A Privacy Framework within the Java Data Security Framework (JDSF): Design Refinement, Implementation, and Statistics

Serguei A. Mokhov, Lee Wei Huynh, Jian Li, Farid Rassai
1455 De Maisonneuve Blvd. W.
Concordia Institute for Information Systems Engineering
Concordia University, Montreal, Quebec, Canada, H3G 1M8
{mokhov,lw_huynh,j_lix48,f_rassai}@ciise.concordia.ca

ABSTRACT

We present a refinement design of something we call a *Confidentiality Framework*, which is a part of a more general formation, that we refer to as a Java Data Security Framework (JDSF), a work-in-progress designed to support various aspects that are related to data security (confidentiality, origin authentication, integrity, etc. where this paper only focuses on the confidentiality aspect). The design refinement considerations include further unification of the parameter structure of concrete modules of the framework as well as the design and implementation of the statistics gathering module for comparison of the implementing methods in the framework and the impact of the statistics collector itself in terms of run-time performance and memory consumption in the example use-cases.

Keywords: Java Data Security Framework (JDSF), Modular Audio Recognition Framework (MARF), HSQLDB, data confidentiality, software engineering design, frameworks, Java

1. INTRODUCTION

In this work we use the words "privacy" and "confidentiality" interchangeably in the reference to the sub-framework's name and the concept.

Motivation

In [11] a Java Data Security Framework (JDSF) was designed for use in the two use-cases, HSQLDB [18] and MARF [19, 4, 5, 7, 6, 8] to allow a plug-in-like implementation of various security aspects, the first of which is the confidentiality of the data. Defining aspects, such as regular encryption, encrypted search, k-anonymity, l-diversity, k-uncertainty, and indistinguishability were considered in the earlier works of Song, Wagner, Wang, Perrig, Sweeney, Jajodia, and others [15, 17, 23, 24, 22] in order to extract a spectrum of possible parameters these aspects require for the design of an extensible frameworked API and its evolution. A particular challenge is an aggregation of diverse approaches and algorithms into a common set of Java interfaces, classes, their methods and related data structures to cover all or at least most common aspects, and at the same time keeping the framework as simple as possible and as general as possible.

Proposed Solution

This paper presents a refinement of the privacy aspect's design as well as the design consideration for performance gathering statistics module in part to apply the framework in the MARF's speaker identification [12] application's database (gathering statistical data on performance and accuracy of the algorithms does not require real identities of speakers to be known), and in a more general purpose Java SQL database engine, HSQLDB.

Background

In this section we briefly review the basic related work on the technologies used for refinement and application of the research described in this work.

JDSF: The Java Data Security Framework (JDSF) [11, 13] was designed as a part of a project on database privacy and security. The design goal of JDSF is to allow a plug-in-like replaceable architecture allowing addition and swapping different algorithm implementations for the researchers to test, verify, validate, and compare using a common API. The framework's design and API consider four major aspects when concerned with data security in Java for different research works, which are realized as frameworks on their own: data confidentiality (privacy) [15, 17, 23, 24], data integrity, data origin authentication, and SQL randomization for SQL-based databases as well as their supporting cryptographic algorithms and protocols required to achieve the stated goals. In [11], JDSF was designed for the use in the two use-cases, specifically HSQLDB [18] and MARF [19, 5, 7, 6, 8, 10] in order to extract the most complete list of parameters these four aspects require for the design its implementation to be flexible and extensible. JDSF relies in part on MARF, which by itself is also one of the case studies, so the JDSF is evolving within MARF's CVS repository and project structure (under a separate code branch) at present. Thus, most of the package naming and general naming conventions come from MARF. In fact, the JDSF framework's design is built upon the MARF's successful approach.

MARF: The Modular Audio Recognition Framework [4, 19, 5, 7, 6, 8] is an open-source research platform and a collection of audio and natural language processing (NLP) algorithms written in Java. It is arranged into a modular and extensible framework facilitating addition of new algorithm implementations for pattern recognition and beyond. MARF's based applications can run distributively [3] over the network (using CORBA, XML-RPC, or Java RMI) and its implementation may act as a library in the applications. One of MARF's applications, `SpeakerIdentApp` [12] has a database of speakers, where it can identify who people are regardless what they say.

