2019 International Conference on Networked Systems (NetSys)

An Enhanced Approach to Cloud-based Privacy-preserving Benchmarking

Kilian Becher
Chair of Privacy and Data Security
TU Dresden
Dresden, Germany
kilian.becher@tu-dresden.de

Martin Beck
Chair of Privacy and Data Security
TU Dresden
Dresden, Germany
martin.beck1@tu-dresden.de

Thorsten Strufe
Chair of Privacy and Data Security
TU Dresden
Dresden, Germany
thorsten.strufe@tu-dresden.de

***Abstract*—Benchmarking is an important measure for companies to investigate their performance and to increase efficiency. As companies usually are reluctant to provide their key performance indicators (KPIs) for public benchmarks, privacy-preserving benchmarking systems are required. In this paper, we present an enhanced privacy-preserving benchmarking protocol, which we implemented and evaluated based on the real-world scenario of product cost optimisation. It is based on homomorphic encryption and enables cloud-based KPI comparison, providing a variety of statistical measures. The theoretical and empirical evaluation of our benchmarking system underlines its practicability.**

***Index Terms*—secure multi-party computation, benchmarking, key figure comparison, homomorphic encryption, oblivious transfer, privacy-preserving, cloud-based**

I. Introduction

Benchmarking is the comparison of key performance indicators (KPIs) of a company's peer group [17]. KPIs are statistical quantities that can be used for evaluating the performance of a company [17]. A peer group is a set of companies that take advantage of comparing KPIs [17]. Usually, the companies of a peer group are competitors of the same industry, which implies a demand for privacy of the KPIs [17]. We define a privacy-preserving benchmarking analysis as the process of comparing KPIs as secure inputs across different companies [9]. Every company learns how it performs compared to the other companies being involved but not their private KPIs [9].

One approach to privacy-preserving benchmarking is using a trusted third party (TTP) that conducts the calculation of a function $f(x)$ without revealing any private data. However, with mutually distrusting companies, finding a TTP might not be trivial [11]. An approach that does not require trust can be found in secure multi-party computation (MPC). In MPC, participants providing an input x_i are called players, those who compute $f(x)$ are called processors [12]. A participant can be both player and processor at the same time. An MPC is secure in the sense that the participants only learn the outputs, their own input, and what they can infer from that [9].

In this paper, we investigate approaches to adding privacy-preserving benchmarking to a product costing suite of the software company SAP. Such product costing software enables a company to calculate the costs of its products. This includes quotation costing as well as cost estimation from the project acquisition to the design and production to the disposal.

978-1-7281-0568-0/19/$31.00 ©2019 IEEE

To increase profit, a company, e.g. of the automotive industry, might want to reduce production costs. To do so, it needs to determine those areas with the best ratio between cost savings and optimisation effort, requiring knowledge of the company's performance. To make well-informed optimisation decisions, optimisation can be facilitated by comparing the company's production KPIs, e.g. assembly time of a car engine, to those of other companies of the industry, i.e. via benchmarking analyses. If according to the benchmarking results a company finds itself among the best performing, it might be rather expensive to further improve the compared KPI, e.g. assembly time. A performance below average might imply higher potential for cost savings. However, as companies might be reluctant to provide their confidential production KPIs, benchmarking needs to be conducted in a manner that ensures privacy of the KPIs and still provides the desired statistical measures. Such privacy-preserving benchmarking analyses could be repeated on a regular basis, e.g. once every quarter, to investigate performance development over time relatively to the industry.

The requirements for our system were defined given the corporate context. They are based on the requirements of an existing, TTP-based benchmarking suite of SAP, serving as a lower bound for functionality and performance. In this TTP-based system, benchmarks are conducted individually for each KPI at least once every six to seven months on predefined dates rather than on demand. It provides the statistical measures mean, bottom quartile, top quartile, and best-in-class. The latter is the mean of the top quarter of the sorted list of inputs. As in the existing benchmarking system and similar to Atallah et al. in [2], we assume companies comparing their product costing KPIs to be interested in correct results and therefore to behave honestly. This can reasonably be assumed as companies need such benchmarking results for well-informed, far-reaching business decisions. These results are only correct if every participant follows the protocol. Hence, we require our system to provide input privacy against semi-honest adversaries. To guarantee that each company gets the same, correct results, output integrity is required. Similar to the TTP-based system, our system should provide anonymity among the players in the sense that participants must not be

referred to with any persistent identifier during benchmarking [16]. This might be of interest in case of computing benchmarks only for a subset of a peer group. Further aspects like availability were beyond the scope of our considerations.

Following the company's cloud strategy, our system should run as a cloud service. This provides scalability and availability and can be assumed to enable protocol execution even for large peer groups [1]. In the TTP-based system, such large peer groups contain up to 300 players, which is sufficient for the product costing context. As suggested in Kerschbaum's guideline for non-functional requirements of a benchmarking platform [16], a benchmark should at most take 24 hours.

Our main contributions are the construction and implementation of a privacy-preserving benchmarking system in the context of product cost optimisation. Furthermore, we conducted security and complexity analyses and an extensive performance evaluation of this protocol in the given context.

Focusing on the technical feasibility of privacy-preserving benchmarking systems, we first performed a state-of-the-art analysis for possible approaches (Section III). We then selected one approach, adapted it to better suit the product costing context, designed a prototype (Section IV), and evaluated it extensively regarding the given requirements (Section V).

