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

Abstract:
In this big-data era, many business vendors (e.g., Uber) store data on clouds. Meanwhile, business vendors and third-parties implement self-defined queries (e.g., MapReduce) to process data, causing two severe privacy problems. First, third-parties can easily leak sensitive fields in data records (e.g., credit cards in Uber orders) through query results. Second, careless or malicious cloud providers can observe the data being queried.

This proposal aims to preserve big-data privacy with a holistic methodology: people can still run unmodified big-data queries, and this proposal will automatically prevent sensitive data leakage at runtime by accomplishing three objectives.

First, to confine malicious third-parties, we will build Kakute, the first Data Flow Tracking (DFT) system for big-data. Kakute provides easy-to-use APIs for business vendors to tag sensitive data fields, it then automatically tracks unmodified queries and prevents sensitive data flowing to query results. A challenge in existing DFT systems is that propagating tags in data-intensive computations is too slow (e.g., a notable DFT system incurs 128X performance overhead compared to native, insecure queries). By leveraging subtle efficiency features of big-data queries, we will create two fast tag propagation techniques. Our Kakute preliminary prototype presented in [ACSAC '17] incurs merely 32.3% performance overhead.

Second, we will create a Fine-grained Differential Privacy (FDP) technique and its new algorithms. Kakute and differential privacy are synergistic on confining malicious third-parties, because differential privacy allows aggregation computations on sensitive data by adding noise to hide individual information. Unfortunately, existing differential privacy techniques are coarse-grained (inaccurate): they often conservatively add excessive noise to all query results, because they cannot track which sensitive inputs flowed to which results. By leveraging Kakute, our FDP technique automatically adds noise to only the sensitive (tagged) parts of results, effectively hiding sensitive data and making most results accurate.

Third, to confine malicious cloud providers, we will leverage the Intel SGX (Software Guard Extensions) hardware to build the first privacy-preserving compiler for unmodified big-data queries. Existing SGX-based systems for big-data have two major challenges: they have to rewrite the queries from Java to SGX-compatible C++, or their trusted components running in SGX are too large (e.g., an entire JVM). To tackle these challenges, our compiler runs only the Java big-data queries in SGX using a thin, verified just-in-time translator created by us. The compiler also carries our new efficient SGX-based runtime techniques.

The success of this proposal will effectively preserve big-data privacy in clouds and benefit most people.

Long term impact:
The big-data and cloud computing trends enable great opportunities to all entities, including data providers (e.g., business vendors and computer users), cloud providers (e.g., Amazon), and computation providers who implement big-data queries (e.g., business vendors and their third-party partners). Unfortunately, despite decades of effort, data leakage remains one of the most severe threats in clouds. In the perspective of data providers (owners), severe data leakage incidents have been triggered by both computation providers (e.g., some iCloud third-parties leaked celebrity accounts on Internet in 2017) and cloud providers (e.g., the 2013 Yahoo Cloud compromise and the 2014 J.P. Morgan account leakage).

In the perspective of big-data queries, data leakage have happened both from within and outside. When a query runs, code running within can often be buggy or malicious. Cloud providers, which run outside the queries, have also incurred numerous data leakage incidents due to compromises on external and insider attacks. Therefore, this proposal takes a holistic methodology: preventing leakage for data providers from both within (Objective 1 and 2) and outside (Objective 3) the queries.

In the short term, to confine malicious third-parties in private clouds (i.e., a data provider is the cloud provider), we plan to accomplish both Objective 1 and Objective 2. Objective 1 proposes Kakute, the first Data Flow Tracking (DFT) system for big data queries. Kakute tackles a notorious performance challenge on porting DFT to data-intensive computations. To achieve a robust DFT architecture for distributed big-data frameworks (e.g., Spark), Kakute completely captures the frameworks' inter-computer data flows. 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 incurs a moderate performance overhead of 32.3% compared to native, insecure queries. It also effectively detects 13 real-world security and performance bugs. These promising preliminary results have been presented in [ACSAC '17] and [TPDS '17].

Kakute and differential privacy are synergistic on confining malicious third-parties: Kakute enforces mandatory access control on sensitive data, but it may cause some query results containing sensitive data to be missing; differential privacy allows the aggregation computations on sensitive data while hiding individual privacy, but due to the lack of precise data flow tracking, it often suffers from excessive noise and inaccurate results. Therefore, the Objective 2 of this proposal takes the first significant step to integrate DFT and differential privacy, leading to a novel Fine-grained Differential Privacy (FDP) technique. By leveraging Kakute, FDP automatically adds noise to only the sensitive parts of results, effectively hiding sensitive data and making most results accurate.