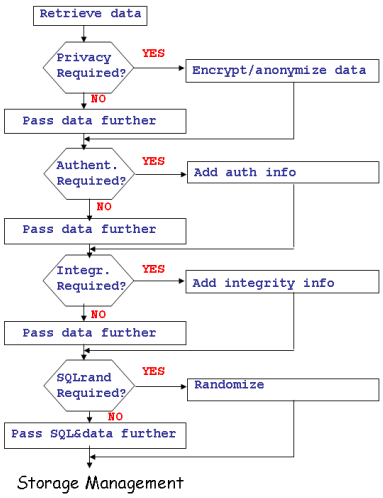

Figure 1: Writing Data With Security Options.

HSQLDB: HSQLDB is a popular open-source SQL relational database engine, as MARF and JDSF written in Java. It has a JDBC driver and supports a large subset of ANSI-92 SQL and SQL 99 and 2003 enhancements. It provides a small and relatively fast database engine, which offers both in-memory and disk-based tables and supports embedded and server modes. Additionally, it includes tools such as a minimal web server, in-memory query and management tools. HSQLDB is currently being used as a database and persistence engine in many Open Source Software projects (e.g. OpenOffice) and even in commercial projects. It is generally known for its small size, ability to execute completely in memory, flexibility, and speed.

2. METHODOLOGY

Java Objects/Beans Serialization

Security of the serialization process of Java objects and beans is where the JDSF security mechanisms kick in. In Figure 1 is a general way the framework's particular adapters (e.g. for MARF and HSQLDB) write the security-enhanced data based on the security configuration options, set by the system administrator. The reading of the security-data is the reverse process. The privacy aspect is the first one on the chain (though this can be altered by the appropriate configuration options).

API Design

Here we summarize the API that is either directly part of the Confidentiality Framework or is needed as a part of the supporting modules that we designed in the JDSF.

- MARF augmented with database security packages is illustrated in Figure 2. These packages roughly follow the same structure as the MARF itself as mentioned earlier – this is where the JDSF evolves in.
- Security-related configuration classes are shown in Figure 3.
- In Figure 5 is a UML class diagram of the main interface `IConfidentialityModule`, that is located in package `marf.security.confidentiality` that all concrete confidentiality modules have to adhere to.
- The interface `ISecurityEnhancedObject`, found in the package `marf.security.Storage` is a bean with security payload, as shown in Figure 4.
- All the modules that provide an independent security algorithm implementation, should adhere to the specification of the top level `IAlgorithmProvider` interface, whose definition is found in `marf.security.algorithms` as shown in Figure 6. The providers typically include for example cryptographic algorithm implementation by whatever vendor (that can be used by the Confidentiality framework depicted here or the Integrity framework, that is not part of this publication).
- Finally, in order to make most of the framework's modules available to our use-cases, there has to be an adaptation layer that transforms the data structures without security to more secure ones and back adapted to a particular software that manages any sort of serializeable data. It is depicted by the `ISecurityAdapter` interface found in the `marf.security.adapters` package, as shown in Figure 7.

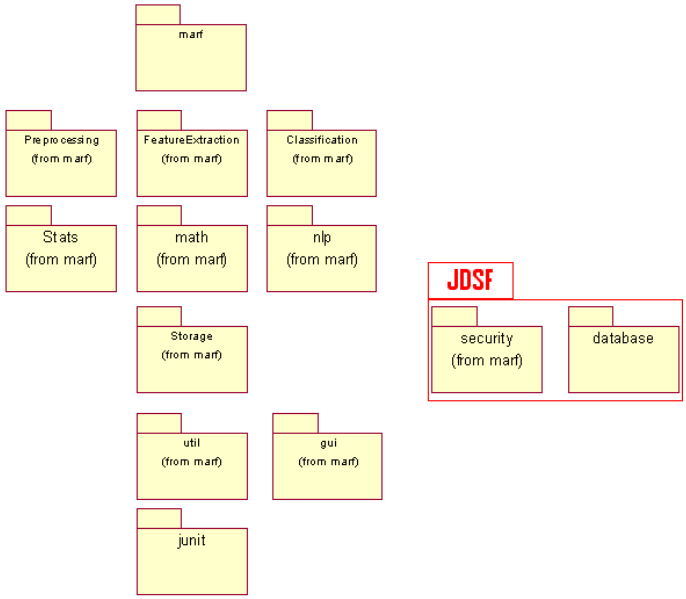

Figure 2: MARF Augmented with the JDSF Packages.

