

## Re: COM Interface Security

**Source:** <http://www.tech-archive.net/Archive/VB/microsoft.public.vb.com/2005-01/0113.html>

---

**From:** Ulrich Korndoerfer (*ulrich\_wants\_nospam\_at\_prosource.de*)

**Date:** 01/20/05

Date: Thu, 20 Jan 2005 03:14:28 +0100

Hi,

Joseph Geretz wrote:

- > *Where it is undesirable to allow other clients to use our public classes,*
- > *then we'll need to implement a bi-directional handshake. That is our own*
- > *software instantiates the class, passes in a 'public key' and then expects*
- > *the class to return an encrypted value which matches its own internal*
- > *private key. This would allow us to develop our software in a modular manner*
- > *but would prevent other developers from making use of our own proprietary*
- > *classes.*
- >
- > *How does that sound? Anyone done anything similar?*

There are several other ways too to prevent unauthorized users to use a dll.

For passing keys around, I agree with you that not the key should be the secret thing but the en/decryption algorithm used to en/decrypt using this key. This algorithms has to be implemented in both the client and the server. Authentication then could go as following:

- The client passes a random key (random in content and length) along with a random text to the server. The server encrypts the text with the given key and returns the encrypted text. The client decrypts the text and compares it with the original. If identical, the server is authenticated (at least one can say now that the server knows the encryption algorithm ;-).
- Then the client calls the server for a random key and random text. The client encrypts the text and passes it to the server. The server decrypts the text and compares it with the original. If identical, the client is authenticated (again at least one can say now that the client knows the encryption algorithm ;-).

The encryption algo could encrypt text to a text which has random length, which makes cracking the algorithm a little bit harder. For example it could include a date/time stamp in the key or the text which decides how long the encrypted text will be. Or use a checksum of the

random key or the random text to decide how long the encrypted text will be.

ARC4 would be an algorithm well suited. He can both provide the random keys and texts as well as doing the encryption/decryption. Mock it up with varying prefix runs and use some wild calculations to extract the real key embedded in the truly random key eg from information found in the random key. Do not use the random key as the real key in direct. At least do some transformation on it before using it as key for encryption. You eg. could permute all random key bytes to create the real key. For the permutation eg. use a 64 bit value (extract it to a variable of type decimal for calculating the permutation) which defines the permutation to be done (should be sufficient for 20 bytes or so). The permutation derived from a number is quickly done. The number eg. could be simply the first 8 bytes of the random key.

The more you scramble the random key to yield the real key the safer you are.

--

Ulrich Korndoerfer

VB tips, helpers, solutions -> <http://www.proSource.de/Downloads/>