New Systems and Algorithms for Preserving Big-data Privacy in Clouds

Abstract:
Driven by the rapidly increasing amount of data, many application vendors (e.g., Uber) store data on clouds. Meanwhile, application vendors and third parties often write self-defined queries (e.g., map/reduce) to process the data. This has caused severe privacy problems. One problem is ``computation leakage'': third-parties can easily acquire sensitive fields in data records (e.g., a credit card in an Uber transaction) using the self-defined queries. Existing techniques (e.g., differential privacy) add random noises to hide individual sensitive data, but due to the lack of precisely tracking how sensitive data flows to a query result, these techniques often add excessive noise and make query results useless. Another problem is ``cloud privacy'': cloud providers can compromise on external attacks and steal the data being queried.

This proposal adopts a holistic methodology to tackle both the two problems with three objectives. First, to prevent ``computation leakage'', we will build Kakute, the first Data Flow Tracking (DFT) system for big-data queries. Kakute provides easy-to-use APIs for application vendors to tag sensitive fields in data records, and it automatically tracks and prevents these fields propagating to query results. An open challenge in existing DFT systems is that propagating tags in data-intensive queries is too slow. For instance, a notable DFT system incurred a 128X slowdown compared to native, insecure queries in our study. By leveraging subtle efficiency natures of big-data queries, we will create two fast tag propagation techniques called Reference Propagation and Tag Sharing. Our Kakute preliminary prototype presented in [ACSAC '17] shows that it incurs merely 32.3% overhead compared to native queries.

Second, we will develop Fine-grained Differential Privacy (FDP), a novel differential privacy technique and its new algorithms. Compared to DFT, differential privacy has complementary strength because it enables the aggregation of sensitive data while hiding individual privacy. However, existing differential techniques are too coarse-grained: they often add excessive noise to all fields of all data records and make results useless due to the lack of precisely tracking data flows. By leveraging Kakute, our new FDP technique will only add noise to sensitive fields, preserving strong differential privacy for sensitive data and good usability for most results.

Third, to tackle the ``cloud privacy'' problem, we will leverage the Intel SGX hardware to build the first just-in-time, privacy-preserving Java compiler for unmodified big-data frameworks (e.g., Spark). Existing SGX-based systems for big-data frameworks have two major challenges: they need to rewrite the Java big-data queries into SGX-compatible C++, or their trusted computing base is too large (e.g., the entire JVM). Our new compiler will run only the self-defined Java queries in SGX with a thin, verified just-in-time translator, and we will create fast SGX runtime techniques to achieve reasonable overhead compared to native queries.

The success of this proposal will effectively preserve big-data privacy in clouds, potentially benefiting every computer user, software vendor, organization, and government.

Long term impact:

The big-data and cloud computing trends bring fascinating opportunities to all entities, including data providers (e.g., application vendors such as Uber), cloud providers (e.g., Amazon), and computation providers (e.g., application vendors and their third-party contractors). Unfortunately, despite decades of effort, data leakage remains one of the most severe threats in clouds. In a data provider's perspective, both computation providers (e.g., the 2017 iCloud account leakage caused by third-parties) and cloud providers (e.g., the 2013 Yahoo Cloud compromise and the 2014 J.P. Morgan account leakage) have caused severe breaches on data privacy and huge financial loss.

Even in a private cloud (i.e., the data provider is the cloud provider), sensitive data such as credit cards, user identities, and healthcare records can easily be leaked in the self-defined queries written by computation providers (e.g., the 2017 iCloud leakage). Real-world breaches include directly acquiring particular user's identities from the query results and sending sensitive data outside the cloud through IO functions in the queries.

In the short term, we plan to accomplish both the Objective 1 and Objective 2 of this proposal, which can effectively tackle the ``computation leakage'' problem in private clouds. Objective 1 proposes Kakute, the first Data Flow Tracking (DFT) system for big data queries. Although numerous DFT systems have been built to prevent accessing sensitive data in mobile phones and server programs, no DFT system exists for big-data frameworks. A key reason is that DFT's tag propagation incurs prohibitive overhead on data-intensive queries (e.g., we found a notable DFT system running with the WordCount query incurred a 128X slowdown compared to its native, insecure query).

Our key insight to address the DFT efficiency challenge is that most fields of a record often have the same tags. Leveraging this insight, we present two new techniques, Reference Propagation and Tag Sharing. To achieve a robust DFT architecture for distributed big-data frameworks (e.g., Spark), Kakute completely captures the frameworks' inter-computer data flows (i.e., shuffles). We have implemented a Kakute prototype and integrated it with Spark. Kakute carries built-in checkers for four security and reliability problems: sensitive data leakage, data provenance, programming bugs, and performance bugs. Kakute not only incurs a moderate performance overhead of 32.3% compared to native queries, but it also effectively detects 13 real-world security and performance bugs. These promising preliminary results have been presented in [ACSAC '17] and [TPDS '17].

In the security community, DFT and differential privacy are complementary: DFT enforces mandatory access control on sensitive data, but it may cause some critical query results to be missing; differential privacy allows the aggregation results of sensitive fields while hiding individual privacy, but due to the lack of precisely tracking data flow, it may suffer from excessively added noise and inaccurate (useless) query results. Therefore, the Objective 2 of this proposal takes the first step to integrate DFT and differential privacy and to develop a novel Fine-grained Differential Privacy (FDP) technique, reaching the best of the both worlds.

In the intermediate term, we will tackle the ``cloud privacy'' problem in a public cloud (e.g., Amazon) by accomplishing Objective 3. Recently, Intel SGX has attracted high attention on protecting the privacy of data being queried, because it enforces hardware protection on the confidentiality of data even if the cloud itself is malicious. Meanwhile, SGX is good fit for big-data queries because these queries do data-intensive computations in userspace and hardly invoke system calls. Existing SGX-based systems for big-data have two major challenges: they need to rewrite the Java big-data queries into SGX-compatible C++, or their trusted computing base is too large. To tackle these two challenges, Objective 3 will build the first just-in-time, privacy-preserving Java compiler for unmodified big-data frameworks. Our new compiler will run only the self-defined Java queries in SGX with a thin, verified just-in-time translator, and the rest of JVM is outside SGX without affecting the privacy of the data being queried. Therefore, our compiler will be the first work to support fast, unmodified Java big-data queries with minimal trusted computing base.

