Encryption – a resolution on CASS-admin protection

Abstract

Encrypting data during database storage is a good way of protecting it, but the method of encryption poses a problem. There are two ways of achieving encryption, and both of them have their problems. Asymmetric cipher would be an elegant and safe solution, but is currently relatively difficult to build and the resources available to the project make it an infeasible option. The other possibility is to use symmetric encryption, which – unlike asymmetric encryption – does not offer a significantly higher amount of protection if the server is completely taken over by malicious crackers. This however is a more feasible option time wise, and as the current deployed system offers no data protection whatsoever, aside from database user control, it is suggested that for now, the system would be built with the symmetric encryption option. Using encryption will effectively destroy the possibility for subjects to review their own answers, unless they are given the encryption key (which would effectively defeat the purpose of having encryption in the first place).

Introduction

Previously it was decided that all identifiable information (text, sound, video and images) should be encrypted during transportation and storage when in the CASS database. This decision was based on the assessment that potential information leak, however unlikely, would be an extremely bad thing from a privacy standpoint, even if potential damages on a concrete level would be trivial. The reason for not including numerical data in encryption, was to allow the database to perform numerical operations on the data. Research revealed that two options exist for safe encryption of the stored data, called symmetric and asymmetric ciphers, or private and public key cryptography. Cryptography means transforming data to an unreadable and unrecognizable form, so that a person without a specific key would not be able to read the data, even with access to it.

Asymmetric encryption

Asymmetric encryption means that the key for encrypting the data is different from the key used in decryption. This means that the key used for encrypting the data can be safely stored anywhere, as it can only be used in adding more data to the system, but not for reading from the system. This method would provide a strong data protection, even if an attacker would get complete control over the server system. As elegant and secure this method is, the current PHP implementation (known as OpenSSL) would require compiling the PHP environment from the original source codes - which is certainly within the boundaries of possibility in the KP-Lab environment, however this procedure would have to be done every time a CASS system is to be deployed on an end user system, and thereby make the already complex installation procedure more difficult (other solutions for this problem exist, but none are any easier). Another problem with this approach is, that not many available and well documented examples are available. Even so, building the CASS-admin system on this framework would certainly be possible, but infeasible within the boundaries of current project resources (mainly man hours). Another implementation also exists, called Mcrypt, however no documentation exists on it's asymmetric cipher functions at this time.

Symmetric encryption

Symmetric encryption means that the same key is used for encrypting and decrypting the data. In practice, this means that the encryption key has to be saved somewhere in the server in order for users to be able to enter data without entering an encryption key. Requiring users to enter the key would be problematic as the key would need to be placed on the subjects mobile phones and transmitted over. The current CASS Study software is not designed to do that. The problem with saving the key on the server is, that from the server it is relatively easy to find such a key, once the attacker has gained control of the server. This problem is derived from the fact that PHP code is always stored in human readable form and not compiled into machine code. This means that even without prior knowledge about the systems structure, the attacker would quickly find out the secret key, and be able to access the data. On the other hand, this method of security does provide protection, if an attacker is only able to gain control of the database server (which is often considerably easier) as the data would still remain in non readable form. Any database dumps and backups would also be protected, so losing a copy of the database's contents would not be catastrophic.

Conclusion

The most effective way of getting the new software version out, while maintaining sufficient data protection, is to use symmetric encryption at this point, but provide a framework for adding the asymmetric encryption at a later point if it is deemed necessary.