Password Based Encryption

By Mohan Atreya (matreya@rsasecurity.com)

Summary

This article is the third in this series. This series aims to give the reader a bottoms-up introduction to the basics of e-security. The goal of this article is to introduce the reader to the available password based encryption standards. The limitations of password based encryption standards are also discussed.

Introduction

This tutorial assumes that the reader is familiar with basic terms in cryptography such as Public Key cryptography, Secret Key cryptography and Message Digest algorithms. Please review these concepts in Articles 1&2 of this series before proceeding further with this tutorial. Alternatively, please follow the link (**http://www.rsasecurity.com/rsalabs/faq/index.html/**) for a set of frequently asked questions (FAQ) on e-security from RSA Laboratories.

In the previous articles, we saw how secret key cryptography works. We also understood that the strength of a secret key strongly depends on the fact that people cannot guess the secret key easily. This means that the secret key would most probably be in a format that normal users will not be able to remember easily. Simply deriving the bytes from a password does not produce enough random bits to generate a strong enough secret key. We will discuss methods to generate random bits for encryption keys in Article-4 of this series.

Why do we need password-based encryption?

Some users want to encrypt and decrypt their files with an easy to remember password (key) and at the same time be confident that their files are secure from prowling eyes. Public key encryption requires the secure storage of the private key. The loss or compromise of the private key can be disastrous to the user.

Password based encryption (PBE) was designed to solve problems of the kind described above. A PBE algorithm generates a secret key based on a password, which will be provided by the end user. Currently there are two standards (PKCS #5 and #12) that define how a password can be used to generate a symmetric key. A good PBE algorithm will also mix in a random number called the salt along with the password to create the key. Without a salt, the hacker can perform a brute force search for the **key-space** with relative ease.

PBE is typically used in systems such as local file encryption tools, which are used to ensure data confidentiality. They are also used as a mechanism to protect the user's private key store (such as the PKCS #8 based protection of private keys). User prompted passwords are typically either a subset of ASCII or UTF-8 for purposes on inter-operability. It should be noted that UTF-8 is a superset of ASCII.

Why do we need a salt?

The salt is a value that can thwart dictionary attacks or pre-computation attacks. An attacker can easily pre-compute the digests of thousands of possible passwords and create a "**dictionary**" of likely keys. Recall the fact that when you perform the digest, changing input data even a little changes the resulting digest. By digesting the password with a salt, the attacker's dictionary is rendered useless. The attacker will need to search through passwords for each value of the salt. Alternatively, the attacker has to wait until a password operation is performed and the salt used in that particular operation is captured. Because the salt is random in nature, it is highly unlikely that the same salt will be used for the next encryption process thus limiting the attacker further.

The salt needs to be generated using a pseudo random number generator (PRNG). It is also strongly recommended not to reuse the same salt value for multiple instances of encryption. Note that the salt is not a secret value. So, it can be transmitted along with the cipher-text to the receiver or via out-of-band transmission methods. Ideally the length of the salt should be same as the output of the hash function being used.

