Crypto-Assistant: Towards Facilitating Developer's Encryption of Sensitive Data

Ricardo Rodriguez Garcia, Julie Thorpe, and Miguel Vargas Martin
Faculty of Business and Information Technology
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
Email: {ricardo.rodriguezgarcia, julie.thorpe, miguel.vargasmartin}@uoit.ca

***Abstract*—The lack of encryption of data at rest or in motion is one of the top 10 database vulnerabilities [1]. We suggest that this vulnerability could be prevented by encouraging developers to perform encryption-related tasks by enhancing their integrated development environment (IDE). To this end, we created the Crypto-Assistant: a modified version of the Hibernate Tools plug-in for the popular Eclipse IDE. The purpose of the Crypto-Assistant is to mitigate the impact of developers' lack of security knowledge related to encryption by facilitating the use of encryption directives via a graphical user interface that seamlessly integrates with Hibernate Tools. Two preliminary tests helped us to identify items for improvement which have been implemented in Crypto-Assistant. We discuss Crypto-Assistant's architecture, interface, changes in the developers' workflow, and design considerations.**

I. Introduction

The lack of encryption for sensitive data is well-known as an important security error; it is one of the top 10 database vulnerabilities [1], occupies the eighth position in the SANS Top 25 Most Dangerous Software Errors [2], and is also one of the top 10 critical web application security flaws according to OWASP [3]. This security flaw occurs when an application stores sensitive data in clear text within a resource that might be accessible to an attacker. When data is sensitive in nature it should be encrypted or otherwise protected so that if it is accessed by an attacker it remains unintelligible.

This security flaw must be addressed by software designers and developers; they can and should make their applications encrypt the sensitive data it manipulates and stores. However, many developers are simply unaware of security considerations. Efforts to help developers gain awareness and understand how to develop secure code exist in educational materials [4]. However, many programmers do not receive such training [5]. Often, security is a secondary goal [6] that might be desired, but not required [5] depending on the criticality and perceived risk of the application developed. Developers' main goals are normally to meet functionality and time to market requirements.

We suggest that this security flaw could be prevented by providing developers with a tool that facilitates the incorporation of security-related tasks early in the development process through the enhancement of their integrated development environment (IDE). We present an IDE extension called the *Crypto-Assistant* with the goals of raising awareness among non-security savvy developers that they should be encrypting sensitive data at rest, and simplifying the encryption process to protect data at rest so that developers without sufficient cryptography knowledge or security training could benefit from its use.

The Crypto-Assistant prototype focuses on data encryption using Hibernate, an Object/Relational Mapping (ORM) tool that facilitates the storage and retrieval of Java objects. One of the easiest ways to achieve the encryption of sensitive information is to use the custom Hibernate data types provided by the Jasypt (Java simplified encryption) library. Nevertheless, the process to encrypt sensitive data with Hibernate and Jasypt is still fairly complex, not particularly intuitive, and prone to errors; this was the motivation for building a tool to help developers in this task. The Crypto-Assistant consists of a series of modifications to the Hibernate Tools plug-in for Eclipse. Crypto-Assistant uses a security warning as a mechanism to communicate the risk of compromising sensitive data due to a lack of encryption. Its ultimate goal is to make the process as easy and intuitive as possible, and reduce the potential number of developer errors.

To improve early versions of Crypto-Assistant, we used an iterative design model by which we conducted two preliminary studies to simulate the usual conditions that a developer has to deal with: little familiarity with the application code, vague requirements, and time constraints. We designed a programming task that involved the use of the prototype and asked the participants to meet certain requirements in a given amount of time. We deliberately hid the fact that we were focusing on security and observed if the changes introduced in the programming system influenced the software artifacts produced. We collected logs, code, and questionnaires from participants to gain insight regarding their behaviour with and perception of the Crypto-Assistant. While these studies involved considerable work, the focus of this paper is on the description of Crypto-Assistant (see discussion in Section IV).

The remainder of this paper is as follows. Section II discusses related work. Section III describes the design of the Crypto-Assistant plug-in including the architecture, user interface, developer workflow, and design considerations. We discuss future work, and conclude in Section IV.

II. Related Work

In Section II-A, we identify relevant literature in the form of taxonomies, standards, guidelines, and frameworks. Next in Section II-B, we discuss other approaches in software tools to help developers achieve stronger security.

