4th international conference on financial cryptography and data security, pp 72–89

Di Crescenzo G (2001) Privacy for the stock market. In: Proc 5th international conference on financial cryptography and data security, pp 269–288

Eurich M, Oertel N, Boutellier R (2010) The impact of perceived privacy risks on organizations' willingness to share item-level event data across the supply chain. Electronic Commerce Research 10(3–4):423–440

Fischlin M (2001) A cost-effective pay-per-multiplication comparison method for millionaires. In: Proc RSA security cryptographer's track, pp 457–471

Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: Proc 41st ACM symposium on theory of computing, pp 169–178

Goldreich O (2002) Secure multi-party computation. http://www.wisdom.weizmann.ac.il/~oded/pp.html. Accessed 2011-02-07

Goldreich O, Micali S, Wigderson A (1987) How to play any mental game. In: Proc 19th ACM symposium on theory of computing, pp 218–229

Halpern J, Teague V (2004) Rational secret sharing and multiparty computation: extended abstract. In: Proc 36th ACM symposium on theory of computing, pp 623–632

Kerschbaum F (2007) Building a privacy-preserving benchmarking enterprise system. In: Proc 11th IEEE international EDOC conference, pp 87–96

Kerschbaum F (2008) Practical privacy-preserving benchmarking. In: Proc 23rd IFIP international information security conference, pp 17–31

Kerschbaum F (2009) Adapting privacy-preserving computation to the service provider model. In: Proc 1st IEEE international conference on privacy, security, risk and trust, pp 34–41

Kerschbaum F (2010) A privacy-preserving benchmarking platform. Dissertation. Karlsruhe Institute of Technology

Kerschbaum F, Dahlmeier D, Schröpfer A, Biswas D (2009) On the practical importance of communication complexity for secure multi-party computation protocols. In: Proc 24th ACM symposium on applied computing, pp 2008–2015

Kerschbaum F, Terzidis O (2006) Filtering for private collaborative benchmarking. In: Proc international conference on emerging trends in information and communication security, pp 409–422

Li J, Atallah M (2006) secure and private collaborative linear programming. In: Proc 2nd international conference on collaborative computing, pp 1–8

Malkhi D, Nisan N, Pinkas B, Sella Y (2004) Fairplay—a secure two-party computation system. In: Proc USENIX security symposium, pp 287–302

Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: Proc Eurocrypt, pp 223–238

Rizzo L (1997) Dummynet: a simple approach to the evaluation of network protocols. ACM Computer Communication Review 27(1):31–41

Sakuma J, Kobayashi S (2007) A genetic algorithm for privacy preserving combinatorial optimization. In: Proc conference on genetic and evolutionary computation, pp 1372–1379

Sander T, Young A, Yung M (1999) Non-interactive crypto-computing for NC1. In: Proc 40th IEEE symposium on foundations of computer science, pp 554–567

ShareMind (2010) http://research.cyber.ee/sharemind/. Accessed 2011-02-07

Shoham Y, Tennenholtz M (2005) Non-cooperative computation: boolean functions with correctness and exclusivity. Theoretical Computer Science 343(1–2):97–113

Toft T (2009) Solving linear programs using multiparty computation. In: Proc 13th international conference on financial cryptography and data security, pp 90–107

VIFF (2010) http://www.viff.dk/. Accessed 2011-02-07

Yao A (1986) How to generate and exchange secrets. In: Proc 27th IEEE symposium on foundations of computer science, pp 162–167