

Re: How good an encryption algorithm is this?

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-11/4871.html>

From: Nicholas Paldino [.NET/C# MVP] (mvp_at_spam.guard.caspershouse.com)

Date: 11/22/04

Date: Mon, 22 Nov 2004 15:02:33 -0500

Bonj,

See inline:

- > 1) *Must be capable of encrypting strings to a byte array, and decrypting back again to the same string*

All encryption algorithms do this. If you couldn't decrypt back, then it would be a hash.

- > 2) *Must have the same algorithm work with strings that may or may not be unicode*

This should not be an issue. Encryption works on byte streams, not strings. As long as the string can be converted to/from a byte stream, you won't have a problem.

- > 3) *Number of bytes back must either be \leq number of `_TCHARs` in * `sizeof(_TCHAR)`, or the relation between output size and input size can be calculated simply. Has to take into account the null terminator on the end of the string.*

The null terminator at the end of a string is just another byte. If you encrypt it, it's going to take up space. If you don't need it, don't use it then. As for the output size, I believe that it is the same as the input size.

- > 4) *Encryption algorithm must also return the exact number of bytes of the encrypted data*

Do you mean that it will return the number of bytes it would take to encrypt the data? Since most algorithms return the same size, this won't be a problem.

I would recommend against rolling your encryption. Rather, you should use one of the classes in the `System.Security.Cryptography` namespace. If you need compression as well, then you can apply that after the encryption.

microsoft.public.dotnet.languages.csharp: Re: How good an encryption algorithm is this?

Hope this helps.

```
--
    - Nicholas Paldino [.NET/C# MVP]
    - mvp@spam.guard.caspershouse.com
> I was struggling to get the CryptoAPI to work, with CALG_RC4 it didn't
> encrypt at all, and with CALG_RC2 it didn't decrypt at all (both claimed
> to have succeeded, but with the former, the encrypted string was exactly
> the same as the input, and with the latter, the decrypted string was
> exactly the same as the encrypted string) - and I thought I'd done
> everything right (at the bottom if you reckon you can spot where I
> havent...)
>
> So I decided to invent my own algorithm, and I just wanted anybody's
> opinion on how secure this could be compared to the Win32 API version.
> First, a C# program generates an array of 256 bytes, as such:
> using System;
> using System.Security.Cryptography;
> class Class1
> {
> [STAThread]
> static void Main()
> {
> byte[] b_raw = new byte[256];
> RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
> rng.GetBytes(b_raw);
> Console.WriteLine("const BYTE b[] = {");
> for(int i = 0; i < 256; i += 8)
> {
> Console.WriteLine("0x{0:x}, 0x{1:x}, 0x{2:x}, 0x{3:x}, 0x{4:x}, 0x{5:x},
> 0x{6:x}, 0x{7:x}",
> b_raw[i], b_raw[i+1], b_raw[i+2], b_raw[i+3], b_raw[i+4], b_raw[i+5],
> b_raw[i+6], b_raw[i+7]);
> }
> }
> }
>
> This is run twice, so I have two arrays of 256 bytes, say key1 and key2.
> These are then hardcoded into the C++ encryption algorithm.
> Then, the C++ encryption algorithm goes as such:
> It memcpy's the _TCHAR array to a byte array, then loops round each byte of
> this array.
> For each byte, it gets the value of key1[n] (where n is the byte number),
> and calls this 'b_current_indir' (the starting 'indirection level').
> Then, it gets the value of key2[n] and calls this 'levels' - the number of
> indirection levels.
> Then, an inner loop runs 'levels' times - and on each loop the following
> happens: the current byte of the data to be encrypted (dictated by the
> outer loop) is XORed with key2[b_current_indir], and THEN, b_current_indir
> is reassigned to take on the value of key2[b_current_indir].
>
> The whole C++ algorithm is defined as such:
> void Decrypt(BYTE* orig, long bytelen, _TCHAR* dataout)
> {
> BYTE* b_temp = new BYTE[bytelen];
> #ifdef _DEBUG
> ZeroMemory(b_temp, bytelen);
> #endif
> for(long bytenum = 0; bytenum < bytelen; bytenum++)
> {
> BYTE levels = b[bytenum % 256],
> b_current_indir = b2[bytenum % 256]; //always starts the same
> b_temp[(long)bytenum] = orig[(long)bytenum];
```

Re: How good an encryption algorithm is this?

microsoft.public.dotnet.languages.csharp: Re: How good an encryption algorithm is this?

```
> for(long level = 0; level < levels; level++)
> {
>     b_temp[(long)bytenum] ^= b_current_indir;
>     b_current_indir = b2[(long)b_current_indir];
> }
> }
> memcpy(dataout, b_temp, bytelen);
> delete[] b_temp;
> }
>
> void Encrypt(_TCHAR* orig, long textlen, BYTE* dataout)
> {
>     long bytelen = textlen * sizeof(_TCHAR);
>     BYTE* b_temp = new BYTE[bytelen];
>     #ifdef _DEBUG
>     ZeroMemory(b_temp, bytelen);
>     #endif
>     memcpy(b_temp, orig, bytelen);
>     for(long bytenum = 0; bytenum < bytelen; bytenum++)
>     {
>         BYTE levels = b[(long)(bytenum % 256)],
>         b_current_indir = b2[(long)(bytenum % 256)];
>         for(long level = 0; level < levels; level++)
>         {
>             b_temp[(long)bytenum] ^= b_current_indir;
>             b_current_indir = b2[(long)b_current_indir];
>         }
>     }
>     memcpy(dataout, b_temp, bytelen);
>     delete[] b_temp;
> }
>
> int main()
> {
>     LPTSTR testtext = _T("TheMagicBonj");
>     long textlen = _tcslen(testtext) + 1;
>     BYTE* b_enc = new BYTE[textlen * sizeof(_TCHAR)];
>     #ifdef _DEBUG
>     ZeroMemory(b_enc, textlen * sizeof(_TCHAR));
>     #endif
>     Encrypt(testtext, textlen, b_enc);
>     _TCHAR* t_enc = new _TCHAR[textlen];
>     ZeroMemory(t_enc, textlen * sizeof(_TCHAR));
>     memcpy(t_enc, b_enc, textlen * sizeof(_TCHAR));
>     _tprintf(_T("The encrypted text is \"%s\"\n"), t_enc);
>     delete[] t_enc;
>
>     _TCHAR* t_dec = new _TCHAR[textlen];
>     Decrypt(b_enc, textlen * sizeof(_TCHAR), t_dec);
>     _tprintf(_T("The decrypted text is \"%s\"\n"), t_dec);
>     delete[] t_dec;
>
> }
>
> It seems to work to my eyes, but is it a pile of rubbish?
>
> My initial thoughts were that somebody could crack it by disassembling the
> machine code into assembly language, discovering where the global
> namespace section of the DLL file was and deriving key1 and key2 from
> that, and then just plugging them back through the algorithm, without
> necessarily understanding the algorithm from the assembly language. But
> even simpler than that, if they discovered that "it was probably *this*
```