II. Preliminaries

A. Oblivious transfer

In an oblivious transfer (OT) protocol, a player P_1 has k secret messages $m_1, ..., m_k$ with $k \geq 2$. A player P_2 wants to select and receive message m_i without P_1 learning the value i [16]. Furthermore, P_1 does not want P_2 to learn any message apart from m_i [16]. We denote such a protocol by $P_1 \xrightarrow{\text{OT}} P_2$.

B. Homomorphic encryption

Assume a cryptosystem with a (randomised) encryption function $E(\cdot)$ and a decryption function $D(\cdot)$ [15]. Homomorphic cryptosystems enable computations for secret values $x_1, ..., x_n$ based on their ciphertexts $E(x_i), ..., E(x_n)$ without needing the decryption key [17]. Applying an operation to such ciphertexts yields the ciphertext of the result of a corresponding homomorphic operation applied to the plaintexts [17]. Partially homomorphic encryption (PHE) schemes enable one operation on the plaintext, e.g. addition or multiplication [16]. Fully homomorphic encryption (FHE) schemes allow for the computation of arbitrary functions, e.g. by providing both addition and multiplication [10]. For example, Paillier's asymmetric, additively (partially) homomorphic encryption scheme has the properties given in Equations (1) and (2) [17].

$$D\left(E\left(x_1\right) \cdot E\left(x_2\right)\right) = x_1 + x_2 \tag{1}$$

$$D\left(E\left(x_1\right)^{x_2}\right) = x_1 \cdot x_2 \tag{2}$$

Homomorphic semantically secure cryptosystems provide rerandomisation of ciphertexts as follows [16].

$$E(x_i + 0) = E(x_i) \cdot E(0) = E'(x_i) \tag{3}$$

With high probability, $E(x_i) \neq E'(x_i)$ is provided such that $E(x_i)$ and $E'(x_i)$ are computationally indistinguishable [16].

III. Related Work

We assess the suitability of several existing approaches and protocols for privacy-preserving benchmarking regarding the requirements described above. It covers the generic approaches of garbled circuits, secret sharing, and homomorphic encryption as well as custom protocols. The former can be used for computing arbitrary functions while the custom protocols only enable computation of predefined functions.

A. Garbled circuits

The protocol presented by Beaver, Micali, and Rogaway in [3] implements the garbled circuits approach for multiple players. The players create a Boolean circuit that implements $f(x)$. Even though one might consider garbled circuits to be impractical [14], it was shown that they can compete with custom protocols regarding efficiency [13]. These findings were driven by improvements of the garbled circuits approach, such as free XOR and garbled row reduction (see [14, 18, 19]). For specific tasks, e.g. calculation of the mean, one only needs to create a garbled circuit that implements the corresponding operations instead of developing an entire protocol.

The protocol of [3] consists of two phases, one for jointly generating a common garbled circuit C together with the garbled input and a second for publishing and evaluating this circuit. Since the protocol implements the generic approach of garbled circuits, it can be used for calculating any computable function [3]. However, up-front effort is required to create the necessary circuits. The most important drawback of this approach is its non-centralised communication model. It requires pairwise communication between the players, which precludes inherent anonymity among them. Therefore, the approach presented in [3] does not suit the requirements properly.

B. Linear secret sharing and homomorphic encryption

This approach is a combination of linear secret sharing and homomorphic encryption. The idea of secret sharing is to split the secret values into shares and spread these shares among the players involved in the scheme [9]. The function $f(x)$ for these inputs can then be computed given the shares [9]. In this approach, the players P_i share their secret input values x_i among the set of n players using a linear secret sharing scheme like Shamir's scheme (see [20]). Each player P_j holds one share $[\![x_i]\!]_j$ for each of the n secret-shared values $[\![x_i]\!]$. Addition of shares is a linear operation and can be performed locally while multiplication requires a subprotocol that introduces overhead [12]. Such a subprotocol for two players is presented by Atallah et al. in [2]. The complexity of this subprotocol is exponential in n.

In its original form, the approach has a non-centralised communication model requiring pairwise communication between the players. Therefore, the anonymity requirement is not met. The most important drawback of this approach is the complexity of the multiplication subprotocol making it impractical for large n, i.e. large peer groups. Consequently, the approach described in this Section does not suit the requirements properly.

C. Fully homomorphic encryption

Given a FHE scheme with an encryption function $E(\cdot)$ and the encrypted secret values $E(x_1), ..., E(x_n)$, one can compute the encrypted result $E(y) = E(f(x_1, ..., x_n))$ for any efficiently computable function $f(x)$ without needing to decrypt [10]. Given that, one can build a secure MPC system where one player conducts the entire computation of $E(f(x))$ locally without learning anything about $x_1, ..., x_n$ or y.

In [10], Gentry presents the first ever FHE scheme. This seminal work started a new research field and led to many significant improvements. However, even enhanced approaches to FHE schemes, such as the one presented by Brakerski, Gentry, and Vaikuntanathan in [6], have complexity at least polynomial in the size of the respective circuit with large constants. Therefore, this approach can reasonably be assumed to not suit the performance requirements properly.

D. Custom benchmarking protocol

In [16], Kerschbaum presents a secure MPC protocol designed for privacy-preserving benchmarking platforms that has constant cost[1], provides anonymity among the players, and is centralised. It requires a single server; the service provider P_S of the benchmarking platform who acts as a processor. The protocol enables the benchmarking platform to compute the statistical measures mean, variance, median, maximum, and best-in-class [16]. The protocol is based on an additively partially homomorphic encryption scheme.

