OpenSSL's Implementation of Infinite Garble Extension
Version 0.1

Ben Laurie
(ben@links.org)

August 30, 2006

1 Introduction

Infinite Garble Extension (IGE) is a block cipher mode[1]. It has the property that errors are propagated forward indefinitely. Bi-directional IGE (biIGE) propogates errors in both directions: that is, any change to the ciphertext will cause all of the plaintext to be corrupted.

IGE and biIGE have some leeway in their definitions admitting several possible implementations. This paper documents the way I chose to implement them in OpenSSL, and also provides test vectors.

It also points out some implementation details that may not be obvious from the original papers.

2 IGE mode

IGE mode uses the following chaining sequence

$$y_i = f_K(x_i \oplus y_{i-1}) \oplus x_{i-1} \tag{1}$$

where f_K is the underlying block cipher encrypting function with key K, i runs from 1 to n, and n is the number of plaintext blocks.

This means that for the first output block, y_1, we need two non-existent inputs, y_0 and x_0. These correspond to the initialisation vector (IV) in more traditional modes such as CBC. The implementation specified in the original paper keeps the IV to a single random block by using a second encryption key, K' and setting

$$x_0 = \{0,1\}^l \tag{2}$$

$$y_0 = f_{K'}(x_0) \tag{3}$$

where l is the block size, in bits.

I decided to use a more general implementation where both x_0 and y_0 are provided by the user, rather than using a second key to derive y_0. The implementation is otherwise as described in the original paper.

This is for three reasons, firstly it is more efficient if both blocks are chosen randomly, and secondly because the original implementation can be expressed in terms of OpenSSL's implemention but the opposite is not the case.

The third reason is to do with the way block cipher modes are often implemented – because it is possible to encrypt each block as it becomes available, rather than waiting until the entire plaintext is available, many implementations (including OpenSSL's) make it possible to encrypt partial plaintexts. The necessary chaining information is commonly stored in the memory which was used for the IV. This makes perfect sense, because on subsequent calls to the encryption (or decryption) function, the output is exactly as if the chaining information had, in fact, been provided as an IV.

The implementation as described maintains this possibility, which the original implementation does not.

Note, however, that this makes the IV two blocks long, instead of the usual one block. OpenSSL uses the convention that the first block of the IV is x_0 and the second block is y_0.

3 Bi-directional IGE mode

Again, I opted for the most general possible implementation of biIGE. The chaining sequence for biIGE is

$$z_i = f(x_i \oplus z_{i-1}) \oplus x_{i-1} \tag{4}$$

$$y_i = f'(z_{n-i+1} \oplus y_{i-1}) \oplus z_{n-i+2} \tag{5}$$

where f and f' are two encryption fuctions (typically the same cipher with different keys), n is the total number of plaintext blocks and i runs from 1 to n. Again this leaves various values to be provided by the user, namely x_0, z_0, y_0 and z_{n+1}. Note that the second part of this chaining sequence appears to be incorrectly specified in the original paper.

Unlike IGE mode, chaining between partial plaintexts is not feasible. Note that the IV is four blocks long, rather than the usual one. OpenSSL uses the convention that the first block of the IV is x_0, the second z_0, the third z_{n+1} and the fourth y_0.

4 Test vectors

4.1 AES IGE Mode Test Vector 1

Key

```
00010203 04050607 08090A0B 0C0D0E0F
```

Initialisation Vector

```
00010203 04050607 08090A0B 0C0D0E0F
10111213 14151617 18191A1B 1C1D1E1F
```

Plaintext

```
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

Ciphertext

```
1A8519A6 557BE652 E9DA8E43 DA4EF445
3CF456B4 CA488AA3 83C79C98 B34797CB
```

4.2 AES IGE Mode Test Vector 2

Key

```
54686973 20697320 616E2069 6D706C65
```

Initialisation Vector

```
6D656E74 6174696F 6E206F66 20494745
206D6F64 6520666F 72204F70 656E5353
```

Plaintext

```
99706487 A1CDE613 BC6DE0B6 F24B1C7A
A448C8B9 C3403E34 67A8CAD8 9340F53B
```

Ciphertext

```
4C2E204C 65742773 20686F70 65204265
6E20676F 74206974 20726967 6874210A
```

4.3 AES Bi-directional IGE Mode Test Vector 1

Key 1

```
00010203 04050607 08090A0B 0C0D0E0F
```

Key 2

```
10111213 14151617 18191A1B 1C1D1E1F
```

Initialisation Vector

```
00010203 04050607 08090A0B 0C0D0E0F
10111213 14151617 18191A1B 1C1D1E1F
20212223 24252627 28292A2B 2C2D2E2F
30313233 34353637 38393A3B 3C3D3E3F
```

Plaintext

```
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

Ciphertext

```
14406FAE A279F256 1F86EB3B 7DFF53DC
4E270C03 DE7CE516 6A9C2033 9D33FE12
```

4.4 AES Bi-directional IGE Mode Test Vector 2

Key 1

```
580a06e9 9707595c 9e19d2a7 bb402b7a
c7d8119e 4c513575 64280f23 ad74ac37
```

Key 2

```
d180a031 47a31113 86269e6d ffaf7274
5ba23581 d2a63d21 677b58a8 18f972e4
```

Initialisation Vector

```
803dbd4c e67b06a9 5335d57e 71c17070
749a0028 0cbf6c42 9ba4dd65 11777c67
fe760af0 d5c66e6a e75e4cf2 7e9ef920
0e546f2d 8a8d7ebd 48793799 ff2793a3
```

Plaintext

```
f1543dca feb5ef1c 4fa643f6 e64857f0
ee157fe3 e72fd02f 11957a17 00aba70b
be44099c cdaca852 a18e7b75 bca4925a
ab46d33a a0d5351c 55a4b3a8 4081a50b
```

Ciphertext

```
42e52830 31c2a023 68494eb3 24599279
c1a5cce6 7653b1cf 208623e8 72559992
0d161c5a 2fcecb51 e267fa10 eccd3d67
a5e6f731 26b00d76 5e28dc7f 01c5a54c
```

References

[1] P. Donescu V. Gligor. Infinite garble extension, November 2000.