In the `marf.security.algorithms` package there are implementations of well known cryptographic algorithms, such as CBC-DES, RSA, DSA, MD5, and SHA1. The actual implementations in Java were provided by various open-source vendors, such as [21, 2, 20, 16, 14, 1]. Since these implementations have sometimes little in common, integrating them into the framework has to be abstracted by a common API of algorithm providers (as in Figure 6), so the rest of the framework does not depend on the vendors' API and can be replaced to use another implementation easier when desired.

The most complexity goes into the implementation and integration of the framework into the actual data management tools, such as MARF and HSQLDB. For this we provide their specific adapters

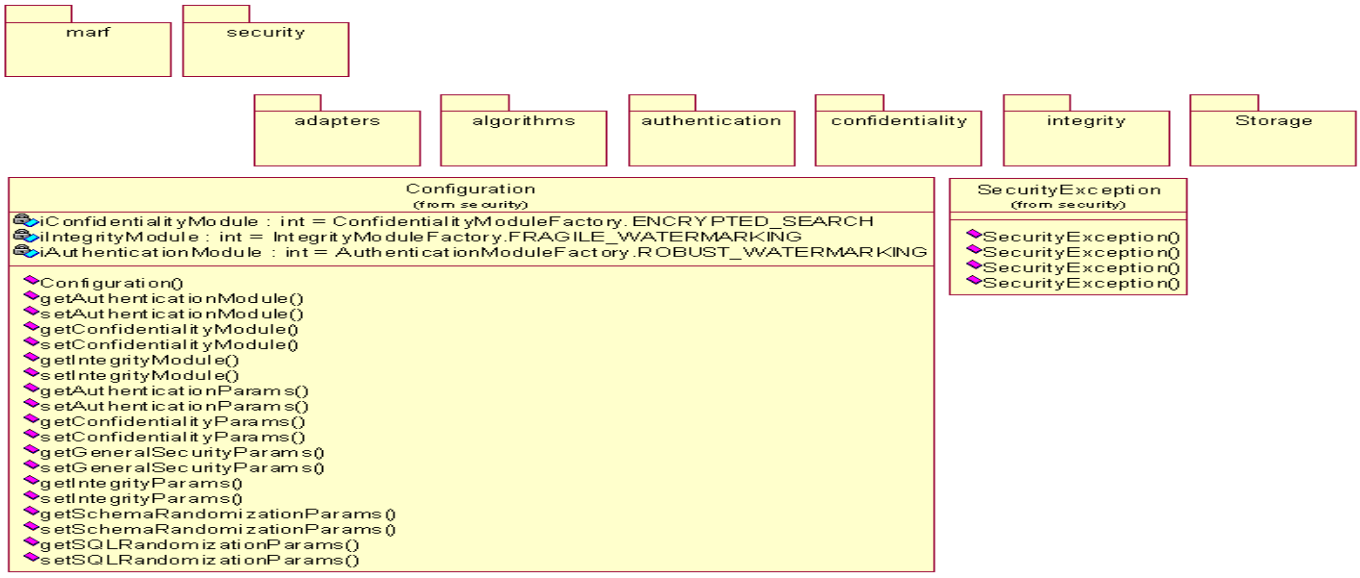

Figure 3: `marf.security` Package.

(see Figure 7). MARF gets its security adapter class implemented in `marf.security.adapters.MARFSecurityAdapter`, which extends MARF-specific storage management and, likewise, `marf.security.adapters.HSQLDBSecurityAdapter` class for HSQLDB, which are there to be "injected" into the original code wrapping storage management functions of the original tools to mandatory go through the security-enhanced API. The replaced and/or extended modules with the security mechanisms exactly are `marf.Storage.StorageManager` for MARF and `org.hsqldb.persist.Log` for HSQLDB.

Parameters Summary

Here we summarize various confidentiality parameters gathered so far during our research.

