PB173 - Tématický vývoj aplikací v C/C++ (podzim 2013)

Skupina: Aplikovaná kryptografie a bezpečné programování

https://is.muni.cz/auth/el/1433/podzim2013/PB173/index.qwarp?fakulta=1433;obdobi=5983;predmet=734514;prejit=2957738;

Petr Švenda svenda@fi.muni.cz

Konzultace: G.201, Úterý 13-13:50

CROCS
Centre for Research on Cryptography and Security

Portability and memory restrictions

Memory restrictions

- Size of the code vs. runtime memory requirements
- Depends on the target platform
  - usually of little concern (RAM is big enough)
  - sometimes critical factor for algorithms selection
    - embedded devices, e.g., sensor nodes
- Algorithms usually provides possibility for optimization
  - precomputed tables – speed vs. memory
  - key schedule vs. on-the-fly key schedule
  - optimizations may increase risk for side channel attacks
- Write correct code first, then optimize
  - especially true in security

Portability – different operating systems

- Usually no problems with algorithms
  - plain C code
- Problems with additional functionality
  - read file, directory listing, user input, GUI
  - often cannot be solved by standardized functions or POSIX
  - abstract and separate platform-dependent functions
    - move them into distinct modules
    - easy to replace/extend for target platform later
- Data generated by your application should be portable
  - ASN.1 encoding
  - TLV encoding
  - binary vs. text formats
  - Base64 encoding

Portability – different hardware platforms

- Little vs. big endian architecture
  - usually problem with bit-based operations
  - e.g., bit rotation
  - problem with interpretation of binary formats
- Highly optimized implementations
  - e.g., Gladman
  - may use architecture specific operations and behaviour
  - multiple byte operations in single tick
  - special representation of memory data
  - may use macros heavily

Reference vs. optimized version

- Double meaning of “reference” word
  - reference implementation from algorithm designers (Rijndael)
  - reference == code you should use
- Reference implementation (e.g., Rijndael)
  - usually simple and understandable API
  - lower performance
  - may not protect against implementation attacks
  - typical usage is as supplementary material to algorithm description document
  - is used to create test vectors

Reference vs. optimized version (2)

- Optimized version of algorithm
  - same results as reference implementation
  - portability usually impacted
- Techniques used
  - pre-computed tables often
  - may use whole size of the architecture registers
    - e.g., AES is byte oriented, but x64 can perform eight xor of single byte per tick
  - may use special instruction of particular CPU
  - may use specifics of target architecture (e.g., cache size)
- Typically for the production environment

Choosing the right length

Length of keys/block/hashes

- Choose length with some reserve
  - many things can go wrong
- Choose algorithms with corresponding lengths
  - key derivation by MD5 of keys for AES256?
- Do not protect keys distribution by keys with lower entropy
  - AES key encrypted by simple DES key
- Asymmetric keys length needs to be much longer
  - space of possible values is not continuous

Bits of security	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
80	2TDEA[19]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Source: NIST SP800

Recommended key sizes

Algorithm security lifetimes	Symmetric key algorithms (Encryption & MAC)	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
Through 2010 (min. of 80 bits of strength)	2TDEA[23] 3TDEA AES-128 AES-192 AES-256	Min.: $L = 1024$; $N = 160$	Min.: $k=1024$	Min.: $f=160$
Through 2030 (min. of 112 bits of strength)	3TDEA AES-128 AES-192 AES-256	Min.: $L = 2048$ $N = 224$	Min.: $k=2048$	Min.: $f=224$
Beyond 2030 (min. of 128 bits of strength)	AES-128 AES-192 AES-256	Min.: $L = 3072$ $N = 256$	Min.: $k=3072$	Min.: $f=256$

Source:
NIST SP800

Symmetric key cryptography

- Key length for symmetric cryptography
  - 80 bits not secure enough against brute-force
  - always good to have some reserve for algorithm flaws
    - flaw => key can be found faster then by brute-force
    - AES-128 is still OK
    - AES-256 do not have 256 bits of security
- Take your application needs into account!

Making the keys

- From what are you making the keys?
  - password must have entropy equivalent to derived key
  - e.g., AES-128 key derived from “hello” will not have 128 bits security
- What if you create two keys from one with 128 bits of entropy?
- Do you really have perfect random generator?
  - 128 generated bits will not have 128 bits of entropy
  - generate more bits and use hash function to condense into 128 bits
- *(2013 - Seems that NSA was involved in intentional crippling of random generators – implementation and even standards)*

Asymmetric cryptography

- RSA is still gold standard
  - use (at least) 2048 bits keys
  - 768 bits broken by brute-force
  - special number with 1024 bits broken by brute-force
  - 1024 bits not broken yet, but…
- Elliptic courve cryptography (ECC) seems cool
  - *Currently (2013), some doubts about ECC security based on leaked Snowden documents arise*
  - But do you really need shorter keys?
  - You will face harder portability, more coding problems, lower level of code testiness etc.

Practical assignment

Practical assignment (2)

- Write following simple unit tests:
  - file not exists or cannot be read/written into
  - encrypted blob was corrupted
  - wrong decryption key was used
  - test vectors for encryption and hashing
- Any UT framework (UnitTest++, MinUnit, CxxTest...)
- Code will be used later in architecture
  - will be used again and extended, so write it well
- Best practices
  - http://blog.stevensanderson.com/2009/08/24/writing-great-unit-tests-best-and-worst-practises/
  - MinUnit http://www.jera.com/techinfo/jtns/jtn002.html

Questions?