PB173 - Tématický vývoj aplikací v C/C++
Domain specific development in C/C++

Skupina: Aplikovaná kryptografie a bezpečné programování

https://is.muni.cz/auth/predmety/uplny_vypis?fakulta=1433;obdobi=6384;predmet=871304

Petr Švenda svenda@fi.muni.cz

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
  - may use architecture specific operations and behavior
  - 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)

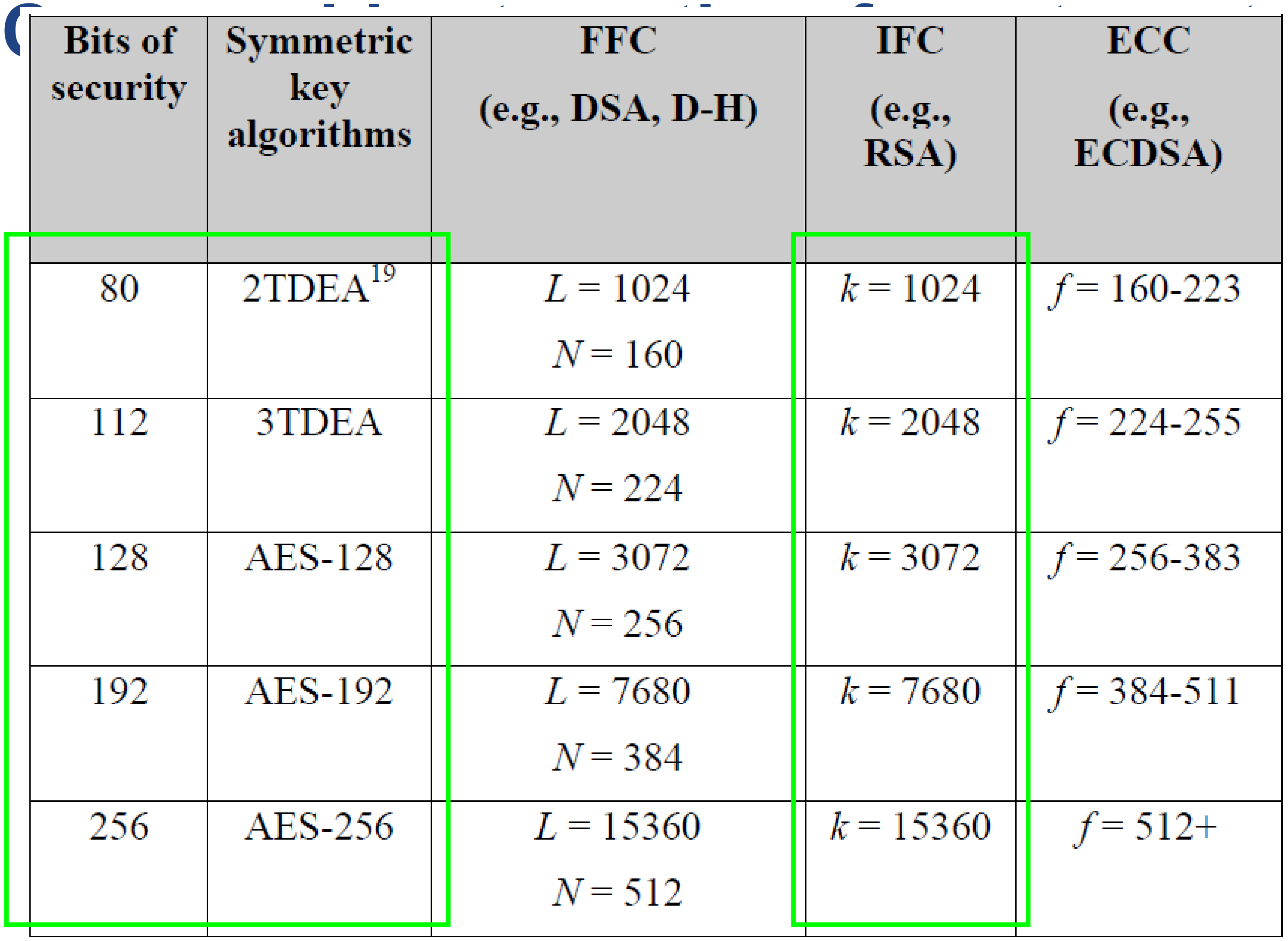
- Optimized version of algorithm
  - same results as reference implementation
  - portability may be impacted
- Techniques used
  - pre-computed tables often
  - may use whole size of the architecture registers
    - e.g., AES is byte oriented, but x64 can perform eight xor of single byte per tick
  - 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 SHA-1of keys for AES256?
- Do not protect keys distribution by keys with lower entropy
  – AES key (128b) encrypted by simple DES key (56b)
- Asymmetric keys length needs to be much longer
  – space of possible values is not dense

