



## Contents

- [1 Overview](#)
- [2 API Notes](#)
- [3 Preconditions](#)
- [4 Source file](#)
- [5 Postconditions](#)
- [6 Supplementary material](#)
- [7 See also](#)

## Overview

When storing sensitive data it is very important that existing encryption support is used rather than "hand rolled" solutions written by inexperienced security developers. Getting cryptographic solutions right is very hard and best left to experts. To this end Symbian have provided a subset of the cryptographic libraries for use by third parties and these should be used to store sensitive data since they are cryptographically secure. This code snippet shows how to securely store data so that the only way it may be retrieved is to supply a password.

The first code snippet contains the code to encrypt a blob of text with a password and write that out to a stream. The second code snippet reverses the first, in that given a password and a stream, the encrypted blob will be decrypted and the plain text returned.

## API Notes

The password based encryption API's provide an object to hide the encryption functionality. This is the CPBEncryptElement and selects the default encryption algorithm and sets up the various vectors to ensure that the data and password are encrypted to accepted standards.

The element then provides two sub elements, one of which performs the encryption and one of which provides the decryption. In addition to providing the cryptography, the element also provides methods to allow the data to be serialized to and from a stream securely.

## Preconditions

The cryptography API's are required to be installed. These can be found [here](#).

## Source file

Example code to encrypt a blob of data with a password and write it to a stream.

## Password\_based\_encryption

```

void CExampleStubAppUi::EncryptDataL(const TDesC& aPassword, const TDesC8& aPlainTextBlob, RWriteStream& aStream)
{
    const TPtrC8 passwordPtr(REINTERPRET_CAST(const TUint8*, aPassword.Ptr()), aPassword.Length() * sizeof(TUint8));

    // create a new element
    const TPBPassword encryptionKey(passwordPtr);
    CPBEncryptionElement element = CPBEncryptionElement::NewLC(encryptionKey);

    // create a new encryptor object based on the encryption data in the element
    CPBEncryptor encryptor = element->NewEncryptLC();

    // allocate enough space to hold the encrypted text
    // and then encrypt the text
    * HBufC8 ciphertext = HBufC8::NewLC(encryptor->MaxFinalOutputLength(aPlainTextBlob.Length()));
    TPtr8 ciphertextPtr(ciphertext->Des());
    encryptor->ProcessFinalL(aPlainTextBlob, ciphertextPtr);

    // finally write out the encryption data
    const CPBEncryptionData& header = element->EncryptionData();
    header->WriteL(aStream);

    // and the encrypted text
    const TInt length = ciphertextPtr.Length();
    aStream->WriteInt32L(length);
    aStream->Write(ciphertextPtr, length);

    CleanupStack::PopAndDestroy(3, element); // ciphertext, encryptor, element
}

```

Example code to decrypt a blob of data (encrypted by the above) given the password and a stream.

```

HBufC8* CExampleStubAppUi::DecryptDataL(const TDesC& aPassword, RReadStream& aStream)
{
    const TPtrC8 passwordPtr(REINTERPRET_CAST(const TUint8*, aPassword.Ptr()), aPassword.Length() * sizeof(TUint8));

    const TPBPassword encryptionKey(passwordPtr);
    // Recover the encryption data from the stream
    CPBEncryptionData encryptionData = CPBEncryptionData::NewLC(aStream);

    // and then read the encrypted text
    const TInt length = aStream.ReadInt32L();
    * HBufC8 encryptedBlob = HBufC8::NewLC(length);
    TPtr8 encryptedBlobPtr(encryptedBlob->Des());
    aStream->Read(encryptedBlobPtr, length);

    // now create an element with the encryption data
    // and get the decryptor object based on the encryption data
    CPBEncryptionElement element = CPBEncryptionElement::NewLC(*encryptionData, encryptionKey);
    CPBDecryptor decryptor = element->NewDecryptLC();

    // Find out how big the result will be and allocate space for it
    HBufC8 * plaintext = HBufC8::NewLC(decryptor->MaxFinalOutputLength(encryptedBlobPtr.Length()));
    TPtr8 plaintextPtr = plaintext->Des();

    // Now decrypt the data
    decryptor->ProcessFinalL(encryptedBlobPtr, plaintextPtr);

    CleanupStack::Pop(plaintext);
    CleanupStack::PopAndDestroy(4, encryptionData); // decryptor, encryptedblob, element

    return plaintext;
}

```

## Postconditions

The text is encrypted, written to a file stream, reloaded and then decrypted. If the password is incorrect the function will leave.

## Supplementary material

- You can test link handling code in a simple, executable application into which this code snippet has been patched. The application is available for download at: [File:ExampleStub PBExample2.zip](#)
- You can examine all the changes that are required to add message box links in an application. The changes are provided in the unified diff format: [File:ExampleStub PBExample2.diff.zip](#)
- For general information on applying the patch, see [Using Diffs](#).
- For unpatched stub applications, see [Example stub](#).

## See also

The documentation on the cryptography libraries supplied with the cryptography download.