In the intermediate term, we plan to confine malicious public cloud providers by accomplishing Objective 3. The Intel SGX hardware enforces strong confidentiality for data and code even if the cloud is malicious. Moreover, SGX is a good fit for big-data queries because these queries do data-intensive computations in user space and rarely invoke system calls.

However, existing SGX-based systems for big-data have two major challenges: they have to manually rewrite the Java big-data queries into SGX-compatible C++, or their trusted computing base (TCB) is too large (e.g., an entire JVM). To tackle these two challenges, Objective 3 will build the first privacy-preserving Java compiler with minimum TCB. Our compiler will run only the Java big-data 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.

In the long term, by integrating the outcomes of all the three objectives in this proposal and extensively applying them on real-world software, we can help data providers enforce comprehensive privacy against both computation providers and cloud providers. This can benefit most people. For instance, many Hong Kong finance companies demand strong privacy for their data deployed in clouds.

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 will make Kakute incur reasonable performance overhead compared to 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 algorithms, which will only add noise to sensitive data fields or query results, preserving strong differential privacy for sensitive data and good accuracy for most 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, verified translator that automatically translates Java bytecode to SGX-compatible code. We will quantitatively evaluate the performance overhead of our compiler and whether it can protect data privacy against real-world privileged attacks.

1 Research Background

This proposal has three entities: data providers, cloud providers, and computation providers (who write big-data queries). In private clouds, data providers are cloud providers; in public clouds, they differ. This section shows relevant techniques (§1.1, §1.2, and §1.3), motivation (§1.4), and related work (§1.5 and §1.6).

1.1 Big-data computing frameworks

Big-data frameworks (e.g., Spark [75] and MapReduce [28]) are popular for computations on tremendous amounts of data records. These frameworks provide self-defined Java functions (e.g., `map`/`reduce`) to let computation providers write their algorithms, and the frameworks automatically apply these functions on the data stored across computers in parallel.

To avoid excessive computation, big-data frameworks adopt a *lazy transformation* approach [48, 75]. Spark often uses lazy transformations (e.g., `map`), and calls to these transformations only create a data structure called RDD with *lineage* (the sequence of transformations on data records). Actual transformations are only triggered when materialization operations (e.g., `collect`/`count`) are called. Collecting operations trigger transformations only along lineages, so unnecessary computations are avoided. **Objective 1** will leverage lazy transformation to create a fast data flow tracking technique: Reference Propagation (§2.1).

1.2 Software-based privacy techniques

Data Flow Tracking (DFT) is a powerful mandatory access control technique for preventing sensitive data leakage [22]. DFT attaches a tag to a variable (or object), and this tag will propagate during computations on the variable at runtime. DFT is used in various areas, including preventing sensitive data (e.g.contacts) leakage in smart phones (TaintDroid [22]), web services [50], and server programs [36]. No DFT system exists for big-data computing, so **Objective 1** (§2.1) will create the first DFT system for big-data.

Complimentary to DFT, statistical techniques, including K-anonymization methods [41, 62] and differential privacy [43, 45, 53], allow the aggregation of sensitive data while adding random noise on inputs or query results to preserve individual privacy. However, statistical techniques are either not secure (K-anonymization) or suffering from great losses of accuracy (differential privacy). Recent work [29] reports more than 30% loss of accuracy. For query results, low accuracy means bad utility. For instance, a K-Means program will return centroids far from the actual ones, because the accuracy loss rate is much larger than the training error rate (usually less than 10% in practice).

A key reason for this bad utility problem is that differential privacy cannot track how sensitive data fields flow to query results, so they have to take a coarse-grained approach, which conservatively adds noise to all data fields or all query results. **Objective 2** (§2.2) will propose 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 and hypervisors compromise. For instance, Intel-SGX [30], 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 100X to 1000X faster. For instance, a SGX-based system Opaque [76] incurs a moderate performance overhead of 30% compared to native, insecure big-data queries.