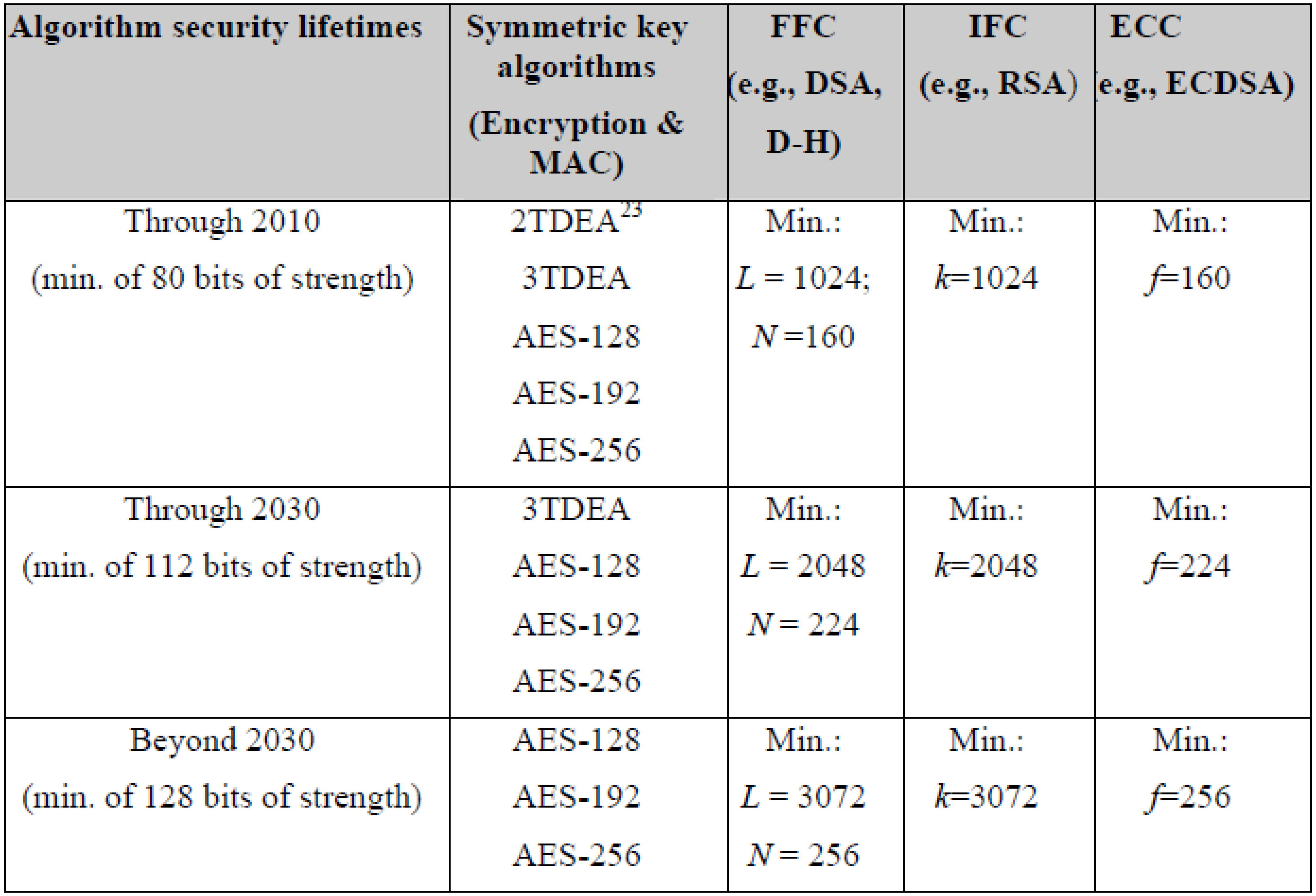
ms



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 now 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 full 256 bits of security, slightly less
- Quantum computers should be able to decrease the search space from 2^n to $2^{n/2}$ (for symmetric crypto)
  - AES-128 => 2^{64} => will be broken by bruteforce
  - AES-256 => 2^{128} => should be OK even with quantum computers
- 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
- *Seems (Snowden) that NSA was involved in intentional crippling of random generators (Dual_EC_DRBG) – implementation and even standards*

