Date: Thu, 15 Apr 1999 18:17:51 +0200
From: Louis Granboulan <Louis.Granboulan@ens.fr>
To: AESFirstRound@nist.gov
Cc: maro@isl.ntt.co.jp, Serge <Serge.Vaudenay@ens.fr>
Subject: Analysis of the RefCode and OptCCode submissions
Reply-To: Louis Granboulan <Louis.Granboulan@ens.fr>
X-Mailer: Mutt 0.95.3i

Here is official comment I have written about the C programs
submitted for AES CD2. This is a technical view point on the
API correctness and portability of these programs.

Best regards,
Louis Granboulan

AES : Analysis of the RefCode and OptCCode submissions

Louis Granboulan

April 15, 1999

Abstract

In this document, I review all AES submission from a C programmer's point of view. I check if they correctly implement AES API and if they really are portable ANSI C.

1 API correctness

	Cast 256	Crypton	DEAL	DFC	E2	Frog	HPC	Loki 97	Magenta	Mars	RC6	Rijndael	Safer +	Serpent	Twofish
Headers	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	no	OK	OK
`makeKey`	OK	no	no	OK	OK	OK	no	OK	OK	no	OK	no	OK	OK	OK
`cipherInit`	no	no	OK	no	OK	OK	no	OK	OK	no	OK	no	OK	OK	OK
`blockEncrypt`	OK	OK	OK	no	OK	no	OK	no	OK	OK	OK	OK	no	OK	OK
Library	OK	no	no	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	no	no
ECB	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
CBC	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
CFB1	OK	no	OK	OK	OK	no	OK	OK	OK	no	OK	OK	OK	OK	OK

1.1 Can we easily make a cryptographic library having NIST API?

The NIST specified an ANSI C interface [2] that should be followed by the submissions in the CD [1]. I tried to build a Unix library with the files from AES CD, either RefCode or OptCCode submissions. I asked that building this library should be as easy as possible. I checked the following points :

- Does it respect API values in header files
  In the NIST API document, there is an example of a header file for AES API, that give some numeric values for error codes. There is the possibility to put additional error codes if needed. All candidates respect the values defined in this file and give other values to their additional error codes, with the exception of
  - Safer + : it adds the error code `BAD_KEY_LEN` which changes the values of NIST defined error codes `BAD_KEY_INSTANCE`, `BAD_CIPHER_MODE` and `BAD_CIPHER_STATE`.
- The function `makeKey`
  The NIST API document describes the structure `keyInstance` that will store all the key information. Since nothing in the API allows to free pointers referenced in this structure, it should contain only scalar or arrays if we don't want to fill the memory [1].
  - Crypton misunderstood the specification and handles `keyMaterial` as a binary value instead of an ASCII string.

[1] But the `aes.h` example shows as an example the `BYTE *KS` pointer to a key schedule. This should not be followed!

- DEAL does not allocate the memory needed for the pointer `key->kss`, and it is never freed (cf. the former remark).
- HPC allocates three blocks of memory `key->KS`, `key->hpc_kmbits` and `((U64**)key->KS)[HPCM]` that may never be freed (cf. the former remark).
- on success, MARS returns `0` instead of `TRUE`.
- Rijndael has an additionnal argument that is not compatible with the prototype in the API : `blockLen`.

- The function `cipherInit`
  The NIST API document ot fair here... the parameter `IV` is a string that is translated into an array of `BYTE` and stored in the `cipherInstance`. But the parameter `keyMaterial` for `makeKey` was stored unchanged in the `keyInstance`.
  - Cast 256 (RefCode only) needs that we set `cipher->verboseoutfile` to `NULL` before calling `cipherInit`.
  - Crypton misunderstood the specification and handles `IV` as a binary value instead of an ASCII string.
  - DFC does not understand `NULL` or non valid `IV` in ECB mode, but it should not use this parameter.
  - HPC allocates one block of memory that cannot be freed and we need to set `cipher->blockSize` to 128.
  - on success, MARS returns `0` instead of `TRUE`.
  - Rijndael has an additionnal argument that is not compatible with the prototype in the API : `blockLen`.

- The function `blockEncrypt` The parameter `inputLen` is the size of the input in bits and the return value is the number of bits encrypted.
  - DFC understands the parameter `inputLen` in bytes instead of bits (mimics the Java API).
  - Frog, Safer + and Loki 97 return `TRUE` instead of the number of bits encrypted.