However, to practically run Java big-data queries with SGX, two open challenges remain. First, existing SGX-based systems [76] 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 [55]) 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 be verified. **Objective 3** (§2.3) tackles these two 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 [35], is a top security threat [7, 33] in cloud computing. In a data provider's perspective, both computation providers (e.g., the 2017 iCloud account leakage caused by third-parties [72]) and cloud providers (e.g., the 2013 Yahoo Cloud compromise [64]) 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 [26] is a technique for computing on encrypted data in untrusted environments. There are two kinds of homomorphic encryption methods: Fully homomorphic encryption (FHE) and Partially Homomorphic Encryption (PHE). An evaluation [25] on FHE shows a slowdown of orders of magnitudes, unacceptable in practice. Systems that adopts PHE (e.g., Monomi [66] and CryptDB [52]) report reasonable overhead, but PHE has limited expressiveness (e.g., supports only a small subset of arithmetic instructions) or requires extra trusted servers. Seabed [49] uses asymmetric encryption schemes to reduce the performance overhead of AHE, but its expressiveness is still quite limited. Therefore, in commercial clouds, most data is processed in plaintext.

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 incur much lower overhead. TrustedDB [9] is a hardware-based secure database. VC3 [54] proposes a secure distributed analytic platform with SGX protection on MapReduce [19]. Opaque [76] supports secure and oblivious SQL operators on SparkSQL [8]. However, all these systems have limited expressiveness (e.g., SQL operators), and VC3 even needs to rewrite the program with C++. A recent work [47] proposes an oblivious machine leaning framework on trusted processors. SGX-BigMatrix [55] proposes an oblivious and secure vectorization abstraction on Python, but its trusted components are too big (an entire Python runtime).

Big-data privacy systems. Airavat[53], PINQ [43], and GUPT [45] propose to apply differential privacy [21] in MapReduce in order to prevent leakage from malicous queries, but existing differential privacy techniques often produce inaccurate results. MrLazy [6] preserves the privacy of bog-data queries with static DFT. Compared to dynamic DFT (§1.2), static DFT is imprecise and may suffer from false positives.

Data and software integrity. This proposal focuses on preserving the privacy of data, while integrity is an orthogonal area and not the focus of this proposal. Prior work is effective on preserving the integrity of data [68], big-data frameworks (e.g., IntegrityMR [71]), and compilers (e.g., JITGuard [24]). These orthogonal integrity-preserving techniques can be directly used in our proposed systems.

1.6 Related work by the PI and co-I

The PI is an expert on secure and reliable distributed systems [13–17, 23, 32, 69, 70, 73, 74]. The PI's work is published in top conferences and journals on systems (OSDI, SOSP, SOCC, TPDS, and ACSAC) and programming languages (PLDI and ASPLOS). The PI has built tools [15, 73] to detect various new security vulnerabilities in widely used real-world software, including data loss [15], buffer overflows [73], and Linux kernel compromises (CVE-2017-7533 [2] and CVE-2017-12193 [3]). The co-I is an expert on high-performance computing [4, 11, 34, 44, 56, 57, 77], big-data computing [23, 38], security [40, 63], and Java compilers [37, 58, 67]. The co-I's work is published in top systems conferences (Cluster, SC, and ICPADS) and journals (JPDC, TPDS, and IEEE Transactions on Computers). As preliminary results for this proposal, the PI has presented KAKUTE [32] in ACSAC '17, and the PI and co-I have collaborated to present CONFLUENCE [23] 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 [10] 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 [32]. 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 [50] 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

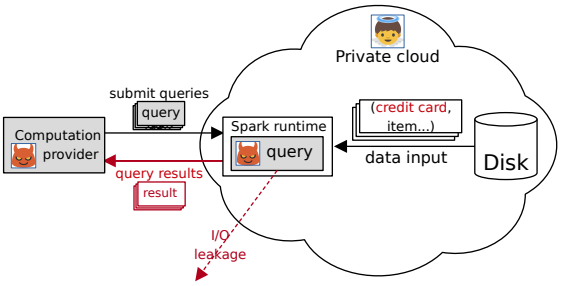

Figure 1: Threat model of KAKUTE. Red colors means sensitive data or leaking channels. Shaded (grey) components may leak data, and KAKUTE is designed to defend against them.

