Privacy-Preserving Techniques and System for Streaming Databases

Anderson Santana de Oliveira
SAP Research
Security & Trust Practice,
Sophia-Antipolis, France
Email: anderson.santana.de.oliveira@sap.com

Florian Kerschbaum
Chair of Privacy and Data Security
Technical University Dresden
Dresden, Germany
Email:florian.kerschbaum@tu-dresden.de

Hoon Wei Lim
School of Physical & Mathematical Sciences
Nanyang Technological University
Nanyang, Singapore
Email:hoonwei@ntu.edu.sg

Su-Yang Yu
School of Computing Science
University of Newcastle
Newcastle upon Tyne, UK
Email: s.y.yu@ncl.ac.uk

***Abstract*—Streaming databases and other distributed, event-based systems are very useful tools for business and security applications. When event sources and event processing are distributed across multiple distinct domains, confidentiality and privacy issues emerge. These can be addressed by a number of cryptographic techniques. In this paper we consider high-performance symmetric encryption techniques. We build a system for privacy-preserving event correlation and evaluate the performance of the its techniques. We demonstrate efficient privacy-preserving event correlation using equality tests, greater-than comparisons and range queries. The results indicate that in comparable settings, it is therefore recommended to employ these techniques to address pertinent security and privacy concerns.**

***Keywords*-Bloom filters, order-preserving encryption, privacy-preserving correlation, streaming databases**

I. Introduction

Streaming databases [2] allow efficiently processing events, i.e. single database tuples. Complex event processing [20] helps aggregate single, stateless events into complex, stateful events. They have many applications, e.g. in supply chain execution [26] or intrusion detection [18], [19], [23].

In a distributed event-based system, data sources and data sinks (also called event processors), are distributed across different domains. In this distributed setting, confidentiality or – in case of person-related data – privacy concerns arise between distrustful parties. Event sources may not want to reveal their event data to the event processor. This may even be necessitated by privacy legislation, such as the Health Insurance Portability and Accountability Act Privacy Rule [24], the Data Protection Act [10], or the Fair Credit Reporting Act [12]. Yet, the event processor must still be able to infer complex events, e.g. a counterfeit item or a coordinated network attack.

This conflict can be solved using a number of techniques. Secure computation [16], [27] allows the distributed computation of any function (with a public output) while preserving the privacy and confidentiality of the inputs. Public key homomorphic encryption [14], [22] allows the computation of any function on encrypted inputs (with an encrypted output). Yet, all of these techniques generally suffer from poor computational performance, e.g. [15], or require bi-directional communication between event sources and processor. They are not able to handle high event arrival rates on commodity hardware.

In this paper we therefore consider high-performance symmetric encryption techniques, i.e. of practical interest, for privacy-preserving event correlation. These techniques enable a non-interactive evaluation of the query at the event processor and thus perform significantly better in WAN settings than their interactive alternatives. The basic approach is to not modify the event processing engine, but only enable existing queries to run on encrypted data. We investigate the performance penalty of these techniques and build a prototypical system implementing them. Hence, we allow the choice of its use according to application-specific constraints. Such a choice is a typical instance of a performance vs. security trade-off.

The contributions of this paper are: *symmetric encryption techniques* for greater-than and range queries based on Bloom filters; a *system implementation* of privacy-preserving event correlation based on MXQuery [21]; and a systematic *performance evaluation* of symmetric encryption techniques allowing equality tests, range queries, and blind addition.

The remainder of the paper is structured as follows: In Section II we describe our system and security model. In Section III we review our encryption techniques. Our performance evaluation results are shown in Section IV. We review related work in Section V and present our conclusions in Section VI.

II. System Architecture

System Model The main components of our privacy-preserving, distributed event-based system are event sources and an event processor. Event sources are commonly sensors, e.g. in the form of service wrappers around common network services. They emit raw (or primitive) events, e.g. in the CBE format [9], when they observe noteworthy behavior of the service, e.g. a TCP SYN packet addressed to a closed port. The event processor is a complex event processing (CEP) engine, e.g. an XQuery engine [21], which discovers complex data patterns from event streams, e.g. a distributed port scanning attack. For simplicity, we consider only a single event processor in the remainder of the paper. Nevertheless, our techniques can be straightforwardly extended to multiple, distributed event processors.