1. Data object to anonymize (encrypt).
2. Encryption key(s) and their size.
3. Encryption algorithm type (CBC-DES, RSA, DSA).
4. Hashing algorithms for HMAC (SHA1, MD5) for encrypted search.
5. n is the size of the search word, and m is the size of the right portion in bits that corresponds to the encrypted portion of W_i on the right R_i and the same size as $F_{k_i}(S_i)$.
6. k – an integer, how many records should appear similar at a minimum, ≥ 2. or how many association there may be for the k-uncertainty, or the parameter for undistinguishability k-SIND.
7. Confidentiality algorithm type (k-anonymity, l-diversity, or k-uncertainty) with ability to chain the algorithm or set any combination of them depending on the desired policy.
8. l – an integer for l-diversity.

3. PERFORMANCE/OVERHEAD ANALYSIS AND STATISTICS GATHERING DESIGN

Adding any security mechanism, whether it is originally designed within a system or as an add-on for an existing system is a source of an overhead that may hurt performance. The overhead may be insignificant compared to the accuracy of results, such as for example distributed scientific computations over the Internet as done for example for Distributed MARF (DMARF) [3, 9], but may be more important in the general user-interactive database-driven applications as well as applications that do a bulk of multimedia processing such as for law enforcement. Thus, we present an analysis of the minimal and per-algorithm overhead JDSF brings as a security layer in the existing tools in terms of time and space requirements.

The basic overhead o_b stems from extra method calls and object wrappings/instantiations that need to take place when a JDSF instance is injected into say MARF's storage operations. To measure the basic overhead, we create a `Dummy` instance of every algorithm type that acts as a "pass-through" module for data, while maintaining the minimal number of method calls and object instantiations JDSF requires. o_b is JDSF-dependent.

The algorithm overhead o_a, is the overhead imposed by each specific security or otherwise algorithm implementation and execution by the implementation's vendor, that are deemed to be "external" to JDSF and may be seen as "black boxes". This overhead depends on the mathematical complexity of the algorithm as well as vendors' implementation choices over data structures and optimizations they may have used in the implementation. Thus, the total overhead stems from the basic and algorithm implementation overheads, where the former is deemed to be a constant penalty for any algorithm implementation, $o_t = o_b + o_a$.

The design now incorporate the automatic statistic collection and management as a configuration flag. The statistics capture includes individual run-time of unit calls as well as extra memory required