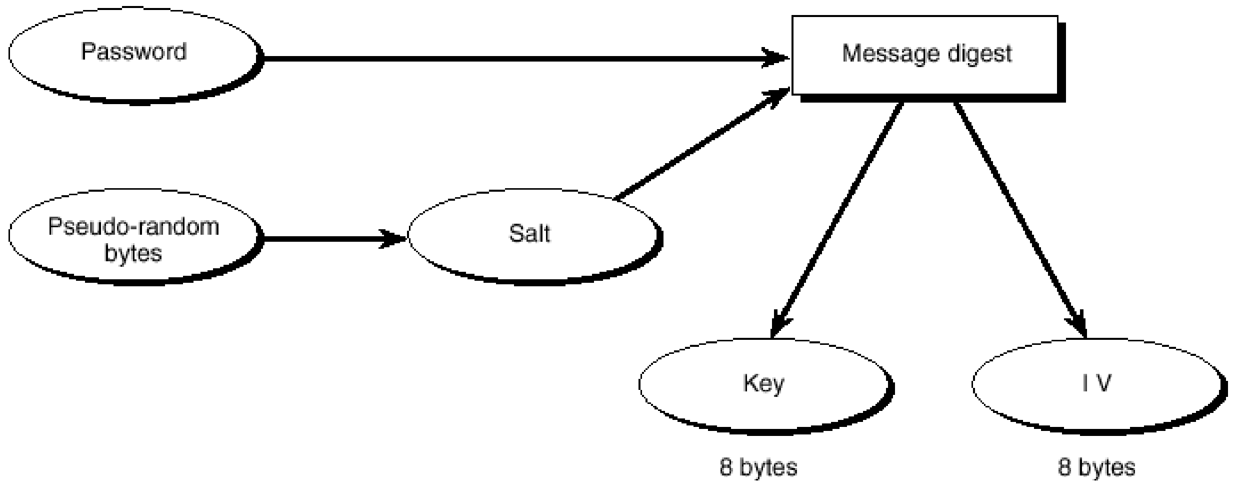
Iterations

Another important deterrent that can be used to thwart the advances of the attacker is to include an iteration count. This will complicate the key derivation function by performing a number of iterations. The iteration count increases the cost of exhaustive password search attacks by a significant amount. A minimum of 1000 iterations is recommended for minimum-security requirements. Just like the salt, the iteration count does not have to be kept a secret and can be transmitted in the clear along with the cipher-text if necessary. Usually the salt, the iteration-count value and are sent to the receiver as a part of the **algorithm identifier** value.

The Standards

The most rudimentary of the standards available for PBE is PKCS#5 v1.5. The following figure illustrates the process of generating a secret key using the PKCS#5 v1.5 standard. The salt is appended to the password before being digested using one-way functions such as MD5 or SHA-1. The choice of the digest limits the key size that can be derived using the standard. MD5 gives a digest of 128 bits and SHA-1 results in a 160- bit digest.

In the following figure, if we used MD5 as the digest algorithm, the end result of this process will be 16 bytes (128-bits) of data. Since most symmetric ciphers need an **Initialization Vector** (IV) for their operation, only 8 bytes (64-bits) can be used as key material for the cipher. Thus, this particular standard can be used to generate 64-bit secret keys or weaker.



In order to generate stronger keys, we need to use standards such as PKCS#5 v2.0 or PKCS#12. The length of the keys that can be generated by these two standards is essentially unlimited. These two standards also go much beyond simple key generation and key derivation functions for password-based encryption. They also have support for password based message authentication schemes. Incidentally PKCS#5 v2.0 supersedes the PKCS#5 v1.5 standard, but includes compatible techniques too.

In general, the PKCS#5 v2.0 and PKCS#12 standard can be used in both "***password secrecy***" and "***password integrity***" modes. The password privacy mode generates a secret key for encryption and the password integrity mode generates a Message Authentication Code (MAC) key.

Limitations

The PBE standards leave some areas open to the discretion of the developer. For example, the choice of the password is not limited in any way by the standards. It is up to the application to determine whether the chosen password is strong or weak. The standard also does not specify the format for the password. But, to be fully interoperable with most applications, it is suggested that developers use ASCII strings and not local strings.

Conclusion

In this article, we explored the internals and the mechanics of password based encryption (PBE) algorithms. We also discussed two important standards, which describe password based encryption methods.

In the next installment in this series, we will learn how pseudo random number generators (PRNGs) work and illustrate how these are used in the crypto world. We will see why PRNGs are so important in order to generate strong ciphers and avoid compromises of keys.

About the Author
Mohan Atreya is a Technical Consultant with RSA Security. He has advanced degrees from National University of Singapore and Nanyang Technological University.

Acknowledgements
The author acknowledges the significant contributions by Kimberly Getgen and Teo LayPeng to this article.

A Message to Developers
The RSA BSAFE (**http://www.rsasecurity.com/products/bsafe/**) family of toolkits provides you with all the components you need to make your applications safe and secure. As a developer, you can save many months of development and testing, thus allowing you to focus on your application development and roll out your application with confidence. The BSAFE family comprises the following toolkits:

Core Functionality	**BSAFE Toolkit Details**
Core Cryptographic Toolkits	BSAFE Crypto-C & BSAFE Crypto-J
Public Key Infrastructure (PKI) Toolkits	BSAFE Cert-C & BSAFE Cert-J
Protocol Level Toolkits	BSAFE SSL-C & BSAFE SSL-J (SSL protocol for point-point security) BSAFE S/MIME-C (S/MIME Protocol for secure messaging) BSAFE WTLS-C (Wireless Transport Layer Security for WAP)