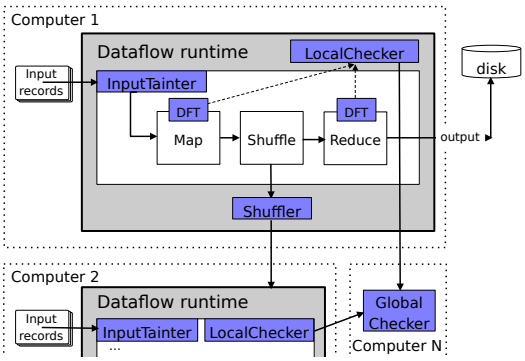

Figure 2: KAKUTE architecture. KAKUTE's key components are shaded (and in blue).

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 a Taobao order record ⟨`time`, `userId`, `productID`⟩, only the `userId` 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 inter-computer data flows (shuffles), 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 plan to implement KAKUTE and integrate it with Spark. We will leverage Phosphor [10], 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 [31] and debugging big-data queries [27].

Preliminary results. We have implemented a KAKUTE prototype and evaluated it on six popular big-data queries, including three text processing queries WordCount [61], WordGrep [39] and TwitterHot [61], two graph queries TentativeClosure [61] and ConnectComponent [61], and one meical query MedicalGroup [51]. We ran them with large, realistic datasets [12, 27, 31]. Our evaluation shows that: (1) KAKUTE incurred merely 32.3% overhead (Figure 3) with INTEGER tag, two orders of magnitudes faster than a recent DFT system Phorspor [10]; and (2) KAKUTE effectively detected 13 real-world security and performance bugs in other papers [18, 27, 53]. These promising preliminary results are presented in ACSAC '17 and TPDS '17.

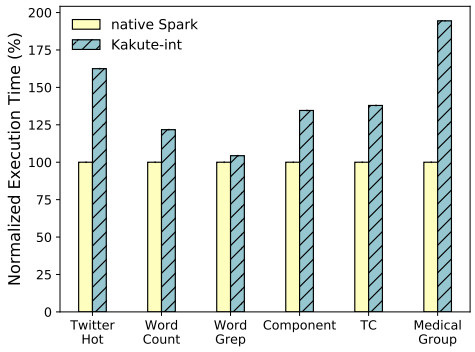

Figure 3: KAKUTE execution time normalized to native Spark executions. 100% means no overhead.

Future directions. We plan to extend KAKUTE in three directions. First, we will port KAKUTE onto more big-data frameworks, including PIG [48] and HADOOP [28]. 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 differential privacy (i.e., **Objective 2**), K-anonymity [62], and L-diversity [41], which will likely lead to the inventions of diverse privacy-preserving techniques and systems.

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 [20] 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 [29] 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 aggregation 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 have higher security sensitivity levels, then we can add more noise accordingly.

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. $\Phi(x)$ returns 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 shows the workflow of FDP with five steps.

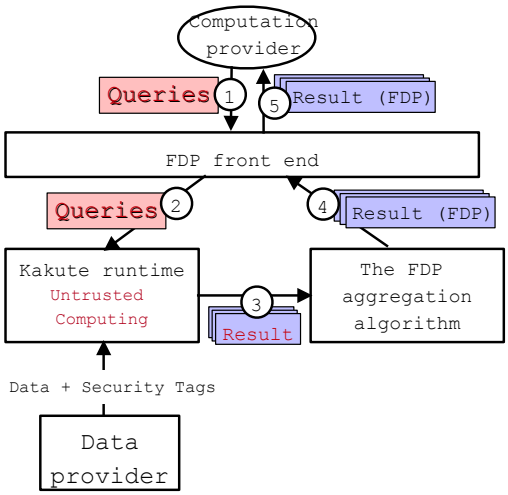

Figure 4: The workflow of FDP with five steps.

Algorithm 1: The FDP aggregation algorithm

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

n = a suitable partition

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

  $O_i \leftarrow Query(D_i)$;

  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 $\leftarrow getTagLevel(O_j)$;

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

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. To realize a differential privacy technique, FDP first needs to determine privacy budgets and security levels (`getTagLevel` in **Algorithm 1**). We use a common differential privacy model with 6 security levels: insecure, dp_1, dp_2, dp_3, dp_4 and non-released. "Insecure" data can be released in query results directly, while "non-released" data must not be released in any condition. dp_1 to dp_4 vary in terms of their privacy budgets for differential privacy.