The protocol natively implements the service provider model and therefore can ensure anonymity among the players as well as cloud suitability. Input privacy is ensured in the semi-honest model as well as in the constrained malicious model [16]. Output integrity is ensured via message authentication codes (MACs). With a fixed set of four rounds, its round complexity is constant. However, the protocol only offers a subset of the required statistical measures. The protocol's computational and communication complexity both are quadratic in n. This may prove critical for large peer groups.

E. Custom aggregation protocol

In [5], Bonawitz et al. present a protocol for secure aggregation. This protocol assumes two kinds of participants: one server P_S, acting as a processor, and a set of n players P_i, each providing a secret input x_i. Players only communicate with the server P_S, who acts as a mediator between players. Only P_S learns the output $y = \sum_{i=1}^{n} x_i$, i.e. the sum of the secret values. The players do not learn anything new [5].

Executed in a central server scenario, the protocol enables anonymity among the players as well as cloud suitability. Rank-based statistical measures such as quartiles cannot directly be computed using this protocol. Even though its round complexity is constant, the protocol has computational and communication complexity that is quadratic in the number of players n, which may prove critical for large n [5].

[1]Constant cost here means constant round complexity and constant, i.e. linear in the size of the security parameter κ, communication complexity. Both are independent of the peer group size [17].

F. Summary and selection of a suitable approach

The three generic approaches garbled circuits, secret sharing, and (fully) homomorphic encryption do not meet the requirements mostly due to their complexity. On top of that, their implementation in a centralised communication model would require additional effort and further increase the complexity. Otherwise, they would not meet the required level of anonymity. The less complex protocol presented in [5] is not suitable due to the lack of rank-based statistical measures. The most suitable approach is the privacy-preserving benchmarking protocol presented in [16]. Even though its computational and communication complexity of $\mathcal{O}(n^2)$ may prove critical for large peer groups, we build upon this protocol.

IV. Design

The privacy-preserving benchmarking protocol of [16] (see Section III-D) only offers a subset of the required statistical measures, i.e. mean, variance, median, maximum, and best-in-class. To provide the additional statistical measures bottom quartile bq and top quartile tq, the protocol had to be enhanced. The full adapted protocol is given in this Section.

Prior to the protocol execution, each player P_i learns the following two keys, e.g. with the help of a certificate authority as described in [16].

- K_{DEC}: The secret decryption key of the PHE scheme.
- K_{MAC}: The symmetric key of the MAC.

Every participant, including the service provider P_S, also learns the public encryption key K_{ENC} corresponding to K_{DEC}. The players use the same secret key for decryption. They directly communicate only with the service provider, via pairwise channels that are secured based on standard methods for protecting transmission over insecure networks [16].

A. Adapted protocol

Both the original protocol and our enhanced version are combinations of the techniques summation, rank computation, selection, and decryption [16], which will be introduced first.

Summation of encrypted values is conducted by multiplying the ciphertexts (see Equation (1)) [16]. For n values x_i, the encrypted sum is

$$E\left(sum\right) = E\left(\sum_{i=1}^{n} x_i\right) = \prod_{i=1}^{n} E\left(x_i\right). \tag{4}$$

Summation is required for calculation of the mean $mean$ (steps 1 and 2) and of the variance var (steps 13 and 14). The sum is blinded by adding a random value [16]. Since the players know the size n of the peer group, each player can compute the mean himself by dividing the sum by n [16].

Rank computation yields the rank of a value x_i in a list which is sorted in ascending order [16]. To achieve that, the value x_i is compared to each value x_j. For that comparison, the indices of the secret values are permuted by the permutations ϕ and ϕ' [16]. The assigned element of i is denoted by $\phi(i)$ while the one of j has index $\phi'(j)$. The difference between $x_{\phi(i)}$ and $x_{\phi'(j)}$ is blinded by two random values $1 \leq r_{2_j}$ and

$0 \leq r_{3_j} \ll r_{2_j}$ [16]. These are chosen individually for each j. The blinded difference

$$c_{\phi(i)_{\phi'(j)}} = r_{2_j} \cdot \left(x_{\phi(i)} - x_{\phi'(j)}\right) + r_{3_j} \tag{5}$$

is stored in the vector $c_{\vec{\phi(i)}}$. Counting the non-negative elements $pos(c_{\vec{\phi(i)}})$ of that vector yields the number of input values that are smaller than $x_{\phi(i)}$ [16]. Given that, its rank is

$$rank_{\phi(i)} = pos\left(c_{\vec{\phi(i)}}\right) + 1. \tag{6}$$

Now, due to the permutations, each player P_i holds the rank of the value $x_{\phi(i)}$ of some player $P_{\phi(i)}$ [16]. Rank computation is done once in the protocol (step 3). It is required for calculation of the the median med, the best-in-class bic, the maximum max, the bottom quartile bq, and the top quartile tq.

Selection is the act of computing the ciphertext of a secret value with specific, i.e. selected, rank [16]. First, P_S chooses a random value r_i individually for each player P_i and computes the ciphertexts $E(x_{\phi(i)}+r_i)$ and $E(r_i)$ [16]. That is, the value of P_i's assigned rank blinded by r_i and a 0 blinded by r_i. By using a 1-out-of-2 OT protocol (see Section II-A), a player P_i receives $E(x_{\phi(i)} + r_i)$, i.e. the blinded secret value, only if his assigned rank is the one selected [16]. The other players receive the blinded 0. As these OT steps are identical, we use a template to increase readability of the protocol. Let

$$\begin{aligned} OT_\circ(m, r_i, p, i) = \;& P_S \xrightarrow{\text{OT}} P_i : \\ & E_i^m = \begin{cases} E\left(x_{\phi(i)} + r_i\right) & \text{if} \quad rank_{\phi(i)} \circ p \\ E\left(r_i\right) & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

be the OT step, indexed with a binary comparison operator $\circ$ that takes a statistical measure descriptor m, a blinding parameter r_i, an array position in the sorted list p, and an index i. After the OT step, each player rerandomises the value he received by multiplying it by an encrypted 0 (see Equation (3)) and sends the product to the service provider [16]. The service provider then multiplies the encrypted values he received, removes the random values r_i, and gets the ciphertext of $x_{\phi(i)}$ [16]. Selection is required for computing the median, best-in-class, maximum, bottom quartile, and the top quartile. It occurs in steps 4 to 6C (OT), steps 10 to 12C (returning the selected values), and steps 15 to 17C (computing the results).

Decryption by the service provider is required since he is supposed to learn the result first [16]. Thus, he is able to round the result before sending it to the players [16]. To decrypt the result v, P_S first blinds the result with a random value r and sends the ciphertext $E(v+r)$ to the players [16]. Each player P_i decrypts the blinded result and sends the plaintext $v + r$ together with the corresponding MAC tag

$$\theta_i = MAC\left(v + r||i, K_{MAC}\right) \tag{8}$$

back to P_S [16]. The service provider gets v by subtracting the random value r. To prove that he sent the same encrypted, blinded result to every player, P_S computes the hash

$$\begin{aligned} h(&\theta_1 = MAC(v + r||1, K_{MAC}), \ldots, \\ &\theta_n = MAC(v + r||n, K_{MAC})) \end{aligned} \tag{9}$$

of the MAC tags θ_i he received by using a cryptographic hash function [16]. Together with the result v, P_S sends this hash to the players. Each P_i then computes the MAC tags and the hash and compares the hash to the one received from the service provider and obtains a validation bit v_{s_i} [16]. This bit, where s indicates the protocol step, is 1 in case of successful hash validation and 0 otherwise. It states whether the service provider has sent the same statistical measure to each P_i [16]. Decryption is required for each of the statistical measures mean, variance, median, best-in-class, maximum, bottom quartile, and top quartile [16]. It occurs in steps 2 and 14 to 17C (sending encrypted results), steps 7, 8, and 19 to 26C (returning decrypted, blinded results), steps 9 and 27 to 30C (sending decrypted results), and steps 18 and 31 to 34C (sending the hashed MAC tags).

Based on these preliminaries, our full enhanced protocol is given below in Table I together with descriptions of the steps we added for the bottom quartile and top quartile computation. These steps are marked with the letter "B" and "C" in the step label, respectively.

a) Round 1 (step 1): Each player P_i sends his encrypted input to the service provider P_S.

b) Round 2 (steps 2-13): The service provider computes the encrypted, blinded sum of the input values and sends it to the players P_i. Furthermore, P_S conducts a rank computation after which each player has the rank of some player P_j's input value. Given that rank, each P_i receives either an encrypted, blinded statistical measure or an encrypted random value via OT depending on whether his assigned rank fits the respective selection criterion. This is repeated for each of the statistical measures median, best-in-class, maximum, bottom quartile, and top quartile. Afterwards, the players decrypt the blinded sum they received and send it back to P_S together with a MAC tag of the blinded sum. Furthermore, P_S sends the sum to each P_i. Then, each player rerandomises his OT step outputs and sends them back to the service provider. Then each player computes the squared difference between his input and the mean and sends the encrypted result to P_S as the basis for the variance computation.

Steps 6B and 6C are OT steps that are part of the selection of the bottom quartile and top quartile values of the sorted list of inputs. In step 6B, the selection criterion of the OT protocol $rank_{\phi(i)} = \lceil \frac{n}{4} \rceil$ is the index of the sorted list's bottom quartile element. For step 6C, the top quartile index $\lfloor \frac{3 \cdot n}{4} + 1 \rfloor$ is used. Steps 12B and 12C rerandomise the ciphertext received from the service provider in steps 6B and 6C.

c) Round 3 (steps 14-30C): The service provider computes the encrypted, blinded statistical measures variance, median, best-in-class, maximum, bottom quartile, and top quartile by multiplying the values received in round 2. He sends them to the players together with the hashed MAC tags of the blinded sum. The latter is then used by the players to validate whether each player previously received the same blinded sum. Similar to round 2, each player then decrypts the blinded statistical measures and sends them to P_S together

