Subject: Re: StreamCypher - A package for stream data cryptography
Posted by mdelfede on Thu, 30 Sep 2010 22:58:39 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Thu, 30 September 2010 23:44
.....
Encrypt/Decrypt fits better than Encode/Decode.
But I must disagree about Decode. There is a problem if user calls Encrypt and then Decrypt. I have two classes: one for decryption and other for encryption. They have different internal mechanisms and I don't really know why they should be merged into one class. The general problem is that encryption and decryption are really different processes in AES and they cannot be called one by one (especially if we want streaming).
My proposal is to have Encryptor class and Decryptor class (they might be the same in your case).

Encrypt() and Decrypt() are the same process in Snow/RC4, that's true, but they can't really mixed without resetting the machine state, so calling SetKey().
I'd like to keep both in a single class and throw an exception if mixing Encrypt/Decrypt calls without key resetting... but we can speak about it, of course.
That's just because of usage simplicity, you'd need just an Snow/AES class and call Encrypt/Decrypt, instead have both SnowEncrypt/SnowDecrypt classes and call, for example, Process() on both.

A "mixed" solution would be to have Snow class with both Encrypt()/Decrypt(), an AESEncrypt with a working Encrypt() and a Decrypt which just throws an exception and the way around for AESDecrypt class.
I think that the better solution depends on how is built the state machine for AES, which I don't know. If machines are the same, and just the encoding/decoding processes are different, I'd vote for a single class.... If also the state machines are different, I'd vote for 2 classes.

Quote:
Quote:String MD5Key(String const &); Latest investigations revealed potential weaknesses in MD5 hashing. That is why I used SHA256 instead of MD5 for internal key generation. So I agree with you in general (yes, we must generate hash from user password as internal crypto-key), but i suggest using SHA256 instead of MD5. Function can be found in AESStream.cpp @ 42:
String AESHashedString(const String &s)

Agree, MD5 was just an example, your's is better

Quote:
Quote:I thought also on removing the constructor with empty params, but it can be useful on reusing the encoder : you can create an encoder without key and then use multiple SetKey(). I suppose it could make more problems than solve them. SetKey() makes a danger of changing the key while en/decryption streaming is in progress. That is why I denied any possible empty constructors. My thought was to make my classes stable by design. It could be good to hear suggestions from AESStream users, is SetKey() really good here.
Finally, if we agree with SetKey(), I will add it for the sake of uniformness. But I'd warn us against creating any potential problems.

Well, Setkey() is maybe not good when streaming, but is indeed comfortable for block encoding/decoding with different keys.
Of course, SetKey() should reset all mechanics, including streaming state and buffers.
Anyways.... SetKey becomes useful also when re-using the class for encoding/decoding, which I find also comfortable behaviour.

Quote:
Quote:BTW, I shall also add an initialization number, I guess that it's a must for stream cypher If you mean initial vector for streamed encryption, I must say that only good way to make it is to use OpenSSL's random number generation. There was a number of investigations about initial vector. In short, the conclusion was: NEVER use constant initial vector. The best initial vector is random vector. That is why I generate random initial vector in constructor without any doubts.

We can't assume that user can accept a random number appended to its data. Just think to disk sector encoding, for example. Size of input data must be identical to size of output data.
In my Protect package it's foreseen to use random data as the 'nonce' vector, but it's separated from encoded block. There are cases in which you have to assume a constant or an user-assignable initialization vector kept outside of data stream. Also encoding a block in-place would be impossible if data sizes don't match.
So, IMHO, we should make it optional somehow, for example with a constructor parameter/SetKey parameter which can be Null by default meaning a random init vector, and provide a GetIV()/SetIv() to get or set it if needed.
The defaults could be different from AES and Snow/RC4, and streaming of 'nonce' vector could be made optional by constructor or member function also.

So, if you want to keep your AES compatible with former version, you could make it on by default in your class, and I could make it off in mines.

Quote:
Quote:But I just don't understand the need of the datasize parameter. Data size is written into the header of my protected container. So decryptor "knows" the overall size of data even if user doesn't (that's good feature). Besides, streamed decryption becomes more robust as it doesn't decrypt more data than needed.

what about an eof() on class ?
Stream crypto is foreseen for variable/unknown stream sizes, so it should anyways have an IsEof() function for streaming.
And also for your class it would be good and allow to retrieve just the encoded data size letting your class to bother about it.

Quote:
Quote:Another stuff... I've seen in your docs that your encoder is 'not reusable', I mean you have to create a new one if encoding a new stream.
I think it would be better to add some reset mechanics inside the (newly added) SetKey() member, which would allow re-keying and resetting the encoder...
Adding a simple Reset() would be dangerous, as it would need to store the key or at least the

S-Box, which could lead to easy keyfinding inside a running app 'Not reusable' is a drawback of design I've chosen (see above). It is discussable of course.
Talking about Reset(), yet we have to keep the key in memory if we support streaming. So we have to think how to crypt the key in memory to avoid plain keyfinding. Any ideas are welcome here.

No, I think we have to keep just the S-Boxes in memory, which is far safer. At least, for Snow and RC4, but I guess for all cases should be the same. I don't want the key hanging around for more time than strictly needed

Quote:
Quote:About the name... what do you think about 'Cypher' ? It doesn't make big difference for me. Maybe, if we use Encryptor/Decryptor name, we could call package Crypto. If you don't like it, please fill free to use Cypher or anything else.

Well, both Cypher and Crypto are equally good

Quote:Last but not least... as we're making a general Cypher package, what about asymmetric-key encoding like RSA ? I would not assume highly different crypto scheme to use the same interface. I propose making what we planned before, catch any bugs there. And only after that to think about RSA. I used it about 10 years ago and as I remember it should use different scheme with a different interface.
[/quote]
ok, that was just for planning the right interface, common to all if possible.