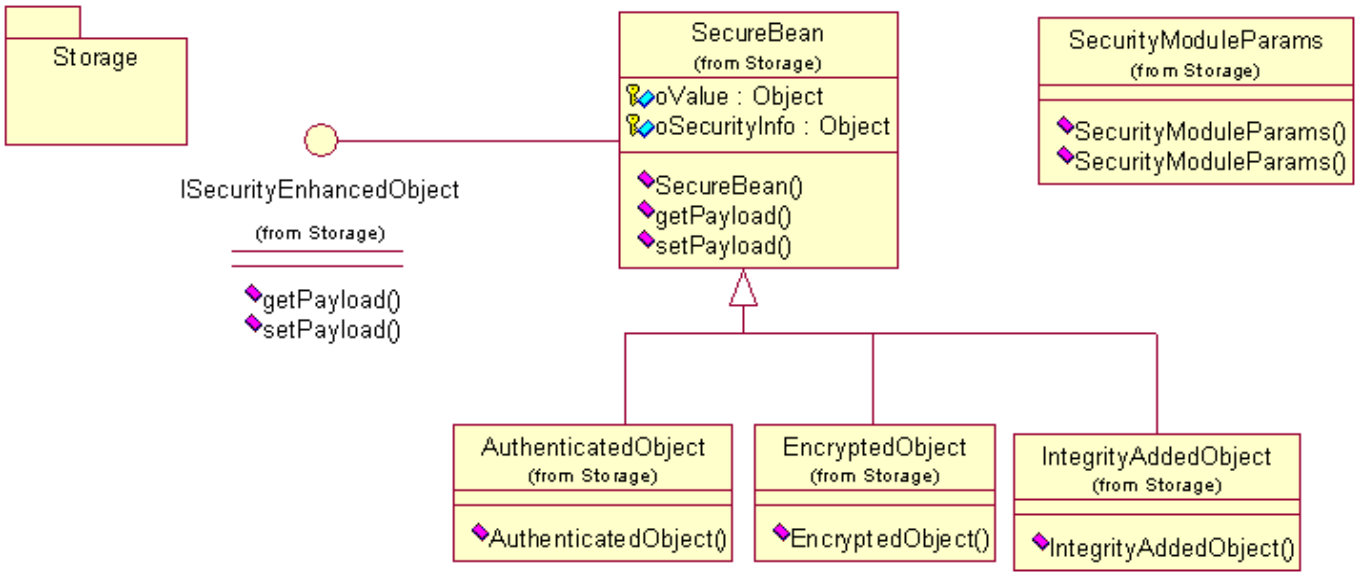

Figure 4: `marf.security.Storage` Package and Classes.

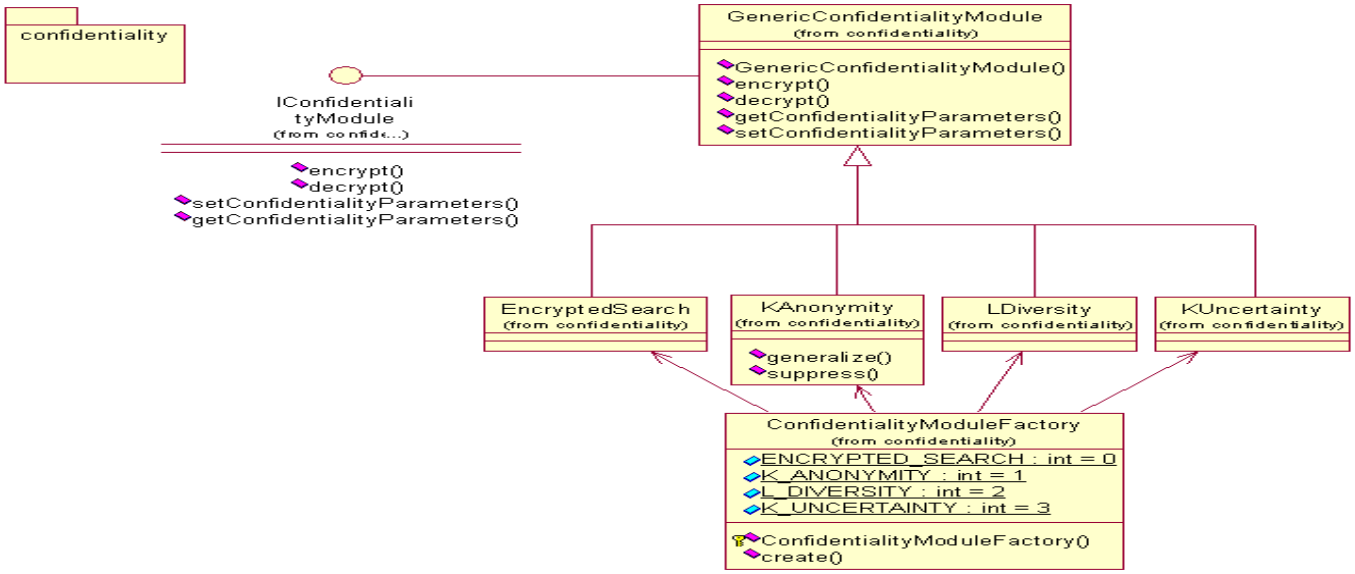

Figure 5: `marf.security.confidentiality` Package and Classes.

to store security-enhanced beans. Understandably, the statistics gathering itself induces overhead, both run-time and memory consumption. This may impact the overall turnaround time of the entire process but generally has very little impact on individual unit measurements. The turnaround time can be measured more precisely with the unit measurements turned off. This is especially pertinent when measuring memory consumption overhead, which is time costly: writing out serialized objects into byte arrays and comparing their sizes before and after application of the confidentiality to it. Thus, the turnaround overhead also including statistics gathering overhead, o_s, that can be significant if both time and memory measurements are activated at the same time, $o_t = o_b + o_a + o_s$. In the current experiments of the confidentiality aspect we measure run-time and memory overhead of the two implementations of CSC-DES and RSA implementations that we have available for this task including multiple implementations by different vendors of the same algorithm. Based on the above analysis we refine our design with the statistics collector based on the Builder design pattern. We leave out the k-anonymity, l-diversity, and others from our experiments because their implementation is not finalized as of this writing and will be provided gradually in the next releases of JDSF.