TABLE I: Enhanced Benchmarking Protocol with Step Labels and Computations

Step	Computation
1	$\underline{P_i \rightarrow P_S}$: $E(x_i)$
2	$\underline{P_S \rightarrow P_i}$: $E(sum + r_1) = E(\sum_{i=1}^{n} x_i) \cdot E(r_1)$
3	$E(c_{\vec{\phi(i)}}) = (\ldots, E(c_{\phi(i)_{\phi'(j)}})$ $= E(r_{2_j} \cdot (x_{\phi(i)} - x_{\phi'(j)}) + r_{3_j}), \ldots)$
4	$\mathrm{OT}_{=}(med, r_4, \lceil \frac{n}{2} \rceil, i)$
5	$\mathrm{OT}_{\geq}(bic, r_5, \lfloor \frac{3 \cdot n}{4} + 1 \rfloor, i)$
6	$\mathrm{OT}_{=}(max, r_6, n, i)$
6B	$\mathrm{OT}_{=}(bq, r_{6B}, \lceil \frac{n}{4} \rceil, i)$
6C	$\mathrm{OT}_{=}(tq, r_{6C}, \lfloor \frac{3 \cdot n}{4} + 1 \rfloor, i)$
7	$\underline{P_i \rightarrow P_S}$: $sum + r_1 = D(E(sum + r_1))$
8	$MAC(sum + r_1 \| i, K_{MAC})$
9	$\underline{P_S \rightarrow P_i}$: $sum = sum + r_1 - r_1$
10	$\underline{P_i \rightarrow P_S}$: $E_i^{med\,\prime} = E_i^{med} \cdot E(0)$
11	$E_i^{bic\,\prime} = E_i^{bic} \cdot E(0)$
12	$E_i^{max\,\prime} = E_i^{max} \cdot E(0)$
12B	$E_i^{bq\,\prime} = E_i^{bq} \cdot E(0)$
12C	$E_i^{tq\,\prime} = E_i^{tq} \cdot E(0)$
13	$E((x_i - mean)^2) = E((x_i - \frac{sum}{n})^2)$
14	$\underline{P_S \rightarrow P_i}$: $E(var + r_7)$ $= E(\sum_{i=1}^{n}(x_i - mean)^2) \cdot E(r_7)$ $= (\prod_{i=1}^{n} E((x_i - mean)^2)) \cdot E(r_7)$
15	$E(med + r_8) = (\prod_{i=1}^{n} E_i^{med\,\prime} \cdot E(-r_{4_i})) \cdot E(r_8)$
16	$E(bic + r_9) = (\prod_{i=1}^{n} E_i^{bic\,\prime} \cdot E(-r_{5_i})) \cdot E(r_9)$
17	$E(max + r_{10}) = (\prod_{i=1}^{n} E_i^{max\,\prime} \cdot E(-r_{6_i})) \cdot E(r_{10})$
17B	$E(bq + r_{10B}) = (\prod_{i=1}^{n} E_i^{bq\,\prime} \cdot E(-r_{6B_i})) \cdot E(r_{10B})$
17C	$E(tq + r_{10C}) = (\prod_{i=1}^{n} E_i^{tq\,\prime} \cdot E(-r_{6C_i})) \cdot E(r_{10C})$
18	$h(MAC(sum + r_1 \| 1, K_{MAC}), \ldots,$ $MAC(sum + r_1 \| n, K_{MAC}))$
19	$\underline{P_i \rightarrow P_S}$: $var + r_7 = D(E(var + r_7))$
20	$MAC(var + r_7 \| i, K_{MAC})$
21	$med + r_8 = D(E(med + r_8))$
22	$MAC(med + r_8 \| i, K_{MAC})$
23	$bic + r_9 = D(E(bic + r_9))$
24	$MAC(bic + r_9 \| i, K_{MAC})$
25	$max + r_{10} = D(E(max + r_{10}))$
25B	$bq + r_{10B} = D(E(bq + r_{10B}))$
25C	$tq + r_{10C} = D(E(tq + r_{10C}))$
26	$MAC(max + r_{10} \| i, K_{MAC})$
26B	$MAC(bq + r_{10B} \| i, K_{MAC})$
26C	$MAC(tq + r_{10C} \| i, K_{MAC})$
27	$\underline{P_S \rightarrow P_i}$: $var = var + r_7 - r_7$
28	$med = med + r_8 - r_8$
29	$bic = bic + r_9 - r_9$
30	$max = max + r_{10} - r_{10}$
30B	$bq = bq + r_{10B} - r_{10B}$
30C	$tq = tq + r_{10C} - r_{10C}$
31	$\underline{P_S \rightarrow P_i}$: $h(MAC(var + r_7 \| 1, K_{MAC}), \ldots,$ $MAC(var + r_7 \| n, K_{MAC}))$
32	$h(MAC(med + r_8 \| 1, K_{MAC}), \ldots,$ $MAC(med + r_8 \| n, K_{MAC}))$
33	$h(MAC(bic + r_9 \| 1, K_{MAC}), \ldots,$ $MAC(bic + r_9 \| n, K_{MAC}))$
34	$h(MAC(max + r_{10} \| 1, K_{MAC}), \ldots,$ $MAC(max + r_{10} \| n, K_{MAC}))$
34B	$h(MAC(bq + r_{10B} \| 1, K_{MAC}), \ldots,$ $MAC(bq + r_{10B} \| n, K_{MAC}))$
34C	$h(MAC(tq + r_{10C} \| 1, K_{MAC}), \ldots,$ $MAC(tq + r_{10C} \| n, K_{MAC}))$

with the respective MAC tags. In the last steps of round 3, P_S sends the unblinded statistical measures to the players.

Steps 17B and 17C return the encrypted, rank-based statistical measures to the players. They are computed by multiplying the n encrypted, rerandomised OT messages that P_S received in steps 12B and 12C. Additionally, the statistical measures are blinded by multiplying the product by an encrypted random value r_{10B} and r_{10C}, respectively. In steps 25B and 25C, each player decrypts the blinded statistical measures he received in steps 17B and 17C and sends the result to P_S. In steps 26B and 26C, each player sends the MAC tags of the blinded bottom quartile and top quartile, respectively. The blinded statistical measures are concatenated with player P_i's index i before computing the corresponding MAC tag. This MAC computation requires the symmetric MAC key K_{MAC}. Steps 30B and 30C are used by P_S to send the output, i.e. the decrypted, unblinded statistical measures, to the players.

d) Round 4 (steps 31-34C): The service provider sends to each player the hashed MAC tags of the blinded statistical measures variance, median, best-in-class, maximum, bottom quartile, and top quartile. These are used by the players for validation of output integrity.