microsoft.public.dotnet.languages.csharp: Re: How good an encryption algorithm is this?

```
> DLL that created *that* encrypted data, so if I find out how to call the
> DLL, I can decrypt it". To counter that, I could put the key in the
> application very easily. But does this help? Could someone with a
> knowledge of assembly language guess with a reasonable degree of accuracy
> which bit of data in a PE file was likely to be a key to an encryption
> algorithm?
> Wouldn't it be just as easy for someone wanting to crack it to still do
> that if I had written a DLL that used the Win32 API Crypto functions? If
> not, why not? Is the effectiveness of private key cryptography only as
> good as how well you can hide the key?
>
> Please give as many thoughts as possible...
>
> My (unsuccessful) attempt to use the Win32 API is as follows:
> BOOL GetData(_TCHAR* datain, long lendatain)
> {
> HCRYPTPROV hCryptProv;
> HCRYPTHASH hCryptHash;
> HCRYPTKEY hCryptKey;
> DWORD bytelen = (lendatain + 10) * sizeof(_TCHAR), databack = 0, bytesback
> = 0;
> BYTE* bData = new BYTE[bytelen];
> _TCHAR* cryptdata = new _TCHAR[lendatain],
> * decryptdata = new _TCHAR[lendatain];
>
> BOOL bSuccess = CryptAcquireContext(&hCryptProv, keysetname, NULL,
> PROV_RSA_FULL, CRYPT_NEWKEYSET);
> if(!bSuccess) bSuccess |= CryptAcquireContext(&hCryptProv, keysetname,
> NULL, PROV_RSA_FULL, 0);
> memcpy(bData, datain, bytelen);
> bSuccess &= CryptCreateHash(hCryptProv, CALG_MD5, 0, 0, &hCryptHash);
> bSuccess &= CryptHashData(hCryptHash, (BYTE*)textkey, _tcslen(textkey),
> 0);
> bSuccess &= CryptDeriveKey(hCryptProv, CALG_RC2, hCryptHash, 0,
> &hCryptKey);
> bSuccess &= CryptEncrypt(hCryptKey, hCryptHash, TRUE, 0, bData,
> &bytesback, bytelen);
> CryptDestroyKey(hCryptKey);
> CryptDestroyHash(hCryptHash);
> CryptReleaseContext(hCryptProv, 0);
> memcpy(cryptdata, bData, bytesback);
> _tprintf(_T("The encrypted data is \"%s\"\n"), cryptdata);
>
>
> bSuccess = CryptAcquireContext(&hCryptProv, keysetname, NULL,
> PROV_RSA_FULL, CRYPT_NEWKEYSET);
> if(!bSuccess) bSuccess |= CryptAcquireContext(&hCryptProv, keysetname,
> NULL, PROV_RSA_FULL, 0);
> bSuccess &= CryptCreateHash(hCryptProv, CALG_MD5, 0, 0, &hCryptHash);
> bSuccess &= CryptHashData(hCryptHash, (BYTE*)textkey, _tcslen(textkey),
> 0);
> bSuccess &= CryptDeriveKey(hCryptProv, CALG_RC2, hCryptHash, 0,
> &hCryptKey);
> bSuccess &= CryptDecrypt(hCryptKey, hCryptHash, TRUE, 0, bData,
> &bytesback);
> memcpy(decryptdata, bData, bytesback);
> databack = (DWORD)(bytelen / sizeof(_TCHAR));
> _tprintf(_T("The decrypted data is \"%s\"\n"), decryptdata);
>
> CryptDestroyKey(hCryptKey);
> CryptDestroyHash(hCryptHash);
> CryptReleaseContext(hCryptProv, 0);
```

Re: How good an encryption algorithm is this?

microsoft.public.dotnet.languages.csharp: Re: How good an encryption algorithm is this?

```
>
> delete[] bData;
> delete[] cryptdata;
> delete[] decryptdata;
> CryptDestroyKey(hCryptKey);
> CryptDestroyHash(hCryptHash);
> CryptReleaseContext(hCryptProv, 0);
>
> return bSuccess;
> }
>
> void test(_TCHAR* string)
> {
>   GetData(string, _tcslen(string));
> }
>
> int _tmain()
> {
>   test(_T("TheMagicBonj"));
>   return 0;
> }
> The output is:
>
> The encrypted data is "^ǒnÍ&lÕ eMagicBo "
> The decrypted data is "@ "
>
> which is disappointing.
>
```