The events are sent from the event sources to the event processor using an event bus. In the simplest case, the bus is comprised of direct connections, but a complex routing infrastructure using publish-subscribe techniques is also feasible [11]. The event sources encrypt the payload of their events before releasing them. We describe the encryption methods we consider in this paper in Section III. This encryption is also done with respect to the processing necessary during the query. For example, if the query needs to perform a greater-than comparison, then the payload is encrypted using the encryption methods that allow greater-than comparison. This means, we assume that the event sources are aware of the queries performed by the event processor, or at least of the processing necessary for those queries.

The event processor correlates and aggregates (encrypted) raw events to produce complex ones with a higher abstraction level. This encompasses comparing payload of events from different sources or comparing payload to constants. The first comparison requires the events to be encrypted under the same key and the second requires the ciphertexts of the constants.

Before event distribution and processing starts, there is a setup phase. During which the cryptographic keys are distributed to the event sources and the ciphertexts of the constants are given to the event processor. Let k be the key for encryption. Note that if we can partition the event sources, such that no cross-comparisons are necessary, we can increase the number of cryptographic keys. Such a partition is application-dependent, but straightforward.

Security Model Although we do not use the techniques for secure computation for performance reasons, our attacker model reflects that of secure computation [16]. We provide a number of inputs from the event sources to the event processor. As in secure computation, these inputs should not be revealed (to the event processor). Nevertheless, the event processor should be able to perform queries over the event stream.

We assume that no event source colludes with the event processor. This assumption is necessitated by the cryptographic keys. Since we use symmetric encryption, any party possessing the key has access to all inputs (events), but also the key is necessary in order to produce encrypted inputs. We therefore assume that each event source holds the same key material. We accept this limitation in favor of the performance symmetric encryption offers.

Yet, because we use encryption-based techniques we do not need to distinguish between passive and active attacks. In secure computation there are passive – semi-honest or honest, but curious – adversaries and active – malicious – adversaries [16]. A semi-honest adversary follows the protocol description, but tries to passively break confidentiality. A protocol secure against malicious adversaries also ensures the integrity of the computation.

When comparing semi-honest security in secure computation to our security guarantees, we need to define an ideal functionality implemented by the protocol. Our ideal functionality corresponds to the complete application of the operators enabled by the encryption. The event processor can compare any event to any other event. Secure computation allows the restriction of the number of comparisons or even their sequence. Of course, this security benefit comes at the performance cost of "one-time" encryption (i.e. inputs cannot be reused).

Confidentiality and integrity during transmission can be ensured using secure and authenticated channels. Note that this may require doubly-encrypting the event: Once for protection against the adversary on the communication link and once for the protection against the event processor.

III. Encryption Techniques

Let e_i be an event emitted by an event source which has obtained a key k. Let $\mathsf{e}_i(x)$ denote the data, such as a value or string, assigned to a variable x (specific field) within an event. We then protect the privacy of data $\mathsf{e}_i(x)$ using an encryption method $\pi_k(\cdot)$, e.g. a cryptographic hash function and encryption algorithm with key k.

The event sources subsequently should further protect the encrypted data $\pi_k(\mathsf{e}_i(x))$ using a MAC scheme and an efficient symmetric encryption scheme (see for example [8]). This implements the necessary secure and authentic channels. Upon receiving an event from the event source, the event processor recovers $\pi_k(\mathsf{e}_i(x))$ which is then used to process queries.

It can be very challenging (if not impossible) to perform meaningful correlation on encrypted data, particularly in an *efficient* manner. We consider four types of privacy-preserving data correlation techniques as part of the query:

- *Equality test* (or exact matching), which checks if two keyed hash values (of values/strings) are equal;
- *Greater-than comparison*, which makes use of an order-preserving technique to check if one encrypted numeric

data is greater than another;
- *Range query*, which checks if a received, encrypted value falls within a pre-defined range;
- *Blind addition*, which is based on homomorphic encryption and is used to aggregate two encrypted values without revealing the actual values.

We now describe how the aforementioned correlation techniques are used by the event processor on encrypted data $\pi_k(\mathsf{e}_i(x))$ that it receives.

Equality Tests We use keyed-hashes for performing equality tests. Using this approach, we employ a keyed hash function $H_k(\cdot)$, such as HMAC [3], as the encryption function, that is $\pi_k(\cdot) = H_k(\cdot)$.

The use of a keyed hash function guarantees privacy of sensitive fields, since an adversary cannot verify its guesses without knowing the key used for hashing. This approach can be very useful for encrypting sources of events, such as user IDs. Moreover, this is a very efficient way of achieving encrypted data correlation, because the cost of comparing two hash values is literally similar to that of two plaintexts.