A. Guidelines and Documentation for Software Security

The Common Weakness Enumeration (CWE) [7] is perhaps the most complete taxonomy of software vulnerabilities. CWE provides a resource to help programmers avoid vulnerabilities as they design new software and write code, and supports security educators in the development of their teaching curriculum. Nevertheless, the information volume and high detail level of the CWE may impose a heavy cognitive load on developers who work under the clock to deliver their software. Taxonomies of common vulnerabilities include the Open Web Application Security Project's (OWASP) Top Ten Project [3] for web applications, the CWE/SANS Top 25 [2], and the seven kingdoms of security errors of Tsipenyuk et al. [8] .

Security patterns help developers by documenting collective knowledge of recurrent scenarios as experienced by the security community. Some examples of these are the Building Security In Maturity Model (BSIMM) [9]. BSIMM does not provide guidelines for courses of action but it shows what everyone else is doing, helping developers figure out where they stand with respect to peers.

B. Software Tools for Security

According to Ivan et al. [10], research in the area of security tools is mainly focused on tools that assist in the testing of software applications, and tools that help developers create components that lead to secure systems. However, software tools for security are not limited to these two branches. Finifter et al. [11] carried out a comparison of how different programming languages and web frameworks influenced the security of web applications. They found that there is a relationship between the features offered by the frameworks employed were the most effective defences were those that were enabled by default or inherent in framework design or language. They also found that optional protections, even when present in the frameworks, were not used. The different programming languages did not show any specific advantage of one over the other except for the fact that there are errors that apply only to some languages but not to others such as buffer overflows that apply e.g., to C and not Java.

A number of tools have been created as plug-ins for specific IDEs, and perform a specific security analysis on the fly at the same time developers write code. An example of a tool that adopts this strategy is the prototype developed by Xie et al. [5], [12]–[15]. Their prototype offers interactive support for secure programming integrated with the Eclipse IDE that helps developers detect some security errors while they are writing code. The prototype proved to be useful for novice programmers; however, their test with experienced users was not very successful, which may have been attributed in part to the experimental design they applied for the evaluation, among other usability issues of the tool. Some popular tools for source code analysis are HP's Fortify, Coverity's products, SSVChecker (Static Security Vulnerability Checker) and LAPSE (Lightweight Analysis for Program Security in Eclipse) [16]. Some of these tools perform static and dynamic analysis to detect vulnerabilities.

Other tools, like the one presented by Mutti et al. [17], take a different approach, presenting a plug-in to develop security policies using ontological web language, which allows to automate part of the process of validation and verification. The Web Goat project [18] is an educational environment for web application security through the creation of a deliberately insecure web application that can be used by security practitioners to analyze and test security tools. According to Microsoft, the Security Development Lifecycle (SDL) Threat Modeling Tool [19] was the first threat modeling tool not designed for security experts. It presumably makes threat modeling easier for all developers by providing guidance on creating and analyzing threat models. The tool enables any developer or software architect to: (1) characterize systems and analyze data flow diagrams; (2) communicate about the security design of their systems; (3) analyze those designs for potential security issues using a proven methodology called STRIDE; and (4) suggest and manage mitigations for security issues.

In an effort to integrate techniques and tools, and improving them to support the design of usable and secure systems, Faily [20] developed IRIS (Integrating Requirements and Information Security). IRIS guides the selection of techniques for the system design processes that impact security, usability, and requirements. Another security tool for developers is Flawfinder [21], a free open source software that analyzes code and points out vulnerabilities along with some severity score. For a list of other static analysis tools see the list compiled by Wheeler [22].

To the best of our knowledge, Crypto-Assistant is the first tool that focuses on guiding the developer through the integration of encryption in their relational databases within an IDE. Being a cornerstone of cyber security, cryptography provides the primitives that help attain security goals and enforce policies. Therefore we believe that it is necessary to design software tools that foster and facilitate the proper use of cryptography.

III. Crypto-Assistant Design

Crypto-Assistant aims to reduce the prevalence of encryption missing for sensitive data at rest. The Crypto-Assistant design was inspired by the scaffolding capabilities of MyEclipse [23], a closed source commercial implementation of Eclipse that only required a database schema to generate the skeleton for a CRUD (Create, Read, Update and Delete) application. The goal was to influence the development of the application in an early stage; before the cost of any necessary modification would become prohibitive.