- The compilation of the library
  - for Crypton and Twofish, the file that defines all the API routines also defines the `main` procedure. You have to delete it to make a library that can be linked with other programs.
  - DEAL is not consistent for the case of filenames : `deal.c`, `dealref.c` and `dealkeys.c` include `DEAL.h` but `dealapi.c` includes `deal.h`.
  - Serpent's header file `serpent-api.h` in RefCode defines a macro named `r` with value 32. This should be avoided because of possible conflicts with variable names or struct members, for example in standard include header files.

1.2 Can we decrypt an encrypted message?

I encrypt a random 128 bits blocks with a random 128 bits key and then decrypt the result and check the equality.

- ECB mode
  All candidates are OK.

- CBC mode
  The NIST API says that `cipherInit` stores the initialisation vector in the `cipherInstance`. It is not clear if encryption or decryption should change the initialisation vector in the `cipherInstance`.
  - Cast 256, Crypton, DEAL, Frog, HPC, RC6, Safer + and Twofish change the initialisation vector.
  - DFC, E2, Loki 97, Magenta and Mars don't change the initialisation vector.
  - Crypton RefCode does not decrypt well, but the NIST did only require OptCCode implementations of CBC.
- CFB1 mode
  Note that the NIST API cannot be used to decrypt in CFB1 mode, since you need to use the `DIR_ENCRYPT` key schedule with the `blockDecrypt` function and that is forbidden (returns a `BAD_KEY_MAT`).
  I changed the API to allow `DIR_ENCRYPT` key schedule (and only this one) for `blockDecrypt` in CFB1 mode. I made the tests with this modified API.
  DEAL and FROG implementors had already noticed this problem.
  - Crypton : neither RefCode nor OptCCode decrypts a 128 bits encrypted sequence.
  - Frog changes only the first 8 bits for a 128 bits message, and the decryption code is missing a `break` to have a valid return value.
  - Mars accepts CFB1 encyption only for 1 bit messages.
  - with Rijndael, only OptCCode has CFB1 mode, since the NIST did not require RefCode implementations of CFB1.
  - with Serpent and Twofish, only RefCode decrypts well, but the NIST did require OptCCode implementations of CFB1.

2 ANSI conformance and portability

	Cast 256	Crypton	DEAL	DFC	E2	Frog	HPC	Loki 97	Magenta	Mars	RC6	Rijndael	Safer +	Serpent	Twofish
RefCode	OK	no	no	OK	OK	OK	OK	no	OK	OK	OK	OK	OK	OK	OK
OptCCode	OK	no	OK	OK	OK	OK	OK	no	OK	OK	OK	OK	OK	OK	OK
Endian & align	OK	OK	OK	OK	no	OK	no	OK	OK	no	no	no	OK	no	OK
Word length	OK	OK	OK	OK	no	OK	no	OK	OK	no	no	OK	OK	OK	OK
Portable	no	no	OK	OK	no	OK	no	OK	OK	no	no	no	OK	no	no

2.1 Is it strict ANSI?

Many compilers understand a superset of ANSI C. We test ANSI conformance with the `lcc` strict ANSI compiler [3] with options `-A -A`. Sun's compiler with options `-v -Xc` give some additional warnings (constant correctness).

- Crypton mixes `char` and `unsigned char` as if they were the same type. Some functions are defined with old-style K&R argument declaration.
- DEAL declares `xor8` and `copy8` as extern and then defines them as static (file `dealref.c`. We also notice that local `int k` is defined in `generateRandomKey` but never used (file `dealapi.c`.
- E2 negates an unsigned value (function `e2ModularInverse` in file `r-e2.c`).

- with HPC, we notice that local `ul64 difference` is defined in `subEq_ul64` and local `int nv` is defined in `strtoU64` but are never used (file `hpc-ansi.c`).
- Loki 97 defines static function `puthex` as returning an `int` but it never returns a value. Some prototypes are missing and other are old-style K&R. The constant `DELTA` is missing `const` qualifier.
- for Safer +, some prototypes are missing and other are old-style K&R.
- Serpent defines the function `hex` that may call `exit` and then return no value (file `serpent-aux.c`). It defines `serpent_encrypt` and `serpent_decrypt` as returning and `int` but they never return a value. (file `serpent.c`). Many prototypes are missing, some local variables are not used.

2.2 Do we have problems with endianess and word length?

The API represents the cleartext and ciphertext as `unsigned char *`, which representation is clearly defined by ANSI. The output on the encryption function should be independant of the internal representation of integers. Many AES candidates suppose that they can have 32-bits integers, but ANSI standard only requires that `long` are at least 32-bits long. No integer type is garanteed to be exactly 32-bits. Luckily, most AES candidates have a `typedef` definition that is easy to set to some 32-bits integer type.
Many AES candidates use "casts" to convert data types. They suppose that the internal representation of characters and integers follow some particular rules. This may cause problems if the system is little endian or big endian or something even stranger. Some AES candidates check their hypothesis about endianess by looking at the internal representation of integers, but their codes are not likely to run on a PDP11 box (I don't have one, so I did not the test). The best solution is to avoid casts that are not specified by the standard.

- Endianess and alignment
  - for Cast 256, you need to change the macro `littleendian` in `cast.h`.
  - Crypton (OptCCode) the function `CryptonExpandKey` casts `BYTE*` to `DWORD*`, but this does not cause alignment problems on the computers I used.
  - E2 (OptCCode) the macros `LOAD_L_HIGH` and many others in `defcode.c` makes dangerous casts : dumps a core on many architectures (alignment problem).
  - HPC does not give the same encrypted messages, depending on endianess.
  - Mars casts `BYTE*` to `DWORD*` and this can cause alignment problems (function `cipherInit`). There is a test in `sbox.c`, `mars-ref.c` and `mars-opt.c` that defines (or not) `SWAP_BYTES` and that you need to update to deal with endianess.
  - RC6 (OptCCode) give different results depending on endianess. The file `aes.c` says that *"We'll make use of the fact that Intels are little-endian."*
  - Rijndael makes some casts from `word8*` to `word32*` that may cause alignment problems.
  - Serpent give byte-swapped encrypted messages depending on endianess.
  - Twofish make many casts to do byte-swap and other things, and this may cause alignement problems. You have to adapt `platform.h` to you architecture and define `LittleEndian` if needed. If we compile Twofish with the bad endianess definition, the function `makeKey` returns `BAD_KEY_MAT`, which is a strange behaviour!

- Word length
  - Cast 256 supposes that the type `uns32` in `cast.h` is exactly 32-bits long.
  - Crypton supposes that the type `DWORD` in `crypton.h` is exactly 32-bits long.
  - for E2, you need to change the type `uint32` in `r-e2.h`. But with OptCCode, it is not a solution because they suppose `uint32` and pointers have same size.
  - HPC suppose that if you did not define `HAVE_64bit_LONG`, then `unsigned long` are exactly 32-bits long.
  - Mars supposes that the type `WORD` in `aes.h` is exactly 32-bits long.
  - RC6 need to check that all occurence of `long` is a 32-bits integer type. This makes RefCode work. For OptCCode, we need to check 16-bits types too... this is nearly impossible to make work.
  - Rijndael supposes that the type `word32` in `rijndael-alg-fst.h` is exactly 32-bits long.
  - Serpent (OptCCode) need to check that all occurence of `long` is a 32-bits integer type.
  - Twofish supposes that the type `DWORD` in `aes.h` is exactly 32-bits long. You also need to change `int` to an integer type of the same size as pointers for all casts in `twofish.c` (test for checking alignment in RefCode).
- Portability
  Best programs are the ones which don't need any change if the internal representation of integers changes. So I test if the type `unsigned long` can be 64-bits or f `unsigned short` can be 32-bits. I did not make exhaustive tests, for example negative integers could have an other representation than their 2's complement...
  - Crypton does special optimizing hacks if `_WIN32` is defined.
  - Twofish does special optimizing hacks if `_M_IX86` or `_MSC_VER` or `__BORLANDC__` are defined. There is a `ALIGN32` macro that should be defined if the cpu needs data alignment.

2.3 Conclusion

It is not easy to write a C program that respects the AES API, that has the same behaviour on all machines that that is an efficient cryptosystem. Only Magenta did a perfect work, all other submissions had major or minor faults. Magenta is the slowest and probably the less secure of the submissions.

People should not give too much importance to the comparison of the performance of C programs, because they are not really more portable than assembly.

References

[1] NIST – AES CD2

[2] NIST – ANSI C Cryptographic API Profile for AES Candidate Algorithm Submissions Revision 5: April 15, 1998

[3] `lcc`, a retargetable compiler for ANSI C (version 4.0) `http://www.cs.princeton.edu/software/lcc/`