4. CONCLUSION

We believe JDSF's Confidentiality Framework provides a general testbed for researches to run and compare data privacy related algorithms implemented in Java. We explored a few popular tech-

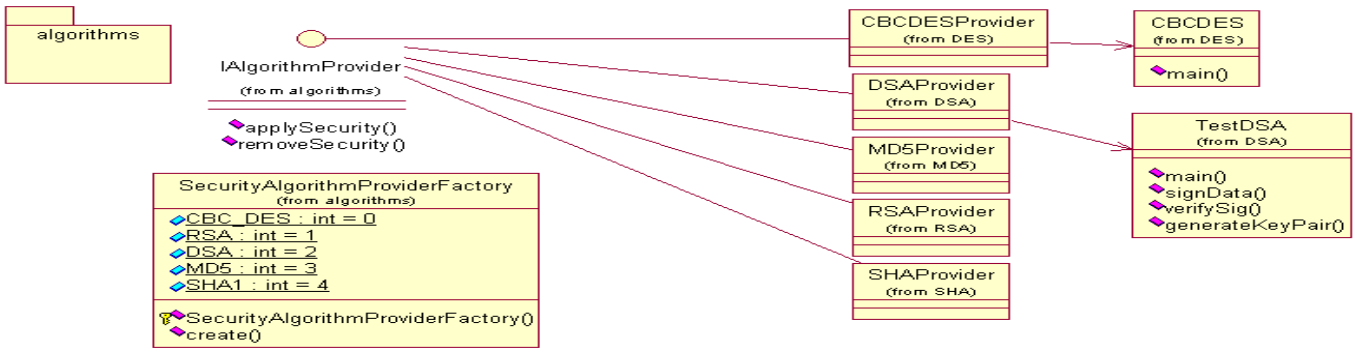

Figure 6: `marf.security.algorithms` Package and Classes.