Theorem 2.1. *Laplace Mechanism [20] 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 enforce differential privacy, one approach is to use the ε inferred by the highest security level of each dimension and to add noise to the output directly, but this approach is too naive for the diversity of security levels. Instead, we adopt the sample-and-aggregate approach in previous work [60]. In this approach, data is partitioned into multiple parts (each part has a size of m) to reduce Δf. Each record x has a security level of ε_x. **Algorithm 1** adds noise to the query results O_i of each partition D_i according to the D_i's security level.

Theorem 2.2. *For any output record O, the mechanism of* ***Algorithm 1*** *is* $max_{i\leq k}(\varepsilon_i)$*-differentially private.*

Proof. For each dimension, we can divide the output dataset D as k disjoin partitions D_1, D_2, ..., D_k according to their security levels. According to prior work[59], each $\mathscr{M}(D_i)(i \leq k)$ is ε_k-differentially private. For each D_i, we have $Pr[\mathscr{M}(D_i) \in S] \leq e^{\varepsilon_i} Pr[\mathscr{M}(D_i') \in S]$, suppose $\varepsilon_{max} = max_{i\leq k}(\varepsilon_i)$ and the differing record x of D and D' is in partition D_x,

$$\begin{aligned} Pr[\mathscr{M}(D)] &= Pr[\mathscr{M}(D_1 + D_2 + \ldots + D_k)] \\ &= Pr[\mathscr{M}(D_1)] + Pr[\mathscr{M}(D_2)] + \ldots + Pr[\mathscr{M}(D_k)] \\ &\leq Pr[\mathscr{M}(D_1)] + \ldots + e^{\varepsilon_x} Pr[\mathscr{M}(D_x')] + \cdots + Pr[\mathscr{M}(D_k)] \\ &\leq e^{\varepsilon_{max}}(Pr[\mathscr{M}(D_1)] + \ldots + Pr[\mathscr{M}(D_x')] + \ldots + Pr[\mathscr{M}(D_k)]) \\ &= e^{\varepsilon_{max}} Pr[\mathscr{M}(D')] \end{aligned} \tag{3}$$

Therefore, $\mathscr{M}(D)$ (i.e., **Algorithm 1**) is $max_{i\leq k}(\varepsilon_i)$-differentially private. □

Future directions. We will enrich our FDP technique by going along two directions. First, we will further optimize the algorithm to reduce its added noise. In the last line of **Algorithm 1**, the added noise on O_j comes from two parts: the Laplace noise added in each partition and the sum of noise added to all partitions. When the number of partitions increases, the Laplace per partition decreases (each partition is smaller), but the sum increases. This brings an interesting optimization problem on minimizing added noise, and we plan to create other algorithms (e.g., algorithms with hill-climbing or simulated-annealing styles) to compute the optimal number of partitions for minimizing added noise. Second, we will conduct an extensive study on diverse real-world big-data queries and realistic datasets in order to quantify the accuracy improvements of FDP and its new algorithms compared to existing differential privacy techniques (e.g., GUPT [45]).

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

Recent real-world privacy breaches have shown that sensitive data are often leaked while being processed in public clouds, including clouds compromises on external attacks [72] and insider attacks [7]. 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 [30] runs programs in an enclave, so code and data are protected and cannot 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 [76] 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 SGX-based systems (Opaque [76], VC3 [54], Azure/Coco [1], and SGX-BigMatrix [55]) for big-data are promising, two major challenges remain. First, these systems require completely rewriting the readily pervasive and familiar Java big-data queries into C++ [1, 54, 76], a time-consuming and error-prone process. Second, to easy implementation, existing systems often run an entire language runtime (e.g., JVM or Python runtime [55]) within SGX, causing the Trusted Computing Base (TCB) to be too large (JVM has millions of lines of code developed by many companies, so it is vulnerable to bugs or insider attacks [7]).

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

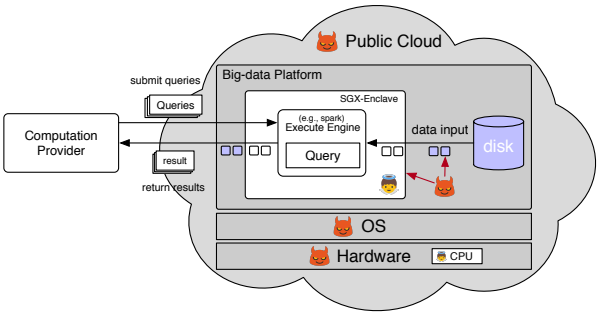

Figure 5: Threat model of MAAT. Data records with blue color are encrypted, and white color are plaintext. Shaded (grey) components may leak data, and MAAT is designed to defend against them.

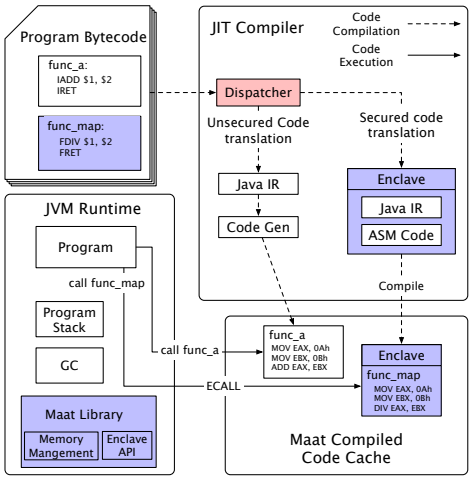

Figure 6: MAAT compiler architecture. Key components are shaded (and blue).

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 SGX-compatible assembly instructions. Therefore, the JVM itself does not run in MAAT's enclaves.

MAAT's goal is to preserve the privacy (confidentiality) of data in public clouds. Other attacks such as changing the execution paths (i.e., integrity) of big-data frameworks or language runtimes have been well defended (e.g., IntegrityMR [71] and JITGuard [24]), and MAAT can directly use these techniques.

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 cannot see the executions of big-data queries or inject malicious code into the queries during MAAT's translation. Translated functions are put in MAAT's code cache for future reuse.

Two MAAT software components run in SGX: our JIT translator and our own management library. The JIT translator is a thin layer which translates a Java bytecode instruction into a number of SGX-compatible assembly instructions. For instance, in Figure 6, an FDIV Java bytecode instruction translates to two MOV and one DIV assembly instructions). The memory management library is for our own use of encryption/decryption on data records and maintaining SGX memory for the queries at runtime. We will proactively implement these two components to be easy to verify (use as few as function recursive calls and loops) as in other verification practice [46], and we plan to use state-of-the-art verification techniques [46] to assure that they both are functionally correct (i.e., free of bugs [46] and malicious code [5]). Then, we do not need to include them in MAAT's TCB, greatly reducing its TCB.

A subtle performance challenge for MAAT is that it should have reasonable performance overhead compared to native executions. When a program calls into (i.e., ECALL) and gets out of (i.e., OCALL) a function in enclaves, enclave transitions are invoked. Such transitions are several hundred times slower than user function calls. Moreover, encryption and decryption on data records are invoked during such transitions. Our study on a SGX-based system Opaque [76] 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 further enhance DAE to support well isolated, secured operating system calls (e.g., library operating system calls [65]), so that MAAT will not only benefit big-data queries, but other distributed computing paradigms (e.g., graph queries [42]). Third, we will augment the translator to automatically transform the big-data queries with dangerous access patterns on particular data into those without (i.e., oblivious executions [55]). This will nurture new theory and algorithms on oblivious executions.

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 security strength 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 Differential Privacy technique (part of **Objective 2**), and S2 will implement the algorithm proposed for this technique (part of **Objective 2**). In the third year, S1 will build the secure big-data compiler (part of **Objective 3**), and S2 will extensively study the privacy guarantee and performance of this compiler on real-world big-data frameworks (part of **Objective 3**).

Appendix (C): Non-text figures

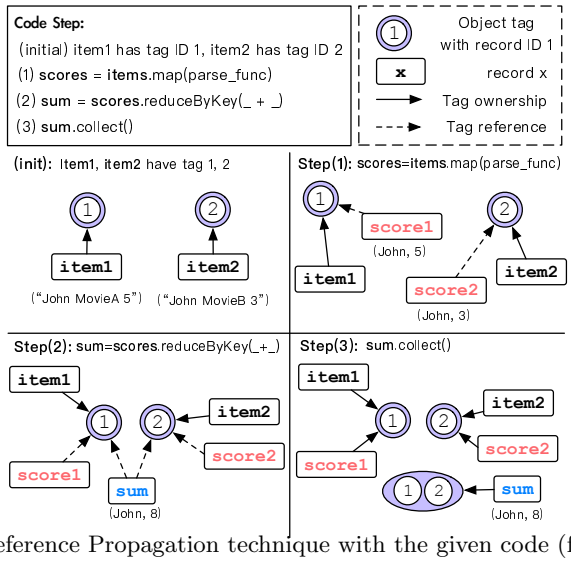

Figure 1: The Reference Propagation technique with the given code (for **Objective 1**).

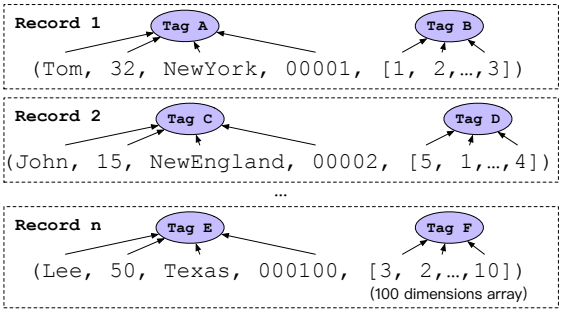

Figure 2: The Tag Sharing technique between fields in each record (for **Objective 1**).

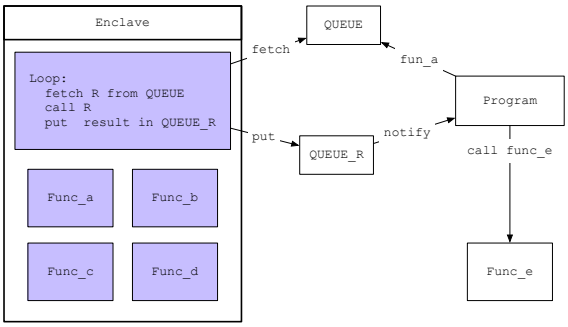

Figure 3: The Data-locality-aware Asynchronous Enclave (DAE) call abstraction (for **Objective 3**).

References

[1] GitHub - Azure/coco-framework. https://github.com/Azure/coco-framework.

[2] CVE-2017-7533. http://seclists.org/oss-sec/2017/q3/240.

[3] CVE-2017-12193. https://access.redhat.com/security/cve/cve-2017-12193.

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

[5] F. Adelstein, M. Stillerman, and D. Kozen. Malicious code detection for open firmware. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 403–412. IEEE, 2002.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] H. Cui, G. Hu, J. Wu, and J. Yang. Verifying systems rules using rule-directed symbolic execution. In *Eighteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '13)*, 2013.

[16] 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.

[17] 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.

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

[19] 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.

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

[23] 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.

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

[25] 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.

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

[27] 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.

[28] Hadoop. Hadoop. http://hadoop.apache.org/core/.

[29] 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.

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

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Z. Lai, K. T. Lam, C.-L. Wang, J. Su, Y. Yan, and W. Zhu. Latency-aware dynamic voltage and frequency scaling on many-core architectures for data-intensive applications. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 78–83. IEEE, 2013.

[39] 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.

[40] T. Ma, L. Chen, C.-L. Wang, and F. C. Lau. G-pass: An instance-oriented security infrastructure for grid travelers. In *Computation and Currency: Practice and Experience*. IEEE, 2006.

[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] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[43] 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/.

[44] 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.

[45] 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.

[46] 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)*.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

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

[59] A. Smith. Efficient, differentially private point estimators. *arXiv preprint arXiv:0809.4794*, 2008.

[60] 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.

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

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

[63] F. Tang, M. Guo, M. Li, C.-L. Wang, and M. Dong. Secure routing for wireless mesh sensor networks in pervasive environments. In *INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS*. IEEE, 2007.

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

[65] C.-C. Tsai, D. E. Porter, and M. Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[66] 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.

[67] 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.

[68] B. Wang, B. Li, and H. Li. Panda: Public auditing for shared data with efficient user revocation in the cloud. *IEEE Transactions on services computing*, 8(1):92–106, 2015.

[69] 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.

[70] 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.

[71] Y. Wang, J. Wei, M. Srivatsa, Y. Duan, and W. Du. Integritymr: Integrity assurance framework for big data analytics and management applications. In *Big Data, 2013 IEEE International Conference on*, pages 33–40. IEEE, 2013.

[72] 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.

[73] J. Wu, Y. Tang, G. Hu, H. Cui, and J. Yang. Sound and precise analysis of parallel programs through schedule specialization. In *Proceedings of the ACM SIGPLAN 2012 Conference on Programming Language Design and Implementation (PLDI '12)*, pages 205–216, June 2012.

[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] 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.

[76] 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.

[77] 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.