Greater-Than Comparisons We can also represent a numeric data, such as age, price and volume, in the form of a Bloom filter [6], particularly in an order-preserving manner.

In a nutshell, a Bloom filter for representing a set $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$ of n elements is described by a bit array of l bits. The Bloom filter is initially all set to 0. It uses m independent hash functions $h_1, \ldots, h_m$ all with range $\{1, \ldots, l\}$. For each element $x_i \in \mathcal{S}$ for $1 \leq i \leq n$, the bits $h_j(x_i)$ are then set to 1 for $1 \leq j \leq m$. A location can be set to 1 multiple times, but only the first change has an effect. To check if an item y is in $\mathcal{S}$, we check whether all $h_j(y)$ are set to 1. If not, then clearly $y \notin \mathcal{S}$. If all $h_j(y)$ are set to 1, we assume that y is in $\mathcal{S}$, although we could be wrong with some probability.

We now explain how Bloom filters can represent numeric data in the form of *partially ordered sets* (or *posets*), and thus enable greater-than comparisons. A similar technique was proposed by Lee *et al.* [18], although no implementation results were available.

A poset consists of a set together with a binary relation that describes, for certain pairs of elements in the set, the requirement that one of the elements must precede the other. Let $\mathsf{e}_i(x)$ be a data element of a poset $\mathcal{P}$. To insert p into a Bloom filter $\mathcal{B}$, the element p is mapped to some bit positions of the Bloom filter through some hash functions. The Bloom filter $\mathcal{B}$ can then be made order-preserving by inserting all other elements in $\mathcal{P}$ that are smaller than p. Clearly, $\mathcal{B}$ representing p is greater than $\mathcal{B}'$ representing another element $p' \in \mathcal{P}$ where $p > p'$, if $\mathcal{B}$ contains both elements p and p'. This way, the order of $\mathcal{B}$ and $\mathcal{B}'$ is preserved and one can tell that one is greater than other without knowing the original data elements, and thus this is useful for privacy-preserving data correlation.

Note that in our case, we require that our encryption method $\pi_k(\cdot)$ comprises m independent keyed-hash functions $H_k^1, \ldots, H_k^m$ instead of hash functions $h_1, \ldots, h_m$, as described above, when constructing a Bloom filter. This is needed to prevent dictionary attacks on data $\mathsf{e}_i(x)$. The output of $\pi_k(\cdot)$ is then a Bloom filter of the form $H_k^1 \cup H_k^2, \ldots, \cup, H_k^m$.

To avoid confusion, we name the standard (non-order-preserving) Bloom filters *singular* bloom filters, while the order-preserving type *cumulative* Bloom filters. Each event sources encrypts payload data through a cumulative Bloom filter. This allows the event processor to simply compare two cumulative Bloom filters and check if one is greater than the other. Similarly, we can also compare directly a cumulative Bloom filter from an event source against a constant pre-distributed to the event processor.

While the use of cumulative Bloom filters is an interesting approach, we note that the cost of comparing two Bloom filters can increase quickly as the input range increases. We will further discuss this in Section IV.

Range Queries Singular Bloom filters are also useful for range queries, *i.e.* membership tests. The event processor is given a cumulative Bloom filter containing a range of input. An event source can then generate a singular Bloom filter based on some payload data and forward it to the event processor, which in turn, tests if the singular Bloom filter is a member of the cumulative Bloom filter. From the event source's view point, we envisage that singular Bloom filters (corresponding to different possible data values $\mathsf{e}_i(x)$) can be pre-computed to optimize the associated computational cost.

Blind Addition We adopt the homomorphic, symmetric encryption scheme $\mathcal{E}(\cdot)$ by Castelluccia et al. [8] in order to allow addition of two or more encrypted numeric data (integers). In our approach, an event source encrypts events that it generates using a homomorphic encryption scheme and its key k. Hence, we define our encryption function as:

$$\pi_k(\mathsf{e}_i(x)) = \mathcal{E}_k(\mathsf{e}_i(x)) = \\ [\mathsf{e}_i(x) + h(f_k(r)) \mod N; r]$$

where f is a pseudo-random function, h denotes a length-matching hash function, r represents a random number and N is a sufficiently large modulus. The event processor can then simply perform addition on two encrypted numeric data by directly adding the corresponding ciphertexts.

$$\mathcal{E}_k(\mathsf{e}_i(x)) + \mathcal{E}_k(\mathsf{e}_j(y)) = \\ [\mathsf{e}_i(x) + \mathsf{e}_j(y) + h(f_k(r_i)) + h(f_k(r_j)); r_i, r_j]$$

We chose the scheme of [8] for its efficiency. It is based on only modular addition and the cost of aggregating ciphertexts is comparable to using keyed-hash functions.