In the long term, by integrating the outcomes of all the three objectives in this proposal and extensively applying them on real-world big-data frameworks, we will help data providers enforce comprehensive privacy against both computation providers and cloud providers. This will benefits almost all individuals, software vendors, and organizations. For instance, many HK finance entrepreneurs demand strong privacy for their data deployed in clouds. Moreover, we envision that the outcomes of this proposal will broadly promote other security techniques, including strengthening the integrity and availability of real-world software.

Objectives:

1. [To build the first Data Flow Tracking (DFT) system for private clouds]
We will create Kakute, a fast DFT system that can track and prevent sensitive data leakage in self-defined big-data queries. We will make Kakute support diverse big-data queries on large, popular datasets, and we aim will design Kakute to incur reasonable performance overhead compared to the native, insecure queries.

2. [To create a Fine-grained Differential Privacy (FDP) technique for private clouds]
We will leverage Kakute to develop FDP and its new algorithm, which will only add noise to sensitive data fields, preserving strong differential privacy for sensitive data and good accuracy for most query results. We will extensively study FDP's accuracy improvements on both sensitive and insensitive data compared to existing differential privacy techniques.

3. [To construct the first compiler for big-data privacy in public clouds]
Our compiler will support unmodified big-data frameworks by creating a thin translator to automatically convert Java bytecode to SGX-compatible code. We will verify the translator with state-of-the-art verification techniques so that our compiler will have a minimum trusted computing base (i.e., only SGX). We will evaluate whether our compiler can protect the privacy of data against real-world privileged attacks. We will quantify the compiler's performance overhead compared to the native queries.

1 Research Background

This proposal involves three main entities: data providers (e.g., Uber or individual computer users), computation providers, and cloud providers (e.g., Amazon EC2). This section presents three most relevant techniques (§1.1, §1.2, and §1.3), and then introduces motivation (§1.4) and related work (§1.5 and §1.6).

1.1 Big-data computing frameworks

Big-data frameworks (e.g., Spark [76] and MapReduce [15]) are popular for computations on tremendous amounts of data. These frameworks provide self-defined Java functions (e.g., MAP/REDUCE) to let computation providers write their algorithms, and they and automatically apply these functions on the data stored across computers in parallel.

To avoid excessive computation, big-data frameworks adopt a lazy transformation approach [50, 75, 76]. Spark often uses lazy transformations (e.g., MAP), and calls to these transformations only create a new data structure called RDD with *lineage* (the sequence of transformations for a data record). The actual transformations are only triggered when collecting operations (e.g., COLLECT, COUNT) are called. These collecting operations trigger transformations along lineages, so unnecessary computations are avoided. **Objective 1** will leverage lazy transformation to create a fast DFT technique called Reference Propagation (§2.1).

1.2 Software-based privacy techniques

Data Flow Tracking (DFT) is a mandatory access control technique for preventing sensitive information leakage [48]. DFT attaches a tag to a variable (or object), and this tag will propagate during computations on the variable at runtime. DFT has been applied to various areas, such as preventing sensitive information (e.g.GPS data and contacts) leakage in cellphone [23, 66], web services [53], and server programs [38]. To the best of our knowledge, no DFT system exists for big-data computing.

Complimentary to DFT, statistical techniques, including K-anonymization methods [41, 65] and differential privacy [42, 46, 56], allow the aggregation of sensitive data while adding random noise to preserve individual privacy. However, these statistical techniques are either not secure (K-anonymization) or suffering from great losses of accuracy (differential privacy). A recent work [30] reports more than 30% losses of accuracy. For a query results, low accuracy means bad utility: a simple KMeans program will return centroids far from the accurate ones, and the accuracy loss rate is much larger than the training error rate which is several percents in practice.

A key reason for this bad utility problem is that differential privacy can not track how sensitive data fields flow to query results, so they have to take a coarse-grained approach, which conservatively adds noise to all fields and records. **Objective 2** (§2.2) proposes a novel fine-grained differential privacy technique, which combines the strengths of DFT and differential privacy.

1.3 Hardware-based privacy techniques

Trusted Execution Environment (TEE) is a promising technique for protecting computation in a public cloud even if the cloud's operating systems or hypervisors are compromised. For instance, Intel-SGX [31], a popular commercial TEE product, runs a program in a hardware-protected *enclave*, so code and data are protected from outside. Compared with the approach of computing on encrypted data (§1.5), TEE is much safer and 100X to 1000X faster. For instance, a SGX-based system Opaque [78] incurs a moderate performance overhead of 30% compared to native big-data queries.

However, to practically run Java big-data queries with SGX, two open challenges remain. First, existing SGX-based systems [78] require computation providers to manually rewrite the readily pervasive Java queries into SGX-compatible C++, a time-consuming and error-prone process. Second, existing SGX-based systems for big-data have too large Trusted Computing Base (TCB). Existing systems (e.g., SGX-BigMatrix [58]) run a whole language interpreter (e.g., JVM and Python runtime) in enclaves, causing a too large (and too dangerous) TCB: JVM code comes from many different parties/vendors and extremely hard to verified. **Objective 3** (§2.3) tackles these two open challenges by building a new just-in-time compiler.

1.4 Motivation of objectives

Data leakage (or breach), defined as the leakage of sensitive customer or organization data to unauthorized users [37], is a top security threat [3, 34] in cloud computing. In a data provider's perspective, both computation providers (e.g., the 2017 iCloud account leakage caused by third-parties [73]) and cloud providers (e.g., the 2013 Yahoo Cloud compromise [67]) have caused severe data leakage and huge financial loss. This proposal aims to preserve the data provider's privacy by going two directions. First, we will propose two novel complimentary techniques in **Objective 1** (KAKUTE) and **Objective 2** (fine-grained differential privacy) to protect privacy against the computation providers in private clouds. Second, we will propose **Objective 3** (a new privacy-preserving compiler) to protect privacy against the (public) cloud providers. By integrating the outcomes from all three objectives, data privacy will be effectively preserved.