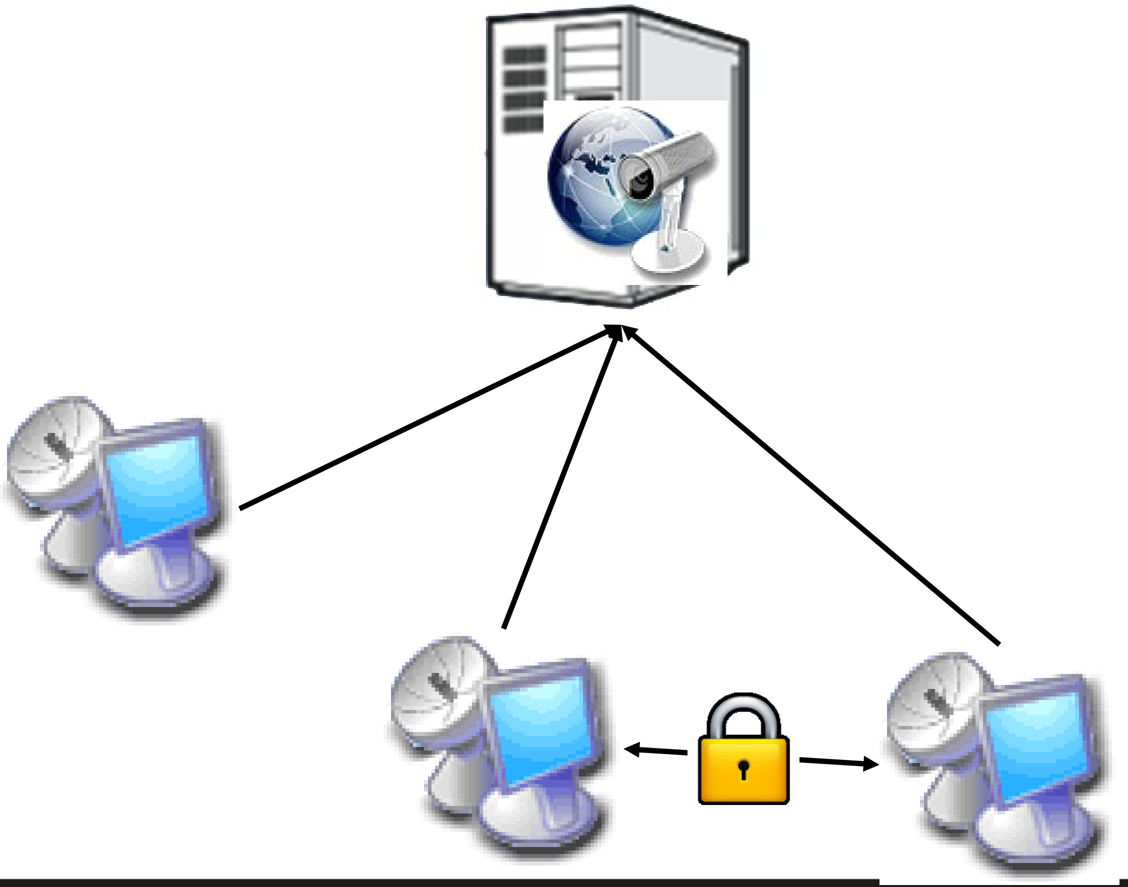
Asymmetric cryptography

- RSA is still “gold” standard
  - use (at least) 2048 bits keys, preferably 4086b
  - regular 768b and special 1024b broken by brute-force
  - regular 1024 bits not broken yet, but…
- Elliptic curve cryptography (ECC) now widely used
  - Shorter key lengths, faster operation
  - Who and how generated the curve you are using is (security) critical
- But:
  - TLS1.3 dropped support for RSA
  - NSA now (2015) discourage use of ECC “...elliptic curve cryptography is not the long term solution many once hoped it would be.”
- Post-quantum cryptography
  - Should withstand quantum computers, not mature yet

THE PROJECT

"Theme" project – Secure IM

- Secure instant messaging and data sharing



"Theme" project – Secure IM

- IM server
  - register and facilitate connection between users
- Client
  - provides operations related to end user usage

- Expected at the end: working networking application with security features

Teams

- 3 persons
- Joint work, but every one presents its contribution
  - Presentation on next seminar
- Use GitHub + Travis CI
- Form the teams now!
  - Team A: ???
  - Team B: ???

Practical assignment

1. Select great name for your group
2. Setup development workflow for your group
   - New Github repository (separate folder for client&server), license
   - Add proper developer rights
   - Initial version of code/tests (just copy from last lecture)
   - Try to pull/commits together and create conflict (intentionally)
3. Send me link to your repo (+ members of the group)
4. Plan your work together
   - Design based on requirements gathered now
   - Add initial issues into separate milestones

Practical assignment – cont.

4. Prepare document and presentation with design decisions
   - 2-3xA4 document (overview, functions, crypto used...)
   - 4-5 slides (presentation)
   - UML diagrams?

- Your design will be presented and discussed next week (13.3.)

Practical assignment

- Design and document API to:
  - new user registration
  - user authentication to server
  - obtain list of other users
  - establish secure channel to other (online) users (ENC, MAC)
  - exchange instant messages (small data packets) with other user
  - close secure channel
  - disconnect user from server
  - ...?
- Document functions in JavaDoc-style (Doxygen)
- Client/IM Server are separate processes
  - Plan for (later) communication over sockets or http requests

Principles of good API

1. Be minimal
2. Be complete
3. Have clear and simple semantics
4. Be intuitive
5. Be easy to memorize
6. Lead to readable code

- read more at e.g., *http://doc.trolltech.com/qq/qq13-apis.html*
- security API even harder: *http://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c18.pdf*
- *http://blog.apigee.com/taglist/security*

Submissions, deadlines

- Upload application source codes as single zip file into IS Homework vault (Crypto - 3. homework (UT))
  - Initial version of project at GitHub, structure, initial code&tests
  - Design documents (doc folder)
  - Header files with documented (Doxygen) function headers (API)
  - Slides for presentation (will take place 13.3.)
- DEADLINE 13.3. 12:00
  - Up to 10 points assigned

Questions?