IV. Performance Evaluation

In this section, we discuss our implementation and empirical results of the previously described encryption techniques.

We simulated our experiments on a machine equipped with a Pentium IV 2.80 GHz processor and 1 GB memory, running on Windows XP Professional SP 2. The development environment is based on Java JDK & JRE 1.6.15 and XMLBeans 2.4.0, the crypto library is based on the JCE implementation, and the CEP engine is based on MXQuery 0.6.0.

We performed two types of experiments. The first type is designed for measuring the computational cost of event generation by each event source. More accurately, we measure the time it takes for an event source to perform encryption of data and to create a raw event in CBE format. The second type of experiment is used to evaluate the computational cost of the event processor (CEP engine). To do this, we measure the processing time required by the event processor to create a query window for incoming event streams and to evaluate query conditions. For simplicity, we only consider the creation of one window, although the CEP engine supports multi-windows for parallel processing. Moreover, since we are not concerned with the window time (the interval between when a window is opened and when it is closed), we feed the CEP engine with the necessary event stream at once into a single window and perform queries on the received events.

The afore-mentioned two types of experiments are run using n number of events, where $n \in \{1, 5, 10, 50, 100, 500, 1000\}$. For each choice of n, we perform 100 iterations and calculate the average cost. Note that the choices of n seem to be sufficient to measure the efficiency and scalability of the CEP engine using a single query window, which is typically associated with processing events emitted by instances of the same complex event.

We use HMAC-SHA1 from JCE as the keyed hash function in our experiments. Bloom filters are implemented using the `java.math.BigInteger` class and are based on four HMAC-SHA1 functions. We use [1,100] and [1,1000] as the input ranges for the Bloom filters. We allow compression to be performed on the Bloom filters, using the JDK implementation of GZip, to optimize the bandwidth requirement. For homomorphic encryption, we set the required modulus (as described in Section III) to 10^8. Our seeded pseudo-random number generator is based on the `java.security.SecureRandom` class. We also implemented 128-bit AES-CBC symmetric encryption for comparison purposes.

Computational Cost of Raw Event Generation Our empirical results regarding the computational costs for raw event generation are shown in Figure 1a and 1b. Note that all charts are plotted on logarithmic scale. It shows the computational costs for raw event generation that makes use of a keyed-hash function and a homomorphic encryption scheme, Figure 1a, in comparison to using a standard AES encryption scheme and not applying any further encryption techniques at all (plaintext). It is clear that keyed-hash generation and homomorphic encryption are relatively lightweight. They are significantly more efficient than AES encryption.

On the other hand, the costs for raw event generation which makes use of compressed and uncompressed Bloom filters are summarized in Figure 1b. We consider the cases when Bloom filters are compressed (with compression rate of more than 90%) before being incorporated into events. We can see from the figure that the cost of generating events containing compressed Bloom filters (both singular and cumulative) can be reduced significantly compared to those which contain uncompressed Bloom filters. However, the computational cost is reduced at the expense of increased computational cost at the event processor, *i.e.* cost of decompression.

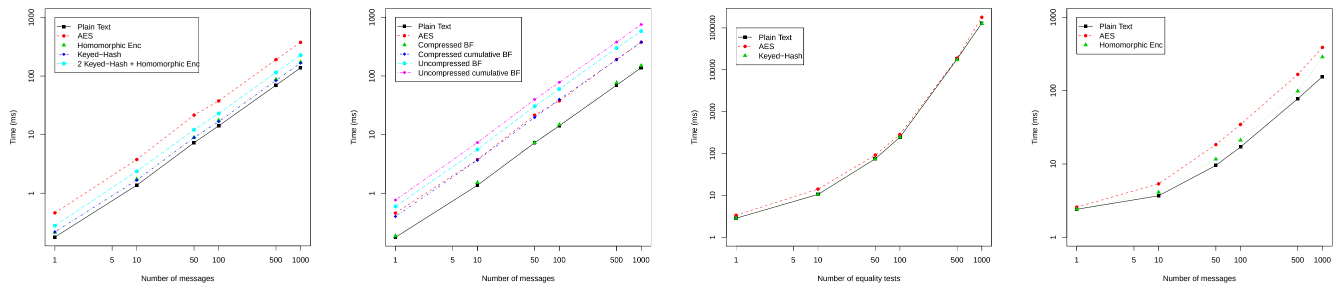
Privacy-Preserving Correlation The main interest of our performance benchmarks rely on measuring the computational cost of privacy-preserving correlation. Recall that our goal is to directly correlate encrypted events without recovering the original data (or plaintext). In what follows, our results show that such an approach can be indeed more lightweight than the AES 'decrypt-then-compare' approach (decrypt encrypted values and then compare the corresponding plaintexts).