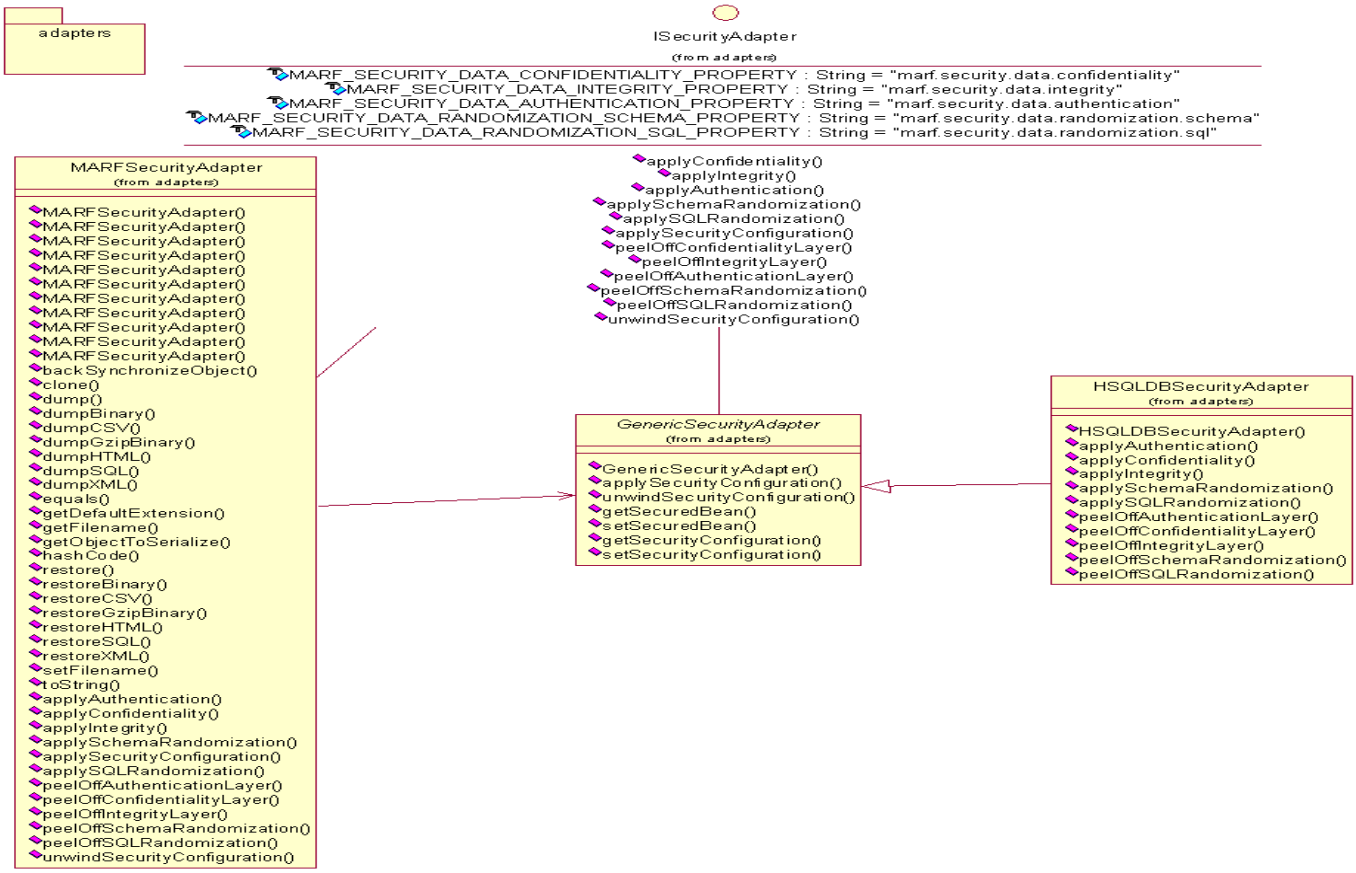

Figure 7: `marf.security.adapters` Package and Classes.

niques for encryption, encrypted search, k-anonymity, l-diversity, etc. We treat non-encryption anonymized data as encryption in our framework, i.e. we use the same API for both except in the latter case "encryption" means for example "generalization" or "suppression" (or even "compression"). In general, JDSF's operation was designed to allow addition of any number of algorithms or techniques to add as plug-ins for comparative study or when better techniques become available. The parameters and the configuration of the framework were made available from the survey/research study of the database security techniques mentioned earlier. It is also general enough to expand beyond MARF and HSQLDB, and as a result the open source community can benefit as a whole. JDSF, just like MARF and HSQLDB, is open source and is hosted at `SourceForge.net` under the umbrella of MARF, in its CVS repository. Please contact the primary maintainer, Serguei Mokhov, for the latest updates on JDSF.

Future Work

The future work will focus on the refinement of other aspects of JDSF as well as adding to our collection of implemented algorithms to augment the statistical knowledge of the actual algorithm implementations by the vendors.

As a further future work we plan on continuing our open-source development effort of the framework and fully integrating it into MARF and HSQLDB, along with comprehensive testing suite and overhead statistics and new algorithm implementations and porting it to other systems that require the features provided by the framework, as studied e.g. in [9].

Acknowledgements
We would like to thank Dr. Lingyu Wang in supervising the initial JDSF project. This research work was funded by the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada.

5. REFERENCES

[1] A. Andreu and M.-A. Laverdière. *SSHA Digest, Modified*. www.securitydocs.com, 2006. http://www.securitydocs.com/library/3439.

[2] J. O. Grabbe. *Java Program for RSA Encryption*. laynetworks.com, 2001. http://www.laynetworks.com/rsa_java.txt.

[3] S. Mokhov. On design and implementation of distributed modular audio recognition framework: Requirements and specification design document. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Aug. 2006. Project Report. A copy is found: http://marf.sf.net, last viewed April 2008.

[4] S. Mokhov, I. Clement, S. Sinclair, and D. Nicolacopoulos. Modular Audio Recognition Framework. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2002-2003. Project Report, http://marf.sf.net, last viewed April 2008.

[5] S. A. Mokhov. Introducing MARF: a modular audio recognition framework and its applications for scientific and software engineering research. In *Proceedings of the IEEE Engineering/Computing and Systems Research E-Conference (SCSS07/CIISE 2007)*, University of Bridgeport, U.S.A., Dec. 2007. Springer. To appear, http://cisse2007.org.

[6] S. A. Mokhov. Choosing best algorithm combinations for speech processing tasks in machine learning using MARF. In S. Bergler, editor, *Proceedings of the 21st Canadian AI'08*, pages 216–221, Windsor, Ontario, Canada, May 2008. Springer-Verlag, Berlin Heidelberg.

[7] S. A. Mokhov. Experimental results and statistics in the implementation of the modular audio recognition framework's API for text-independent speaker identification. In *Proceedings of the 6th International Conference on Computing, Communications and Control Technologies (CCCT'08)*, Orlando, Florida, USA, June 2008. To appear, http://www.infocybereng.org/ccct2008i.

[8] S. A. Mokhov. Study of best algorithm combinations for speech processing tasks in machine learning using median vs. mean clusters in MARF. In B. C. Desai, editor, *Proceedings of C3S2E'08*, pages 29–43, Montreal, Quebec, Canada, May 2008. ACM and BytePress. ISBN 978-1-60558-101-9.

[9] S. A. Mokhov. Towards security hardening of scientific distributed demand-driven and pipelined computing systems. In *Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC'08)*, Krakow, Poland, July 2008. IEEE Computer Society Press. To appear, http://ispdc2008.ipipan.waw.pl/.

[10] S. A. Mokhov. Towards Syntax and Semantics of Hierarchical Contexts in Multimedia Processing Applications using MARFL. In *Proceedings of the 1st IEEE International Workshop on Software Engineering for Context Aware Systems and Applications (SECASA 2008)*, Turku, Finland, July 2008. To appear, http://conferences.computer.org/compsac/2008/workshops/SECASA2008.html.

[11] S. A. Mokhov, L. W. Huynh, J. Li, and F. Rassai. A Java Data Security Framework (JDSF) for MARF and HSQLDB. Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, Apr. 2007. Project Report. Hosted at http://marf.sf.net, last viewed April 2008.

[12] S. A. Mokhov, S. Sinclair, I. Clement, D. Nicolacopoulos, and the MARF Research & Development Group. Text-Independent Speaker Identification Application. Published electronically within the MARF project, http://marf.sf.net, 2002-2008. Last viewed April 2008.

[13] S. A. Mokhov, L. Wang, and J. Li. Simple Dynamic Key Management in SQL Randomization. In *Submitted for publication at IDEAS'08*, 2008. https://confsys.encs.concordia.ca/ideas08/.

[14] S. Paavolainen and S. Ostermiller. *MD5 hash generator*. ostermiller.org, 2007. http://ostermiller.org/utils/MD5.java.html.

[15] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. University of California, Berkley.

[16] Sun Microsystems, Inc. *Security Features in Java SE*. java.sun.com, 2007. http://java.sun.com/docs/books/tutorial/security/index.html.

[17] L. Sweeney. k-anonymity: A model for protecting privacy. In *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, pages 557–570, 2002.

[18] The hsqldb Development Group. *HSQLDB – Lightweight 100% Java SQL Database Engine v.1.8.0.4*. hsqldb.org, 2006. http://hsqldb.org/.

[19] The MARF Research and Development Group. The Modular Audio Recognition Framework and its Applications. SourceForge.net, 2002-2008. http://marf.sf.net, last viewed April 2008.

[20] Unascribed. *Sign and Verify a DSA Signature*. java2s.com, 2004. http://www.java2s.com/Code/Java/Security/VerifyaDSAsignature.htm.

[21] Unascribed. *CBC-DES Java Implementation*. Unascribed, 2007.

[22] L. Wang. *INSE691A: Database Security and Privacy, Course Notes*. CIISE, Concordia University, 2007. http://users.encs.concordia.ca/~wang/INSE691A.html.

[23] L. Wang and S. Jajodia. *Security in Data Warehouses and OLAP Systems in The Handbook of Database Security: Applications and Trends*. Springer, Berlin; Michael Gertz, Sushil Jajodia, editors, 2007.

[24] L. Wang, S. Jajodia, and D. Wijesekera. *Preserving Privacy in On-line Analytical Processing (OLAP)*. Springer, Berlin, 2007. ISBN: 0-387-46273-2.