We describe the high-level architecture of the Crypto-Assistant in Section III-A, the Crypto-Assistant interface in Section III-B, how the Crypto-Assistant changes the developer's workflow in Section III-C, and some cryptography-related design considerations in Section III-D.

A. Architecture

Crypto-Assistant is built on top of the Hibernate Tools plug-in [24] for the Eclipse IDE. The prototype uses Jasypt to provide its encryption capabilities. Figure 1 shows a broad overview of the Crypto-Assistant architecture. The components depicted in Figure 1 are described further in this section.

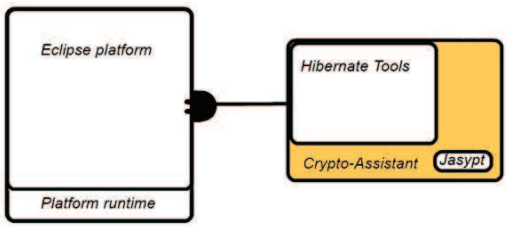

Fig. 1. Crypto-Assistant's high-level architecture.

Hibernate & Hibernate Tools: Hibernate's main goal is to enable developers to keep Java objects persistent by storing them in relational databases. Hibernate abstracts the underlying database and it claims to increase developer productivity by reducing 95% of the Java code that is typically required to access databases. Hibernate provides its own data types that act as translators between the applications and the underlying database. To achieve its functionality, Hibernate uses a set of XML files for configuration and data mapping between the objects and the relational database; data mapping can also be done using code annotations embedded in Java code. Hibernate Tools makes it easier to work with Hibernate; it is a toolset for Hibernate implemented as an integrated suite of Eclipse plug-ins. Hibernate Tools includes an XML mapping editor, a console interface for database queries, a reverse engineering tool to generate HTML documentation and domain model classes, and a set of wizards including one to generate the Hibernate configuration files.

Jasypt Java Simplified Encryption: Jasypt [25] is a Java library that allows developers to add symmetric encryption capabilities with minimum effort, and without the need of having deep knowledge on how cryptography works. Normally, the use of encryption in Java requires the programmer to have a broad understanding of Java and cryptography recommended modes of use. Jasypt simplifies the use of encryption, providing a more clear and concise application programming interface (API) that is easy to understand and use. With Jasypt, encrypting and checking a password can be as simple as:

```
BasicPasswordEncryptor passwordEncryptor = new BasicPasswordEncryptor();
String encryptedPassword = passwordEncryptor.encryptPassword(userPassword);
...
If (passwordEncryptor.checkPassword(inputPassword, encryptedPassword)) {
   // correct!
} else {
   // bad login!
}
```

The encryption and decryption of text can be as simple as:

```
BasicTextEncryptor textEncryptor = new BasicTextEncryptor();
textEncryptor.setPassword(myEncryptionPassword);
String myEncryptedText = textEncryptor.encrypt(myText);
...
String plainText = textEncryptor.decrypt(myEncryptedText);
```

The encryption of sensitive data directly from Hibernate involves modifying the property tags in an XML mapping file as follows:

```
<class name="Employee" table="EMPLOYEE">
   ...
   <property name="address" column="ADDRESS" type="encryptedString" />
   <property name="salary" column="SALARY" type="encryptedDecimal" />
   ...
<class>
```

These are steps in the right direction, but further steps can be taken to simplify the process even more, and build the tasks directly into the developer's workflow in the hope of increasing awareness. This is the reason why Crypto-Assistant was developed.

B. Interface

Most of the Crypto-Assistant functionality is not visible to the user. The only visible modification consists of the addition of two new pages in the middle of the wizard, right before the user finishes the mapping from Java classes to database tables. These new pages are followed depicted in Figures 2 and 3, in the same order as they appear in the wizard.

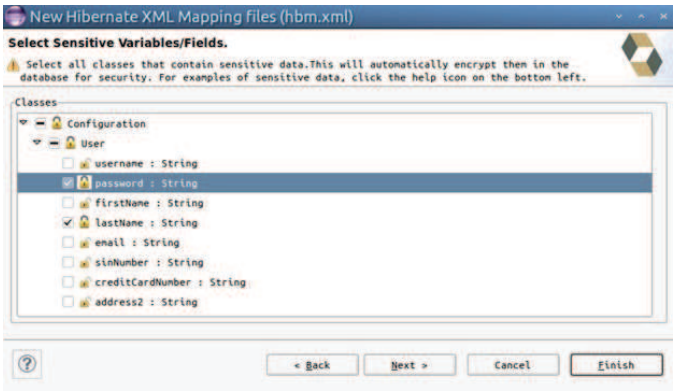

Fig. 2. Selection of fields to be encrypted.