Steps 34B and 34C are used by the service provider to distribute the hashed MAC tags of the statistical measures. Step 34B is the hash of the n bottom quartile MAC tags that the players sent to P_S in step 26B. Step 34C in turn is the hash of the n top quartile MAC tags of step 26C.

B. Implementation

Our prototype consists of two main parts: the secure benchmarking client and the secure benchmarking service. Both are written in Java and have their own PostgreSQL database to enable persistent data storage. The service is a Java servlet, running on Cloud Foundry (see [7]), while the client is a Java console application. Additionally, we developed a C# front end add-in for the product costing suite. It utilises the client implementation to execute the benchmarking protocol for actual product costing key figures. Client and service communicate via HTTPS sending JSON strings. An overview of this secure benchmarking system is depicted in Fig. 1.

V. Evaluation

This Section provides an evaluation of the adapted approach implemented in the prototype. The evaluation refers to the requirements that are described in Section I.

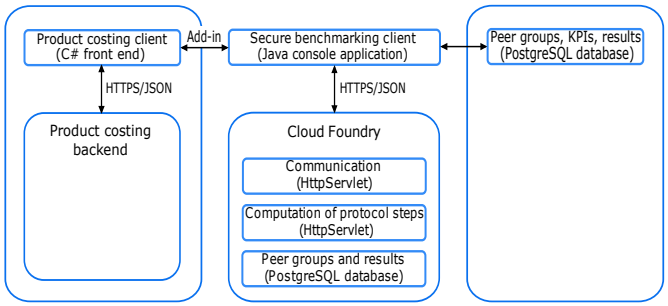


Fig. 1: Block Diagram of the Prototype's Architecture

A. Statistical measures

The selected protocol of [16] computes the statistical measures mean, variance, median, maximum, and best-in-class while the adapted protocol additionally computes the bottom quartile and top quartile. The correctness proof of our protocol is provided in the long version of this paper [4]. Therefore, the adapted protocol fulfils the requirements for necessary and optional statistical measures in the context of product costing.

B. Communication

As described in Section III-D, the protocol of [16] is based on the service provider model. The same applies to the adapted protocol since our adaptations presented in Section IV-A only add further computation steps but do not affect the communication. The service provider's implementation of the protocol runs as a Cloud Foundry service (see Section IV-B) proving the protocol's cloud suitability. No pairwise communication between players takes place. Messages sent from P_i to P_S and vice versa do not contain any private data of some player $P_{j\neq i}$ that allows for an identification of P_j. This enables anonymity among the players in case of a non-colluding P_S.

C. Performance

As stated in Section III-F, the quadratic complexity of the selected approach may prove critical for large n. Therefore, we conducted an extensive evaluation of the prototype's performance. We examined the computational and communication complexity of our protocol both theoretically and empirically.

1) Theoretical performance of the adapted protocol: The most expensive part of the original benchmarking protocol is the rank computation in step 3. Its computational and communication complexity is quadratic in the number of players n, i.e. $\mathcal{O}(n^2)$ [16]. This also holds for the rank computation of the adapted protocol since step 3 is not affected by the adaptations. Since the newly added steps each have complexity that is below $\mathcal{O}(n^2)$, the total computational and communication complexity of the protocol is $\mathcal{O}(n^2)$. The computational and communication complexity of each step of our protocol is given in the long version of this paper [4].

2) Practical performance of the prototype: To compare the prototype's performance to the theoretical complexity of the adapted protocol, a practical evaluation is performed in this Section. For this evaluation, a number of benchmarks were executed while the net execution time was measured. The net execution time is the time required for one entire protocol execution minus the time during which an inactive participant delays the execution, causing the others to wait. These benchmarks were conducted in different scenarios to determine the influence of the parameters peer group size n, network latency, bandwidth, and asymmetric (Paillier) key length. Furthermore, two worst case scenarios were considered combining several of these parameters.

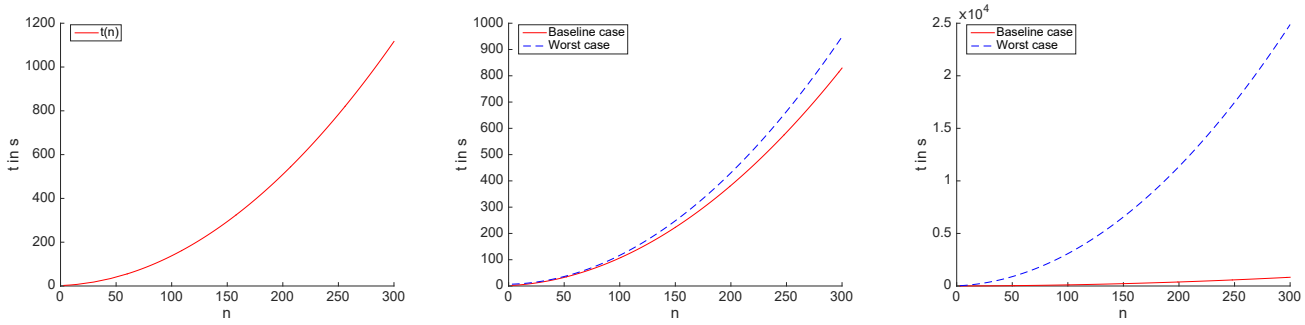
For the empirical analysis, the Java servlet was uploaded to a Cloud Foundry trial instance. The n clients were run in the form of the Java console application on three local client notebooks. Preliminary analyses indicated hardware limitations on the part of the service provider and the client machines. Hence, the performance evaluation was conducted for peer groups of $n \leq 60$ players and extrapolated for larger n. These n clients were equally distributed among the three PCs. In each of the corresponding tests, the net execution time t was measured for a default Paillier key length of 768 bits, a default bandwidth of circa 17 Mbit/s, and a network latency to the service provider of about 6 milliseconds.

a) Peer group size: Fig. 2a depicts the extrapolated net execution time for peer group sizes of $n \leq 300$ players given the measured net execution time for $n \leq 60$ players. With circa 1.200 seconds, the net execution time was below the required maximum of 24 hours. For the remaining parameters network latency, bandwidth, and key length, the default values were used. This scenario serves as the baseline scenario for the remaining analyses of the performance evaluation.

b) Network latency: To examine the effect of the network latency, the net execution time was measured for the default network latency of 6 milliseconds as well as for this default value plus an offset. The network latency offset had to be suitable for simulating global communication. Based on the network latencies for different continents as well as for intercontinental communication given in [22], an average offset of 150 milliseconds was chosen. Compared to the baseline, the offset only significantly affects the net execution time for smaller peer groups of $n \leq 40$ players. With increasing peer group size n, the effect of the network latency decreases.

c) Bandwidth: Customers can reasonably be assumed to have access to a broadband network, which is defined as a network with a transmission capacity of at least 2 Mbit/s [21]. Even for such a low bandwidth, the difference between the net execution times was small. Increasing the bandwidth by a factor of 8.5 barely affected the net execution time.

d) Key length: For investigating the effect of the key length on the net execution time, the key lengths 768, 1024, 1536, and 2048 bits were taken into consideration. These investigations showed two facts. On the one hand, the net execution time increased disproportionately with growing key length. This is due to the complexity of the modular exponentiation of Paillier's cryptosystem, which is cubic in the key length [16]. On the other hand, for keys of 2048 bits, the net execution time for $n = 300$ was about 25.000 seconds, i.e. approximately 7 hours.



(a) Different Peer Group Sizes, Extrapolated (b) Worst Case 1, Extrapolated (c) Worst Case 2, Extrapolated

Fig. 2: Net Execution Times of the Enhanced Benchmarking Protocol

e) Worst case: For the worst case analysis, two scenarios are taken into consideration. This analysis refers to the worst case from a net execution time perspective, not from a security perspective. The first scenario investigates the combined effect of the peer group size, network latency, and bandwidth while the second worst case scenario also takes the key length into consideration. The extrapolated net execution time for the first worst case scenario is depicted in Fig. 2b. For growing peer group size, the fitted curves diverge. The net execution time for $n = 300$ was below the required maximum of 24 hours. The fitted curves for both cases grow faster than they diverge. Hence, the influence of the computations on the net execution time can be assumed to be higher than the impact of the communication. The extrapolated net execution time of the second worst case scenario is depicted in Fig. 2c. Due to the large difference in the key length, which was 768 bits in the baseline case and 2048 bits in the worst case, the fitted curves grow very differently. However, both grow quadratically. The net execution time of the second scenario was circa 7 hours. Hence, the prototype's performance meets the requirements even for larger keys of 2048 bits, poor internet connection, and peer groups of $n \leq 300$ players. However, with its quadratic complexity, the net execution time of this system would soon exceed the desired maximum of 24 hours for larger peer groups. This is mostly due to the extensive rank computation.

D. Security

We proved our protocol to be secure against computationally bounded semi-honest adversaries (see the long version of this paper [4]). This privacy proof shows that up to $n-1$ players can be corrupted without them learning anything about the non-corrupted players' inputs that cannot be inferred from the corrupted players' inputs and the output. If the service provider himself is corrupted, privacy of the inputs ensures that he learns nothing about the players secret values as long as none of the players is corrupted at the same time [16]. The possibility of a player eavesdropping on the communication between another player and the service provider is precluded as they communicate over pairwise secure channels. Integrity of the output is enabled via cryptographic hashes and MACs. Further security related properties, such as secure data storage, access control, availability of the service, and robustness regarding players dropping out of a protocol execution, are beyond the scope of this paper.

E. Summary of the evaluation

The statistical measures provided by the implemented protocol are the mean, variance, median, maximum, best-in-class, bottom quartile, and the top quartile. The prototype implements the service provider model, enabling anonymity among the players as well as cloud suitability. Our enhanced secure benchmarking protocol has been proven to ensure confidentiality of the players' secret inputs against computationally bounded semi-honest adversaries. Output integrity is also provided. The computational and communication complexity both are $\mathcal{O}(n^2)$. The net execution time for 300 players is circa 7 hours. Consequently, the prototype implementing the adapted protocol meets each of the requirements defined in Section I. It even provides additional statistical measures. However, our extensive evaluation showed the bottleneck of our protocol, which is the quadratic rank computation.

VI. Discussion

A variety of attacks that are possible for TTP-based benchmarking analyses can also be applied to our system. This includes inference attacks, where an adversary-controlled player P_i repeatedly runs benchmarks for the same KPI and peer group, each time with a different input x_i, in order to gain additional knowledge of other players' secret inputs. Such attacks can be precluded via organisational measures like retention periods. In the described product cost optimisation scenario, benchmarks are not supposed to be executed on demand but rather once every one or two quarters on predefined dates. Given the long period of time between two benchmarks and assuming that at least some players' KPIs changed during that period, the value of such an attack would be rather limited.

Attacks where a player inputs an incorrect KPI to temper with the results are much harder to circumvent as they require a mechanism for analysing the semantics of the inputs. However, as this is immanent in benchmarking in general, such behaviour was beyond the scope of our considerations.

VII. Conclusion and Future Work

This paper provides an overview of benchmarking based on secure multi-party computation. It elaborates requirements for a secure benchmarking system in the context of product cost optimisation. Based on these requirements, generic approaches to secure KPI benchmarking were discussed, the existing related work was reviewed, and fitting approaches were described. The most suitable approach was selected, extended, implemented in a prototype, and extensively evaluated. The resulting protocol meets all of the requirements for a privacy-preserving benchmarking system in the given context of product cost optimisation. However, its computational and communication complexity that is quadratic in the number of players n, i.e. $\mathcal{O}(n^2)$, shows potential for further optimisation.

In our future work, we will focus on reducing the computational complexity by simplifying the rank computation. In the presented protocol, this sorting step compares each player's value to any other player's value, causing a quadratic number of comparisons. Fortunately, the lower bound for comparisons in sorting is $\mathcal{O}(n \cdot \log n)$ [8]. Such a sorting mechanism with less comparisons would likely need to be more interactive, causing a higher communication complexity. Furthermore, it would require an additional step for hiding the order of the inputs. Otherwise, the service provider would learn the players' ranks in the sorted list of inputs.

References

[1] Michael Armbrust et al. "A View of Cloud Computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[2] Mikhail Atallah et al. "Private collaborative forecasting and benchmarking". In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. WPES '04. ACM, 2004, pp. 103–114.

[3] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The Round Complexity of Secure Protocols". In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*. STOC '90. ACM, 1990, pp. 503–513.

[4] Kilian Becher, Martin Beck, and Thorsten Strufe. "An Enhanced Approach to Cloud-based Privacy-preserving Benchmarking (Long Version)". In: *ArXiv e-prints* (2018). arXiv: 1810.04971 [cs.CR].

[5] Keith Bonawitz et al. "Practical Secure Aggregation for Federated Learning on User-Held Data". In: *NIPS Workshop on Private Multi-Party Machine Learning*. PMPML '16. 2016.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption Without Bootstrapping". In: *ACM Transactions on Computation Theory* 6.3 (2014), pp. 1–36.

[7] Cloud Foundry Foundation. *Cloud Foundry Foundation's website*. Cloud Foundry Foundation. 2017. URL: https://www.cloudfoundry.org (visited on 02/01/2019).

[8] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd ed. Cambridge, MA, USA: MIT Press, 2009. ISBN: 978-0-262-03384-8.

[9] Ronald Cramer, Ivan Damgård, and Jesper Nielsen. *Secure Multiparty Computation and Secret Sharing*. 1st ed. Vol. 1. New York, NY, USA: MIT Press, 2015. ISBN: 978-1-107-04305-3.

[10] Craig Gentry. "A fully homomorphic encryption scheme". PhD thesis. Stanford University, 2009.

[11] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play ANY Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. STOC '87. ACM, 1987, pp. 218–229.

[12] Martin Hirt. "Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting". PhD thesis. ETH Zürich, 2001.

[13] Yan Huang, David Evans, and Jonathan Katz. "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?" In: *19th Annual Network & Distributed System Security Symposium*. NDSS '12. The Internet Society, 2012.

[14] Yan Huang et al. "Faster Secure Two-party Computation Using Garbled Circuits". In: *Proceedings of the 20th USENIX Conference on Security*. SEC '11. USENIX Association, 2011, pp. 35–35.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Chapman and Hall/CRC Cryptography and Network Security Series. Boca Raton, FL, USA: Chapman & Hall/CRC, 2008. ISBN: 978-1584885511.

[16] Florian Kerschbaum. "A privacy-preserving benchmarking platform". PhD thesis. Karlsruhe Institute of Technology, 2010.

[17] Florian Kerschbaum. "Practical Privacy-Preserving Benchmarking". In: *Proceedings of The IFIP TC-11 23rd International Information Security Conference*. IFIP SEC '08. Springer, 2008, pp. 17–31.

[18] Vladimir Kolesnikov and Thomas Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications". In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*. ICALP '08. Springer, 2008, pp. 486–498.

[19] Benny Pinkas et al. "Secure Two-Party Computation Is Practical". In: *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*. ASIACRYPT '09. Springer, 2009, pp. 250–267.

[20] Adi Shamir. "How to Share a Secret". In: *Commununications of the ACM* 22.11 (1979), pp. 612–613.

[21] International Telecommunication Union. *The Birth of Broadband. Frequently Asked Questions*. International Telecommunication Union. 2011. URL: https://www.itu.int/osg/spu/publications/birthofbroadband/faq.html (visited on 02/01/2019).

[22] Verizon Communications. *IP Latency Statistics*. Verizon Communications. 2017. URL: http://www.verizonenterprise.com/about/network/latency/ (visited on 09/20/2017).