1.5 Related work by others

Computing on encrypted data. Homomorphic encryption [22, 27, 51] is a technique for computating on encrypted data in untrusted environments. Homomorphic encryption contain two kinds: Fully homomorphic encryption (FHE) and partial homomorphic encryption. Partial homomorphic encryption (e.g. Additive Homomorphic Encryption [51]) incurs a much lower overhead compared with FHE. A evaluation [26] on FHE shows a $10e9$ slowdown, which is acceptable in practice. Systems that adopts PHE (e.g. Monomi [69], Crypsis [64], CryptDB [55], MrCrypt [68]) reports a much better overhead, but it has limited expressiveness (e.g., SQL operators) and requires extra trusted servers. Seabed [52] proposes asymmetric encryption schemes and reduces the performance overhead of AHE, but its expressiveness is still quite limited.

SGX-based systems. Intel SGX is a promising technique to provide privacy-preserving analytic in public clouds. Compared with software-based solutions, hardware-based solutions incurs much lower overhead. TrustedDB [6] is a hardware-based secure database. VC3 [57] proposes a secure distributed analytic platform with read-write validations on MapReduce [15]. Opaque [78] supports secure and oblivious SQL operators on SparkSQL [5]. However, all these systems have limited expressiveness (e.g.SQL operators), and VC3 even needs to rewrite the program with C++. A recent work [49] proposes a oblivious machine leaning framework on trusted processors. BigMatrix [58] proposes an oblivious and secure vectorization abstraction on python, but it has limited expressiveness and it needs to rewrite the original program with this new abstraction. Although BigMatrix provides guideline for writing a oblivious program, but it would be a time-consuming and error-prone process.

Big-data privacy systems. Big-data privacy has been a top emerging threat [4, 37] as more and more user sensitive data is stored and processed in clouds. Airavat[56], PINQ [42] and GUPT [46] propose to apply differential privacy [21, 44] in MapReduce, to prevent leakage from user query, but differential privacy can result in incorrect results. Sedic [77] proposes to offload sensitive computations to private clouds. MrLazy [2] proposes a framework of combining data provenance and static DFT analysis for self-defined queries, to provide fine-grained information flow for security. However, static DFT is not precise and may suffer from false positive. KAKUTE provides fine-grained information control of sensitive data, with no need to modify the original program.

1.6 Related work by the PI and co-I

The PI is an expert on secure and reliable distributed systems [10–13, 24, 33, 71, 72, 74]. The PI's works are published in top conferences on systems (OSDI, SOSP, SOCC, TPDS, and ACSAC) and programming languages (PLDI and ASPLOS). Recently, the PI has collaborated with Huawei to launch a technology transfer project based on his dependable distributed system [72]. The co-I is an expert on high-performance computing [1, 8, 36, 45, 79], fault-tolerance [59, 60], and Java compilers [39, 61, 70]. The co-I's works are published in top systems conferences (Cluster, SC, and ICPADS) and journals (JPDC, TPDS, IEEE Tran. Computers). As preliminary results for this proposal, the PI has presented KAKUTE [33] in ACSAC '17, and the PI and co-I have collaborated to present CONFLUENCE [24] in TPDS '17.

2 Research Plan and Methodology

2.1 Objective 1: preventing big-data computation leakage with KAKUTE

This section presents major challenges in existing DFT systems (§2.3.2) and KAKUTE (§2.1.2).

2.1.1 Challenges: existing DFT systems are too slow and incomplete for big-data

Although DFT is a powerful access control technique, existing DFT systems incur high performance overhead, especially for data-intensive computations. For instance, we ran a recent DFT system Phosphor [7] in Spark with a WordCount algorithm on a small dataset of merely 200MB, and we observed 128X longer computation time compared with the native Spark execution [33]. The second challenge is completeness: big-data frameworks usually contain *shuffle* operations, which redistribute data and results across computers. However, most existing DFT systems ignore data flows across computers. For the few [53] who support cross-host data flows, transferring all tags in shuffles consumes excessive network bandwidth.

2.1.2 KAKUTE: a fast, precise DFT system for big-data

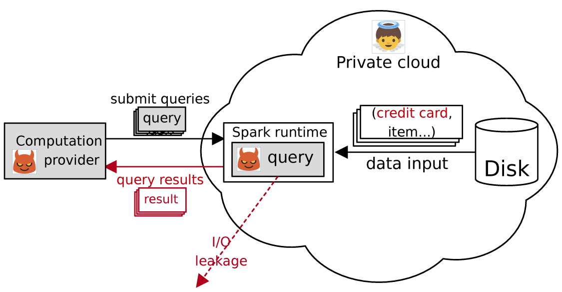

Figure 1: Threat model of KAKUTE. Red colors means sensitive data or leaking channels. Shaded (grey) components may leak data, and KAKUTE is designed to defend against them.

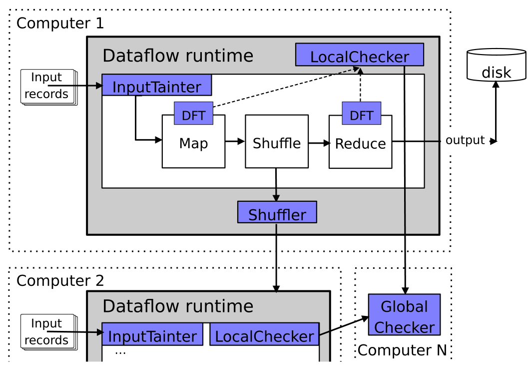

Figure 2: KAKUTE architecture. KAKUTE's key components are shaded (and in blue).

We present KAKUTE, the first precise and complete DFT system for big-data frameworks. Our key insight to address the DFT performance challenge is that multiple fields of a record often have the same tags with the same sensitivity level. For example, in an Taobao order record ⟨t', ǔserId, p̌roductID⟩, only the ǔserId field is sensitive, while the other fields are insensitive and they can share the same tag. Leveraging this insight, we present two new techniques, Reference Propagation and Tag Sharing. Reference Propagation avoids unnecessary tag combinations by only keeping the *lineage of tags* in the same self-defined queries, while Tag Sharing reduces memory usage by sharing tags among multiple fields in each record. To tackle the completeness challenge, KAKUTE completely captures data flows in shuffles by intercepting the , and it efficiently reduces the amount of transferred DFT tags using Tag Sharing. Both techniques are illustrated in Appendix (c) of this proposal.

Figure 1 defines KAKUTE's threat model. Figure 2 shows KAKUTE's design. The InputTainter component provides easy-to-use APIs for data providers to automatically tag sensitive fields. The DFT component is enabled in self-defined functions. The Local- and Global-Checker detect and prevent illegal flows of sensitive fields (e.g., credit cards flow to IO functions in self-defined functions). Shuffle operations across computers are intercepted and tags are added. Therefore, DFT is completely captured across computers.

We will implement KAKUTE and integrate it with Spark. We will leverage Phosphor [7], an efficient DFT system working in the Java byte-code level. KAKUTE instruments computations of a Spark worker process to capture data flows inside self-defined-functions. KAKUTE provides different granularities of tracking with two types of tags: INTEGER and OBJECT tags. INTEGER provides 32 distinct tags for identifying 32 sensitivity levels, suitable for detecting data leakage and performance bugs. OBJECT provides an arbitrary number of tags, which is suitable for data provenance and programming debugging.

Preliminary results. We have implemented a KAKUTE prototype and evaluated it on seven popular big-data algorithms, including three text processing algorithms WordCount [63], WordGrep [40] and TwitterHot [63], two graph algorithms TentativeClosure [63] and ConnectComponent [63], and two medical analysis programs MedicalSort [54] and MedicalGroup [54]. We evaluated these algorithms with real-world datasets that are comparable with related systems [9, 28, 32]. Our evaluation shows that: (1) KAKUTE incurred merely 32.3% overhead (Figure 3) with INTEGER tag, about two orders of magnitudes faster than a recent DFT system Phorspor [7]; and (2) KAKUTE effectively detected 13 real-world security and reliability bugs presented in other papers [14, 28, 56]. These promising preliminary results are published in ACSAC '17 and TPDS '17.

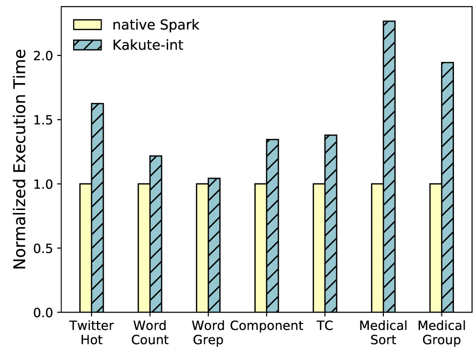

Figure 3: KAKUTE execution time normalized to native Spark executions. 100% means no overhead.

Future directions. We will extend KAKUTE in three directions. First, we will port KAKUTE onto more big-data frameworks, including PIG [50] and HADOOP [29]. Second, we will extend KAKUTE to detect broader types of real-world security bugs. Third, we will apply KAKUTE to augment other complementary privacy techniques, including anonymization techniques and differential privacy (**Objective 2**).

2.2 Objective 2: developing the Fine-grained Differential Privacy (FDP) technique

KAKUTE (**Objective 1**) strictly prevents sensitive data flowing to IO functions or query results, but in some scenarios it is still desirable to let computation providers acquire aggregation results (e.g., the sum of citizens who have got cancer in a country) on sensitive fields as long as individual information is not leaked. Differential privacy [16–18] can enforce statistical bounds on aggregation results and prevent individual information leakage, so it is complementary to DFT and has attracted much attention recently.

2.2.1 Challenge: existing differential privacy techniques are coarse-grained and thus inaccurate

Existing differential privacy techniques often suffer from low accuracy for query results. To prevent computation providers revealing individual data, differential privacy typically adds noise either on input data records or query results. However, due to the lack of precisely tracking how inputs are computed and propagated to outputs, to enforce statistical guarantee on outputs, differential privacy often conservatively add the same noise to all fields of a data record and to all records, causing inaccurate results. For instance, prior work [30] reports more than 30% loss of accuracy when the security guarantee is high (the probability of leakage is low). Therefore, a KMeans program will return centroids far from the accurate ones. This low accuracy makes results useless as it is much larger than the KMeans training error rate (a few percents).

2.2.2 FDP and its new algorithm

Our key insight is that DFT and differential privacy can complement each other, getting the best of both worlds. Considering each data record, DFT can precisely track how sensitive data fields flow to which query result, so differential privacy needs only add noise to the sensitive input fields or results. Considering all data records, DFT can also distinguish which records are more important, so we can add give important records more noise than the others.

This insight nurtures Fine-grained Differential Privacy (FDP). In FDP, each record belongs to a user who assigns a security tag to all its data records. A security tag that is related to the privacy budget ε (or accuracy level). When the privacy budget is high, the probability of leakages is high. We adopt the personalized differential privacy model in recent work [35]. $\Phi(x)$ return the ε of a particular record x. The threat model of FDP is the same as KAKUTE's (Figure 1), because FDP aims to defend against malicious computation providers in private clouds.

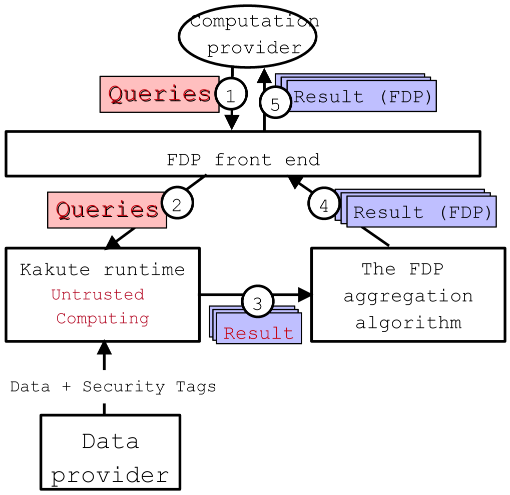

Figure 4: The workflow of FDP with five steps.

Algorithm 1: The FDP aggregation algorithm

Input: Dataset T, dataset size N, privacy budget ε_k for security level k, output range (min, max)

n = a suitable partition

for $i \leftarrow 1$ **to** n **do**

  $O_i \leftarrow f(T_i)$;

  if $O_i > \max$, $O_i \leftarrow \max$ if $O_i < \min$, $O_i \leftarrow \min$

for *dimension j of the output O* **do**

  k $\leftarrow getLevel(O_j)$;

  $O_j \leftarrow \frac{1}{n}\sum_{i=1}^{n} O_{ij} + Lap(\frac{max - min}{n\varepsilon_k})$

Output: O

Definition 2.1. *Differential Privacy For two neighbor dataset D and D' differing at record x, a mechanism $\mathscr{M}(y)$ is differentially private with the following condition:*

$$Pr[\mathscr{M}(D) \in O] \leq e^{\Phi(x)} \times Pr[\mathscr{M}(D') \in O] \tag{1}$$

Intuitively, Differential Privacy guarantees that the probability of producing different result with neighboring dataset is low. For the personalized model, the probability is different for records belonging to different users, so that different users have various levels of protections.

To start with, we need to determine the relation of the privacy budget and the security level (a `getLevel` function). We adopt a deterministic model, and there are 6 security levels: insecure, dp_1, dp_2, dp_3, dp_4 and non-released. Insecure records can be release directly, while non-released can not be used in any computation to the final result. dp_1 to dp_4 varies in terms of their privacy budgets for differential privacy.

Theorem 2.1. *Laplace Mechanism [16] Adding noise with Laplace distribution Laplace$(\frac{\Delta f}{\varepsilon})$ enforces Differential Privacy, and global sensitivity Δf is defined as*

$$\Delta f = max||f(D) - f(D')||_1 \tag{2}$$

To calibrate out to enforce differential privacy, a simple approach is to use the ε inferred by the highest security level of each dimension, then add noise to the output directly, but this approach is too naive for diversity of security levels. Instead, we adopt the sample-and-aggregate framework proposed in previous work [62]. In this model, data is partitioned into multiple parts (size of each part is m). Each partition may consists different level. The noise aggregator adds noise to final result of each partition according to the security level of each partition. Formally, data is partitioned into multiple parts denoted as p_1, p_2, ..., p_n. The security level and its corresponding ε are ε_1, ε_2, ..., ε_n.

Theorem 2.2. *For any output record in O, the estimator is $max(\varepsilon_k)$-differentially private.*

Proof. For each dimension, we can divide the output dataset as k parts according to their security levels. Then we apply the Laplace Mechanism to each subset of data using their privacy budget ε_k. According to the composition theorem [19, 43, 80], the whole output dataset is $max(\varepsilon_k)$-differentially private. □

The error of the final result incurs in this aggregator comes from two parts: the Laps noise error and the partition error. It is crucial to reduce the final error incurs by this aggregator while keeping the differential security guarantee. We can adopt a hill-climbing approach for optimization.

Future directions. We will fully develop this FDP technique by going along two directions. First, we will do an extensive study on real-world big-data queries and quantify the improvements on result accuracy. Second, currently we propose a end-to-end differentially private computation system. It adopts a personalized differential privacy model [35], which improves usability without reducing the security guarantee. As for future direction, we can consider a (ε, δ)-differentially [20] private model.

2.3 Objective 3: creating a privacy-preserving compiler for big-data queries in public clouds

More and more sensitive data are stored and processed in public clouds (e.g., Amazon EC2 and Dropbox), and recent real-world privacy breaches have shown that data are often leaked while being processed by public cloud providers. Trusted Execution Environment (TEE) is a promising technique to protect computation on public clouds even if the cloud's operating system is compromised. For example, Intel-SGX [31] runs programs in a enclave, so code and data are protected and can not be seen by the attackers. Meanwhile, SGX is good fit for big-data queries because these queries are data-intensive in userspace and they hardly invoke system calls (OS kernel can easily break SGX's security on memory). A latest big-data analytic system Opaque [78] reports only 30% overhead compared to native, insecure executions.

2.3.1 Challenges: existing SGX-based systems require rewriting queries and have too-large TCB

Despite recent advances (Opaque, VC3, Coco, and SGX-BigMatrix) on building SGX-based systems for big-data queries, two major challenges still remain. First, enclaves require completely rewriting the readily pervasive Java big-data queries into C++, an time-consuming and error-prone process. Second, to easy implementation, existing systems typically run the entire language runtime (e.g., JVM or Python runtime) within SGX, causing the Trusted Computing Base (TCB) to be too large and vulnerable (e.g., JVM has millions of lines of code from many companies and is vulnerable to insider attacks [3]).

2.3.2 MAAT: a just-in-time (JIT), privacy-preserving Java compiler for big-data queries

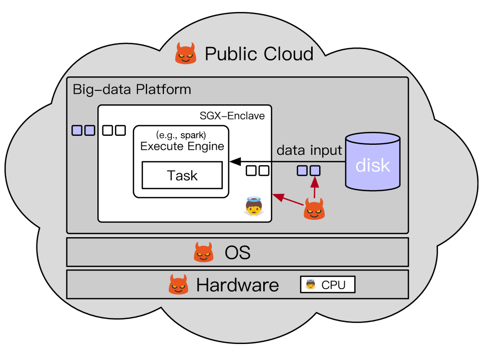

Figure 5: Threat model of MAAT. Data records with blue color are encrypted, and white color are plaintext. Shaded (grey) components may leak data, and MAAT is designed to defend against them.

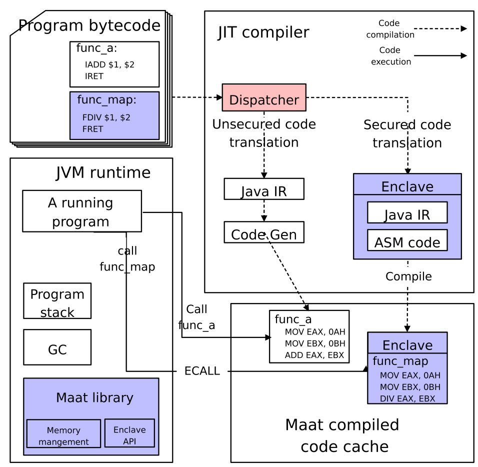

Figure 6: MAAT JIT compiler architecture. Key components are shaded (and in blue).

We propose MAAT, the first compiler that runs unmodified Java big-data queries in SGX enclaves securely with minimal TCB (i.e., the TCB contains only SGX and self-defined code itself). MAAT works as a Java JIT compiler which automatically compiles self-defined big-data functions (e.g., MAP/REDUCE) into enclave-compatible assembly code. Therefore, the JVM itself does not run in MAAT's enclaves.

The goal of MAAT is to preserve the confidentiality of data while being processed in public clouds. Other attacks such as changing execution paths (i.e., integrity) have been well defended in prior work [25], and MAAT can directly use it.

Figure 5 shows MAAT's threat model: both SGX and computation providers are trusted, and cloud providers are malicious. Figure 6 shows the architecture of MAAT. In MAAT, both the translation of self-defined code and the execution of the code are protected by enclaves, so that even if the cloud's OS compromised, it can not see the executions of big-data queries or inject malicious code into the queries during MAAT's translation.

Our MAAT architecture contains two software components: our JIT translator and our own management library. The JIT translator is a thin layer which translate each Java bytecode instruction into a number of SGX-compatible assembly code (e.g., Figure 6). The management is for our own use of encryption/decryption on data records and maintaining SGX memory for the queries at runtime. We will proactively implement the JIT translator and management library to be easy to verify, and we plan to use state-of-the-art verification techniques [47] to verify both of them, so that we do not need to include them into MAAT's TCB. Specifically, we plan to implement the two components without recursions and with as few as loops.

One subtle performance challenge for MAAT is that it should have reasonable performance overhead compared to native executions. When calling into and out of a function in enclaves, an ECALL and OCALL will be invoked in the CPU and enclave transitions are invoked. Such transitions are several hundreds times slower than user function calls. Moreover, encryption and decryption on data records are invoked during such transitions. Our study on a SGX-based big-data system Opaque [78] shows that it incurs 3.4k enclave transitions for processing only 10k data (with two queries `select` and `groupBy`), which confirms the challenge.

To mitigate this challenge, we will create a new enclave runtime abstraction called Data-locality-aware Asynchronous Enclave calls (DAE). DAE converts the synchronous enclave calls (similar to Java function calls) to asynchronous, data-locality-aware calls into enclaves. Specifically, DAE will run a number of n processes (E_1 to E_n) in an enclave on each computer. When a JVM process P calls a big-data query function, the call and its parameters are appended to a queue to DAE, and DAE arranges a process E_i with good data locality (e.g., according to prior arrangements and the decrypted data held by E_i) to execute the call. The call result is appended to a return queue of the DAE for process P. We expect that DAE will achieve reasonable performance, data locality, and parallelism.

Future directions. By realizing a privacy-preserving Java JIT compiler for public clouds, MAAT has broad applications in other security areas, and we will further extend it along three directions. First, we will fully implement it and evaluate its efficacy on defending against diverse real-world privacy attacks launched by cloud providers. Second, we will augment the translator to automatically translate the big-data queries with access patterns on particular data into those without (e.g., oblivious executions). Third, we will further enhance DAE to support well isolated, secured operating system calls (e.g., library operating system calls), so that MAAT will not only benefit big-data queries, but other distributed computing paradigms (e.g., key-value stores and SQL queries).

2.4 Research plan

This project will require two PhD students S1 and S2 to work for three years. In the first year, S1 will design and fully implement the KAKUTE system (part of **Objective 1**), and S2 will evaluate its performance and robustness on various real-world big-data queries (part of **Objective 1**). In the second year, S1 will use KAKUTE to fully develop the proposed fine-grained privacy model (part of **Objective 2**), and S2 will implement the two algorithms proposed for this model (part of **Objective 2**). In the third year, S1 will build the secure big-data compiler **Objective 3** and S2 will evaluate this compiler on diverse real-world big-data queries.

Appendix (C): Non-text figures

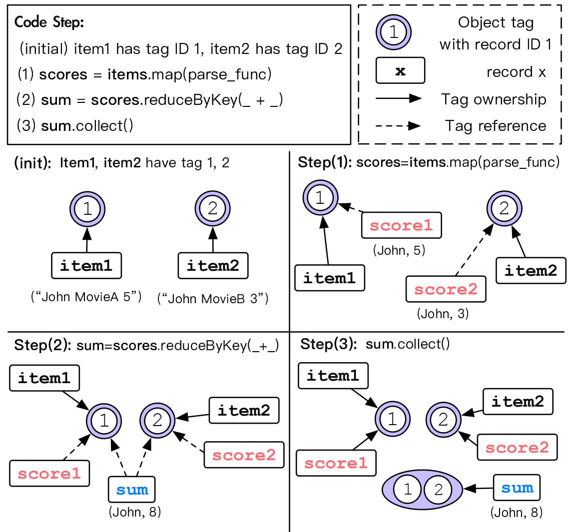

Figure 1: The Reference Propagation technique with the given code (for **Objective 1**).