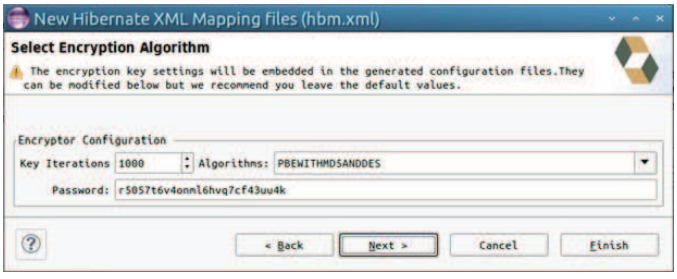

Fig. 3. Property Encryption page, added by the Crypto-Assistant.

The new page added by the Crypto-Assistant, shown in Figure 2, presents a security warning whose intention is to raise awareness about the risk of storing sensitive data without encryption, and it offers a course of action to mitigate that risk,

allowing developers to select the properties or fields of a class containing sensitive information. On a subsequent screen, it is possible to configure the password, encryption algorithm, and key iterations used to generate the encryption key for the fields selected.

The target users are developers in general, who have little knowledge regarding security. The new page added displays a warning message with the purpose of influencing the developer to incorporate the selection of sensitive data for encryption as part of his or her current goals. Developers must select the checkbox beside the fields that they believe to be sensitive. The page does not explicitly indicate which fields must be selected, as this will depend on the data stored by each individual program. The selection area stands out from the other components on the page by taking most of the space available, to draw developers' attention to the selection task required. All of the controls and messages on the screen are associated with the protection of sensitive data through encryption. Developers will understand they are making progress towards protecting the sensitive data as they will see a locked padlock icon next to the fields they select and an unlocked padlock next to the fields they have not selected.

C. How Crypto-Assistant Changes The Developer's Workflow

A typical sequence of steps to encrypting sensitive fields of a database using Hibernate and Jasypt *without* the Crypto-Assistant would involve: (1) generating a mapping file between object classes and the database; (2) for those fields that require encryption, the user needs to manually change the mapping file to use the appropriate Jasypt type, making sure the selected fields are not primary or foreign keys whose encryption may break the relations of the database; (3) select a password, encryption algorithm, and key derivation cycles; and (4) update the Hibernate configuration file. Based on surveys in our two preliminary user studies of Crypto-Assistant, we identified several factors that contribute to the problem of developers not implementing encryption, which can be summarized as follows: (a) lack of awareness about the risks of storing sensitive data in plain text; (b) lack of knowledge about available protection mechanisms and their effective use; and (c) lack of usability of the protection mechanisms. Crypto-Assistant makes this process easier by providing a graphical user interface where the user may chose eligible (i.e., non-key) fields for encryption and then an interface for adjusting default values (if desired) for the encryption algorithm, password, and key derivation cycles. Without Crypto-Assistant, steps 2-4 are totally unsupervised and non-validated, and thus prone to syntax or logical errors that can break the relationship of tables in the database. Crypto-Assistant validates these steps by providing an encryption user interface integrated within the IDE, inside the "Hibernate Mapping File" wizard.

Crypto-Assistant helps reduce the chance of human error in several ways. More specifically, it hides properties such as the ones used as primary or foreign keys whose encryption would break the entity relations. This might be confusing if users are looking for these specific fields but it prevents them from breaking the relations in the database by mistake. For the encryption algorithm, AES (Advanced Encryption Standard) and 3DES (Triple Data Encryption Standard) are two recommended algorithms by NIST [26]; however, the default security policy of the Java Virtual Machine (JVM) put limits on the cryptographic strength available by default. The process to enable stronger cryptography requires the manual installation of unrestricted policy files (see discussion in Section III-D below). Development of a tool to assist developers in the installation and the detection of this file requires significant effort, and we considered this task out of the scope of the prototype. Once the user decides to proceed to the preview page, heuristics are applied to assign a suitable encrypted type to each one of the properties selected. At the end the mapping files generated contain embedded configuration settings to allow Hibernate to use Jasypt's custom data types to perform the encryption and decryption of the selected properties.