From Figure 1c, we see that comparing hash values is as efficient as comparing plaintexts, while more efficient than AES decrypt-then-compare. Moreover, we note that the difference between keyed-hashes, AES decryption-and-compare, and plaintext (in terms of computational cost) become smaller as the number of equality checks increases. This seems to imply that the cost of preserving data privacy is relatively small compared to the associated query processing time.

In a separate experiment, we measure the computational cost of aggregating encrypted data directly as compared with aggregating plaintext and data encrypted with AES (by recovering the associated plaintext). Our results in Figure 1d show that the use of a homomorphic encryption for preserving data privacy is still significantly more efficient than using the AES decrypt-then-compare approach.

When Bloom filters are used, our results also show that the associated computational cost is slightly lower than using the AES decrypt-then-compare approach. However, we observe that this is true only for input ranges on the order of hundreds. The computational cost increases quickly when the input range increases beyond that. We believe that this is due to the cost of parsing Bloom filters through the CEP engine (query processing).

Computational Cost of Correlation Based on Bloom Filters In terms of privacy-preservation using Bloom filters for range queries and greater-than comparisons, our experiments show that processing compressed Bloom filters is significantly more costly than uncompressed ones, which in turn offers a processing speed comparable to the speed of



(a) Keyed-hashes and homomorphic encryption (b) Singular/cumulative Bloom filters (c) Equality tests using keyed-hashes (d) Blind addition using homomorphic encryption

Figure 1: Performance evaluation results

comparing plain-text when dealing with small Bloom filter sizes, meaning shorter input ranges. A second observation is that the time it takes to correlate Bloom filters, *i.e.* range queries and greater-than comparison, can increase significantly as the input range increases. This suggests that if the input range is large, then Bloom filters may not be a suitable choice for data privacy-preservation. Otherwise, this approach can be more efficient than or comparable with using AES.

V. Related Work

To our knowledge, little work has been done on privacy-preserving, distributed event-based systems. One closely related domain is privacy-preserving alert (or event) correlation for intrusion detection systems, for example proposals by Lee et al. [18], Lincoln et al. [19] and Parekh [23]. In these proposals, security alerts generated by different users or organizations are shared with each other in order to detect attack patterns or malicious activities. The motivation for preserving privacy of alerts is to hide information such as network topology, open services and source addresses of an organization. The techniques presented in these papers follow the same architecture and key distribution as ours, although they often lack the necessary focus on efficiency. We provide performance evaluation for the techniques with the most promising performance characteristics.

The SEPIA library [7] implements equality, short range, less than comparison operations for privacy preserving correlation on aggregated network data using multi-party computations, based on a classical secret sharing scheme. Extensive performance benchmarks were conducted in [7] showing the robustness of the approach in the network event analysis domain. We advocate that the algorithms we analyse in this paper also serve the analysis of business data, wihtout the burden of extra communication required by multi-party computations.

Agrawal *et al.* [1] introduced the notion of order-preserving encryption (OPE) to allow efficient queries on encrypted databases. They proposed an OPE scheme that enables a user to store her data encrypted while preserving the order of the ciphertexts according to the order of the corresponding plaintexts. This has been extended to an encryption scheme that provides the best possible protection for an order-preserving encryption scheme in [5]. While this is a very useful concept, we explore an alternative approach using Bloom filters (for range queries and greater-than comparisons) because of intellectual property issues.

Another related research domain is privacy-preserving data aggregation in wireless sensor networks [8], [17]. Here, the goal is to protect privacy of data collected by sensor nodes while allowing the data from different nodes to be efficiently aggregated (by any neighbor nodes) without revealing the original data. Nevertheless, the existing work focuses on summation of numeric data. Other types of data correlation required for complex event processing, such as range queries, have not been considered.

Data anonymization (or sanitization) techniques, such as k-anonymization [25], can be used to generalize sensitive data attributes and introduce uncertainty into datasets, while allowing correlation on anonymized data. While this approach has been shown to be very useful for protecting privacy of databases, it is less clear how it can be applicable to real-time data streams. Furthermore, it can be challenging to compare generalized data values with high accuracy, *i.e.* two similar sanitized values may correspond to different original values.

There are many other proposals that are related but not very applicable to our work. For example, secure two-party [27] or multi-party computation [16], homomorphic encryption [14], [22] or privacy-preserving set operations [13] can be used to securely perform join computations between two or more entities (or participants) without revealing their data. Although theoretically attractive, these approaches are generally considered too computationally expensive to handle large data streams.

VI. Conclusions and Future Work

We examined the performance of several privacy-preserving data correlation techniques based on symmetric encryption that seem suitable for efficient, distributed event-based systems. Our empirical results show that privacy-preserving data correlation that makes use of a keyed hash function and a homomorphic symmetric encryption scheme is sufficiently efficient, comparable to correlating raw data or plaintexts. On the other hand, although Bloom filters enable reasonably efficient range queries and greater-than comparisons over encrypted data, the correlation cost incurred by the event processor can increase very quickly as the chosen size of Bloom filters increases. This implies that Bloom filters are useful when the required input range is small, for example of the order of hundreds or less.

Nevertheless, there are open problems which require further investigation. One immediate issue to be addressed is the efficiency of privacy-preserving data correlation associated with range queries and greater-than comparisons. Moreover, we should study how to optimize the key distribution and event generation for different types of queries.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2004.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. *Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS)*, 2002.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Advances in Cryptology – Proceedings of CRYPTO*, 1996.

[4] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communication of the ACM 13(7)*, pp. 422–426, 1970.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. *Advances in Cryptology – Proceedings of EUROCRYPT*, 2009.

[6] A.Z. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics 1(4)*, 2003.

[7] M. Burkhart, M. Strasser, D Many, X.A. Dimitropoulos: SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. *Proceedings of the 19th USENIX Security Symposium*, 2010.

[8] C. Castelluccia, A.C-F. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Transactions on Sensor Networks 5(3)*, 2009.

[9] Common Base Event. http://www.ibm.com/developerworks/library/specification/ws-cbe/.

[10] Data Protection Act 1998 (DPA). http://www.opsi.gov.uk/acts/acts1998/ukpga_19980029_en_1.

[11] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys 35(2)*, 2003.

[12] Fair Credit Reporting Act (FCRA), Public Law No. 91-508, 1970. http://www.ftc.gov/os/statutes/031224fcra.pdf.

[13] M.J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. *Advances in Cryptology - Proceedings of EUROCRYPT*, 2004.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. *Proceedings of the 41st ACM Symposium on Theory of Computing*, 2009.

[15] C. Gentry and S. Halevi. Implementing Gentrys fully-homomorphic encryption scheme. *Advances in Cryptology - Proceedings of EUROCRYPT*, 2011.

[16] O. Goldreich. Secure multi-party computation, 2002. http://www.wisdom.weizmann.ac.il/~oded/pp.html.

[17] W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T.T. Abdelzaher. PDA: Privacy-preserving data aggregation in wireless sensor networks. *Proceedings of the 26th IEEE Conference on Computer Communications (INFOCOM)*, 2007.

[18] A.J. Lee, P. Tabriz, and N. Borisov. A privacy-preserving interdomain audit framework. *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2006.

[19] P. Lincoln, P.A. Porras, and V. Shmatikov. Privacy-preserving sharing and correlation of security alerts. *Proceedings of the 13th USENIX Security Symposium*, 2004.

[20] D. Luckham. The power of events: an introduction to complex event processing in distributed enterprise systems. *Addison-Wesley*, 2002.

[21] MXQuery: A Lightweight, Full-featured XQuery Engine. http://mxquery.org/.

[22] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Advances in Cryptology - Proceedings of EUROCRYPT*, 1999.

[23] J.J. Parekh. Privacy-Preserving Distributed Event Corroboration. *Ph.D thesis, Columbia University*, 2007.

[24] The Privacy Rule of Health Insurance Portability and Accountability Act of 1996 (HIPAA). http://www.hhs.gov/ocr/hipaa/finalreg.html.

[25] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(7)*, 2002.

[26] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2006.

[27] A.C. Yao. Protocols for secure computations. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, 1982.