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.