D. Design Considerations

1) Recommended encryption algorithms: The encryption algorithm was an important factor to consider. Our implementation allows the developer to choose their encryption algorithm. Some databases use DES (Data Encryption Standard) encryption to protect sensitive data. However, DES has long been considered insufficient to protect any information. As mentioned above, AES and 3DES are, at the time of writing, two of the recommended algorithms by NIST for symmetric encryption [26]. AES encrypts and decrypts data in 128-bit blocks, using 128, 192 or 256 bit keys. All three key sizes are considered adequate by NIST for Federal Government applications [27]. To minimize the chance of using a weak encryption algorithm, we implemented a warning mechanism to alert the user if the algorithm selected for encryption is different from AES. The mechanism is limited to inform the user that AES is the recommended algorithm but it requires the strong encryption configuration.

2) Key management and key generation password: Part of managing keys is deciding where to store them. One easy solution is to store the keys in a restricted database table or file. But, all administrators with privileged access could also have access to these keys, decrypt any data within the system, and then cover their tracks. The recommended approach is to use a Hardware Security Module (HSM) to store the keys. In this case, the keys never leave the hardware and therefore access can be controlled so neither administrators nor intruders can penetrate the machine and steal them.

To avoid using a default key generation password, a random password is generated every time the Crypto-Assistant is used. The developer is responsible for keeping track of the password for future use. Optionally, the users can choose their own password. If the wizard is used to make modifications to the configuration files, a new password will be generated by default, users would have to enter it manually each time they make changes and want to keep the same encryption key.

We recommend the separation of the database and application servers. This architecture protects against rogue database

administrators and media stealing; even if the data can be accessed the key needed to decrypt the data is still out of reach. Proper management involves restricting personal access to key storage locations, random key updates, and encoded key storage servers. An effective key management system involves every aspect of key creation like distribution, revocation, network access, and personnel management. Key management is outside of the scope of this work.

3) Database encryption strategy: The database encryption strategy implemented takes encryption and decryption out of the DBMS; the workload takes place at the application server where Hibernate is running and it integrates transparently with the application. Thus, the application does not require any changes in its code. Some code changes may be required, but only to support cryptography best practices (e.g., key rotation [27] that involves decryption of the data with the old key and re-encryption of it using a new key).

IV. Future Work and Conclusion

There are some avenues for future work in the functionality of Crypto-Assistant, e.g., by adding support for key management and key rotation. Some progress has already been made towards moving the definition of the encrypted types to a separate file that would be managed by the Crypto-Assistant to minimize the exposure of the encryption keys. It may also be worth allowing developers to revisit the tool through a menu interface. If users become aware of the feature through the warning screen, but ignore it at first due to time considerations, this would allow them to return at a later time at their leisure.

The two preliminary user studies proved to be a challenging task for a number of reasons, including finding large numbers of qualified participants with knowledge of Java and Hibernate, as well as the length of the study per participant. As a result, we felt that it would be premature to draw solid conclusions on the effectiveness of Crypto-Assistant based on these preliminary studies. Nevertheless, these studies helped shape the current version of Crypto-Assistant as we were able to realize what needed to be done to improve the tool in its early versions. Future work would include launching the tool publicly and collecting information as to how users utilize it in their normal development tasks, resulting in a more ecologically valid study outside of the laboratory environment.

It is our hope that this paper stimulates IDE developers to create and study more tools like Crypto-Assistant to help create more secure applications.

Acknowledgments

The second and third authors would like to thank the Natural Sciences and Engineering Council of Canada (NSERC) for funding Discovery Grants. We thank Destiny Ewansiha for his role in improving an earlier version of Crypto-Assitant.

References

[1] Team Shatter, 2012, [Accessed: 16-Sep-2012]. [Online]. Available: http://www.teamshatter.com/

[2] "CWE - 2011 CWE/SANS Top 25 Most Dangerous Software Errors," 2012, [Accessed: 06-Nov-2012]. [Online]. Available: http://cwe.mitre.org/top25/index.html#CWE-311

[3] "Category: OWASP Top Ten Project - OWASP," 2012, [Accessed: 04-Nov-2012]. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[4] M. G. Graff and K. R. V. Wyk, *Secure Coding: Principles and Practices*. OReilly & Associates, Inc., 2003.

