Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'EVP_SealUpdate.3ossl' command

$ man EVP_SealUpdate.3ossl

EVP_SEALINIT(3ossl) OpenSSL EVP_SEALINIT(3ossl)

NAME

EVP_SealInit, EVP_SealUpdate, EVP_SealFinal - EVP envelope encryption

SYNOPSIS

```
#include <openssl/evp.h>
int EVP_SealInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
                 unsigned char **ek, int *ekl, unsigned char *iv,
                 EVP_PKEY **pubk, int npubk);
int EVP_SealUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
                  int *outl, unsigned char *in, int inl);
int EVP_SealFinal(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);
```

DESCRIPTION

The EVP envelope routines are a high-level interface to envelope encryption. They generate a random key and IV (if required) then "envelope" it by using public key encryption. Data can then be encrypted using this key.

EVP_SealInit() initializes a cipher context ctx for encryption with cipher type using a random secret key and IV. type is normally supplied by a function such as EVP_aes_256_cbc(). The secret key is encrypted using one or more public keys, this allows the same encrypted data to be decrypted using any of the corresponding private keys. ek is an array of buffers where the public key encrypted secret key will be written, each buffer must contain enough room for the corresponding

encrypted key: that is ek[i] must have room for
EVP_PKEY_get_size(pubk[i]) bytes. The actual size of each encrypted
secret key is written to the array ekl. pubk is an array of npubk
public keys.

The iv parameter is a buffer where the generated IV is written to. It
must contain enough room for the corresponding cipher's IV, as
determined by (for example) EVP_CIPHER_get_iv_length(type).
If the cipher does not require an IV then the iv parameter is ignored
and can be NULL.

EVP_SealUpdate() and EVP_SealFinal() have exactly the same properties
as the EVP_EncryptUpdate() and EVP_EncryptFinal() routines, as
documented on the EVP_EncryptInit(3) manual page.

RETURN VALUES

EVP_SealInit() returns 0 on error or npubk if successful.

EVP_SealUpdate() and EVP_SealFinal() return 1 for success and 0 for
failure.

NOTES

Because a random secret key is generated the random number generator
must be seeded when EVP_SealInit() is called. If the automatic seeding
or reseeding of the OpenSSL CSPRNG fails due to external circumstances
(see RAND(7)), the operation will fail.

The public key must be RSA because it is the only OpenSSL public key
algorithm that supports key transport.

Envelope encryption is the usual method of using public key encryption
on large amounts of data, this is because public key encryption is slow
but symmetric encryption is fast. So symmetric encryption is used for
bulk encryption and the small random symmetric key used is transferred
using public key encryption.

It is possible to call EVP_SealInit() twice in the same way as
EVP_EncryptInit(). The first call should have npubk set to 0 and (after
setting any cipher parameters) it should be called again with type set
to NULL.

SEE ALSO

evp(7), RAND_bytes(3), EVP_EncryptInit(3), EVP_OpenInit(3), RAND(7)

COPYRIGHT