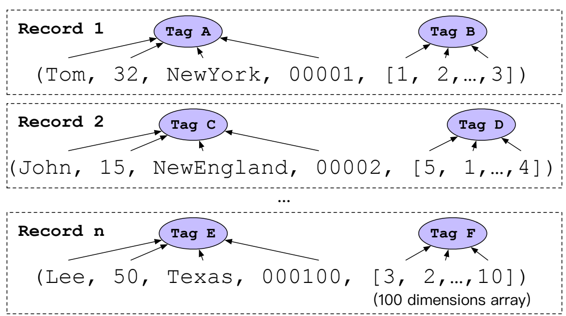

Figure 2: The Tag Sharing technique between fields in each record (for **Objective 1**).

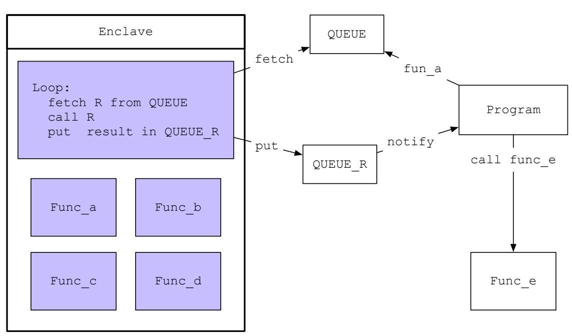

Figure 3: The Data-locality-aware Asynchronous Enclave (DAE) call abstraction (for **Objective 3**).

References

[1] K. A, V. P. M. Shaaban, and W. C.L. Heterogeneous computing: Challenges and opportunities. In *IEEE Computer*, 1993.

[2] S. Akoush, L. Carata, R. Sohan, and A. Hopper. Mrlazy: Lazy runtime label propagation for mapreduce. In *Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'14, pages 17–17, Berkeley, CA, USA, 2014. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2696535.2696552.

[3] C. S. Alliance. Top threats to cloud computing v1.0. https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf, Feb. 2010.

[4] C. S. Alliance. The notorious nine: Cloud computing top threats in 2013. https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf, Feb. 2013.

[5] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[6] S. Bajaj and R. Sion. Trusteddb: A trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 205–216, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: 10.1145/1989323.1989346. URL http://doi.acm.org/10.1145/1989323.1989346.

[7] J. Bell and G. Kaiser. Phosphor: Illuminating dynamic data flow in commodity jvms. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 83–101, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2585-1. doi: 10.1145/2660193.2660212. URL http://doi.acm.org/10.1145/2660193.2660212.

[8] C. B.W.L., W. C.L., and K. Hwang. A migrating-home protocol for implementing scope consistency model on a cluster of workstations. In *PDPTA*, 1999.

[9] Z. Chothia, J. Liagouris, F. McSherry, and T. Roscoe. Explaining outputs in modern data analytics. *Proceedings of the VLDB Endowment*, 9(12):1137–1148, 2016.

[10] H. Cui, J. Wu, C.-C. Tsai, and J. Yang. Stable deterministic multithreading through schedule memoization. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[11] H. Cui, J. Wu, J. Gallagher, H. Guo, and J. Yang. Efficient deterministic multithreading through schedule relaxation. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 337–351, Oct. 2011.

[12] H. Cui, J. Simsa, Y.-H. Lin, H. Li, B. Blum, X. Xu, J. Yang, G. A. Gibson, and R. E. Bryant. Parrot: a practical runtime for deterministic, stable, and reliable threads. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Nov. 2013.

[13] H. Cui, R. Gu, C. Liu, and J. Yang. Paxos made transparent. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[14] A. Dave and M. Zaharia. Arthur: Rich post-facto debugging for production analytics applications.

[15] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, 2004.

[16] C. Dwork. Differential privacy. *Lecture Notes in Computer Science*, 26(2):1–12, 2006.

[17] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of MODELS of Computation*, pages 1–19, 2008.

[18] C. Dwork. A firm foundation for private data analysis. *Communications of The ACM*, 54(1):86–95, 2011.

[19] C. Dwork and J. Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380. ACM, 2009.

[20] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876, pages 265–284. Springer, 2006.

[21] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-32731-2, 978-3-540-32731-8. doi: 10.1007/11681878_14. URL http://dx.doi.org/10.1007/11681878_14.

[22] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[23] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, pages 1–6, 2010.

[24] L. Feng, L. F.C.M., C. Heming, and W. Cho-Li. Confluence: Speeding up iterative distributed operations by key-dependency-aware partitioning. In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2017.

[25] T. Frassetto, D. Gens, C. Liebchen, and A.-R. Sadeghi. Jitguard: Hardening just-in-time compilers with sgx. 2017.

[26] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.

[27] C. Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[28] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim. Bigdebug: Debugging primitives for interactive big data processing in spark. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 784–795, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3900-1. doi: 10.1145/2884781.2884813. URL `http://doi.acm.org/10.1145/2884781.2884813`.

[29] Hadoop. Hadoop. `http://hadoop.apache.org/core/`.

[30] X. Hu, M. Yuan, J. Yao, Y. Deng, L. Chen, Q. Yang, H. Guan, and J. Zeng. Differential privacy in telco big data platform. *Proceedings of the VLDB Endowment*, 8(12):1692–1703, 2015.

[31] Intel. Software guard extensions programming reference. `https://software.intel.com/sites/default/files/329298-001.pdf`.

[32] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, and T. Condie. Titian: Data provenance support in spark. *Proc. VLDB Endow.*, 9(3):216–227, Nov. 2015. ISSN 2150-8097. doi: 10.14778/2850583.2850595. URL `http://dx.doi.org/10.14778/2850583.2850595`.

[33] J. Jianyu, Z. Shixiong, A. Danish, W. Yuexuan, C. Heming, L. Feng, and G. Zhaoquan. Kakute: A precise, unified information flow analysis system for big-data security. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC '17)*, 2017.

[34] S. D. I. John and A. O. Osonde. Privacy preservation in the age of big data. In *RAND '16*, 2016.

[35] Z. Jorgensen, T. Yu, and G. Cormode. Conservative or liberal? personalized differential privacy. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1023–1034. IEEE, 2015.

[36] H. K., J. H., C. E., W. C.L., and X. Z. Designing ssi clusters with hierarchical checkpointing and single i/o space. In *IEEE Concurrency*, 1999.

[37] M. Kazim and S. Y. Zhu. A survey on top security threats in cloud computing. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2015.

[38] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis. Libdft: Practical dynamic data flow tracking for commodity systems. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, VEE '12, pages 121–132, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1176-2. doi: 10.1145/2151024.2151042. URL `http://doi.acm.org/10.1145/2151024.2151042`.

[39] L. King-Tin, S. Jinghao, H. Dominic, W. Cho-Li, L. Zhiquan, Z. Wangbin, and Y. Youliang. Rhymes: A shared virtual memory system for non-coherent tiled many-core architectures. In *ICPADS 2014*, 2014.

[40] D. Logothetis, S. De, and K. Yocum. Scalable lineage capture for debugging disc analytics. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 17. ACM, 2013.

[41] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 24–24. IEEE, 2006.

[42] F. McSherry. Privacy integrated queries. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Association for Computing Machinery, Inc., June 2009. URL `https://www.microsoft.com/en-us/research/publication/privacy-integrated-queries/`.

[43] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636. ACM, 2009.

[44] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3010-9. doi: 10.1109/FOCS.2007.41. URL `http://dx.doi.org/10.1109/FOCS.2007.41`.

[45] M. M.J.M., W. C.L., and L. F.C.M. Jessica: Java-enabled single-system-image computing architecture. In *Journal of Parallel and Distributed Computing*, 2000.

[46] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. Gupt: Privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 349–360, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi: 10.1145/2213836.2213876. URL `http://doi.acm.org/10.1145/2213836.2213876`.

[47] L. Nelson, H. Sigurbjarnarson, K. Zhang, D. Johnson, J. Bornholt, E. Torlak, and X. Wang. Hyperkernel: Push-button verification of an os kernel. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*.

[48] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

[49] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, Austin, TX, 2016. USENIX Association. ISBN 978-1-931971-32-4. URL `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko`.

[50] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[51] P. Paillier et al. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, volume 99, pages 223–238. Springer, 1999.

[52] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *OSDI*, pages 587–602, 2016.

[53] V. Pappas, V. P. Kemerlis, A. Zavou, M. Polychronakis, and A. D. Keromytis. Cloudfence: Data flow tracking as a cloud service. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 8145*, RAID 2013, pages 411–431, New York, NY, USA, 2013. Springer-Verlag New York, Inc. ISBN 978-3-642-41283-7. doi: 10.1007/978-3-642-41284-4_21. URL http://dx.doi.org/10.1007/978-3-642-41284-4_21.

[54] pigmix. https://cwiki.apache.org/confluence/display/PIG/PigMix.

[55] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[56] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855711.1855731.

[57] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 38–54. IEEE, 2015.

[58] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan. Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors. In *Proceedings of the 17th ACM conference on Computer and communications security (CCS '10)*, 2017.

[59] D. Sheng and W. Cho-Li. Error-tolerant resource allocation and payment minimization for cloud system. In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2013.

[60] D. Sheng, R. Yves, V. Frederic, K. Derrick, W. Cho-Li, and C. Franck. Optimization of cloud task processing with checkpoint-restart mechanism. In *SC '13*, 2013.

[61] D. Sheng, K. Derrick, and W. Cho-Li. Optimization of composite cloud service processing with virtual machines. In *IEEE Transactions on Computers*, 2014.

[62] A. Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 813–822, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0691-1. doi: 10.1145/1993636.1993743. URL http://doi.acm.org/10.1145/1993636.1993743.

[63] Spark example. https://spark.apache.org/examples.html.

[64] J. J. Stephen, S. Savvides, R. Seidel, and P. Eugster. Practical confidentiality preserving big data analysis. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, PA, 2014. USENIX Association. URL https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/stephen.

[65] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[66] Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda. CleanOS: limiting mobile data exposure with idle eviction. In *Proceedings of the Tenth Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 77–91, 2012.

[67] S. Technology. 7 most infamous cloud security breaches. https://www.storagecraft.com/blog/7-infamous-cloud-security-breaches/, Feb. 2017.

[68] S. D. Tetali, M. Lesani, R. Majumdar, and T. Millstein. Mrcrypt: Static analysis for secure cloud computations. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 271–286, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2374-1. doi: 10.1145/2509136.2509554. URL http://doi.acm.org/10.1145/2509136.2509554.

[69] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the VLDB Endowment*, volume 6, pages 289–300. VLDB Endowment, 2013.

[70] Z. W, W. Cho-Li, , and L. F.C.M. Jessica2: A distributed java virtual machine with transparent thread migration support. In *IEEE Fourth International Conference on Cluster Computing (Cluster2002)*, 2002.

[71] C. Wang, J. Yang, N. Yi, and H. Cui. Tripod: An efficient, highly-available cluster management system. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, 2016.

[72] C. Wang, J. Jiang, X. Chen, N. Yi, and H. Cui. Apus: Fast and scalable paxos on rdma. In *Proceedings of the Eighteenth ACM Symposium on Cloud Computing*, pages 17–28. ACM, 2017.

[73] N. World. Leaked icloud credentials obtained from third parties, apple says. https://www.networkworld.com/article/3184471/security/leaked-icloud-credentials-obtained-from-third-parties-apple-says.html, Feb. 2017.

[74] J. Yang, H. Cui, J. Wu, Y. Tang, and G. Hu. Determinism is not enough: Making parallel programs reliable with stable multithreading. *Communications of the ACM*, 2014.

[75] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language.

[76] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[77] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 515–526. ACM, 2011.

[78] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, pages 283–298, 2017.

[79] L. Zhiquan, L. King-Tin, W. Cho-Li, , and S. Jinshu. Powerock: Power modeling and flexible dynamic power management for many-core architectures. In *IEEE Systems Journal*, 2016.

[80] T. Zhu, G. Li, W. Zhou, and S. Y. Philip. Differential privacy and applications, 2017.