[5] J. Xie, H. Lipford, and B. Chu, "Why do Programmers Make Security Errors?" in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2011, pp. 161–164.

[6] A. Whitten, "Making Security Usable," PhD thesis, Carnegie Mellon University, 5000 Forbes Avenue Pittsburgh, PA, USA, 2004.

[7] "CWE - VIEW SLICE: CWE-2000: Comprehensive CWE Dictionary (2.3)," 2012, [Accessed: 04-Nov-2012]. [Online]. Available: http://cwe.mitre.org/data/slices/2000.html

[8] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven Pernicious Kingdoms: a Taxonomy of Software Security Errors," *IEEE Security Privacy*, vol. 3, no. 6, pp. 81–84, 2005.

[9] "The Building Security In Maturity Model (BSIMM)," 2012, [Accessed: 14-Nov-2012]. [Online]. Available: http://bsimm.com/

[10] I. Ivan and L. Breda, "Informatics Security Metrics Comparative Analysis," *Informatica Economica*, vol. XI, no. 4, pp. 107–110, 2007.

[11] M. Finifter and D. Wagner, "Exploring the Relationship Between Web Application Development Tools and Security," in *Proceedings of the 2nd USENIX Conference on Web Application Development*, Portland, USA, Jun. 15–16 2011.

[12] J. Xie, H. Lipford, and B.-T. Chu, "Evaluating Interactive Support for Secure Programming," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, USA, 2012, pp. 2707–2716.

[13] J. Xie, B. Chu, , and H. Richter Lipford, "Idea: Interactive Support for Secure Software Development," in *Proceedings of the Third International Symposium Engineering Secure Software and Systems*, Madrid, Spain, Feb. 9–10 2011, pp. 248–255.

[14] J. X. J. Zhu, B. Chu, "OWASP ASIDE Project," 2014, [Accessed: 11-Mar-2014]. [Online]. Available: https://www.owasp.org/index.php/OWASP_ASIDE_Project

[15] J. Zhu, J. Xie, H. R. Lipford, and B. Chu, "Supporting Secure Programming in Web Applications through Interactive Static Analysis," *Journal of Advanced Research*, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2090123213001422

[16] "OWASP LAPSE Project - OWASP," 2012, [Accessed: 8-Nov-2012]. [Online]. Available: https://www.owasp.org/index.php/OWASP_LAPSE_Project

[17] S. Mutti, M. Neri, and S. Paraboschi, "An Eclipse Plug-in for Specifying Security Policies in Modern Information Systems," in *The Sixth Workshop of the Italian Eclipse Community*, 2011.

[18] "Category: OWASP WebGoat Project - OWASP," 2012, [Accessed: 10-Dec-2012]. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

[19] "SDL Threat Modeling Tool," 2012, [Accessed: 11-Dec-2012]. [Online]. Available: http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx

[20] S. Faily, "A Framework for Usable and Secure System Design," PhD thesis, University of Oxford, Oxford, UK, 2011.

[21] D. Wheeler, "Flawfinder," 2014, [Accessed: 11-Mar-2014]. [Online]. Available: http://www.dwheeler.com/flawfinder/

[22] ——, "Other Static Analysis Tools for Security," 2014, [Accessed: 11-Mar-2014]. [Online]. Available: http://www.dwheeler.com/flawfinder/#othertools

[23] "MyEclipse for Spring: Spring MVC Scaffolding," 2013, [Accessed: 16-Jan-2013]. [Online]. Available: http://www.myeclipseide.com/documentation/quickstarts/scaffoldingtutorial/scaffolding.html

[24] "Hibernate Tools - JBoss Community," 2012, [Accessed: 03-Dec-2012]. [Online]. Available: http://www.hibernate.org/subprojects/tools.html

[25] "Java Simplified Encryption," 2013, [Accessed: 29-Mar-2013]. [Online]. Available: http://www.jasypt.org/

[26] Computer Security Resource Center, "Cryptographic Toolkit: Block Ciphers," 2013, [Accessed: 07-Jan-2014]. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html

[27] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST Special Publication 800-57: Recommendation for Key Management Part 1: General (Revision 3)," 2012, [Accessed: 07-Jan-2014]. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf