Crypto libraries introduction

Milan Brož
xbroz@fi.muni.cz

PV181, FI MUNI, Brno

Open source cryptographic libraries

- Linux environment – up to you:
  - Debian / VirtualBox VM (see course materials)
    - **some optional examples need OpenSSL 3.0**
  - Your own distro – need to install development env.:
    - libgcrypt: Fedora: **libgcrypt-devel**; Debian/Ubuntu: **libgcrypt20-dev**
    - OpenSSL:Fedora: **openssl-devel**; Debian/Ubuntu: **libssl-dev**
    - libsodium:Fedora: **libsodium-devel**; Debian/Ubuntu: **libsodium-dev**
  - aisa.fi.muni.cz (OpenSSL v1 only)
- All examples in C language
- Home assignments (10 points each)

Lab environment
VirtualBox image

- Unpack zip archive from IS
- Open VirtualBox (click **blue** icon – config file)
- Login and password is **pv181**
  (same for **sudo** and **root** password)
- Scripts to switch OpenSSL 1.1.x / OpenSSL 3.0.x
  - see /home/pv181 directory
- Examples on gitlab

```
git clone https://gitlab.fi.muni.cz/xbroz/pv181.git
make clean; make; ./example
```

Cryptographic libraries Goals for this lab

- Crypto libraries and API / abstraction
- More practical and implementation view
- Why legacy code, compatibility and standards
- Coding practices – in C language
- Defensive approach: **It will fail, be prepared for it :-)**

Why not use a modern language with garbage collection and functional programming and free massages after lunch?
Here's the answer: Pointers are real. They're what the hardware understands. Somebody has to deal with them.
You can't just place a LISP book on top of an x86 chip and hope that the hardware learns about lambda calculus by osmosis.
- James Mickens, https://www.usenix.org/system/files/1311_05-08_mickens.pdf

Why implementation matters

- It works, but …
- How many possible bugs do you see?

```
/* Read a key from Linux RNG */
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
    int fd;
    char key[32];

    fd = open("/dev/random", O_RDONLY);
    read(fd, key, 32);
    close(fd);
    /* Do something with the key[] */
    memset(key, 0, 32);
    return 0;
}
```

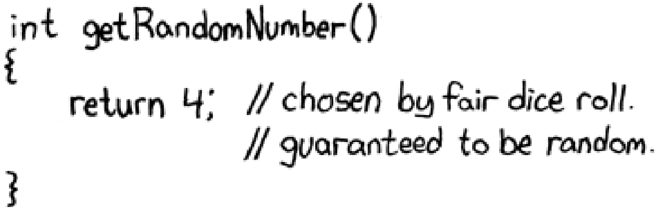
Why implementation matters

- How many possible bugs do you see?
  - *No check for return code, open(), read()*
  - *Posible reading from invalid fd (no random at all)*
  - *Partial read() is not detected*
  - *Failed read() is not detected*
    (mandatory access control can block reading)
  - *Magic numbers (one constant on several places)*
  - *Compiler can optimize memset() out*
    (secret key remains in memory)
  - *No error exit code, cannot check for failure*

Why implementation matters

- Fixes? Let’s see **example 0** in git.

- It is better to use a crypto library.
- Usually, maintainers implement it correctly :-)

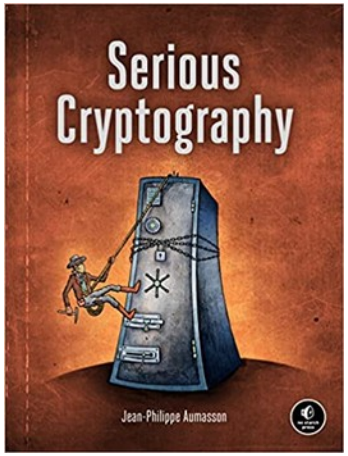


https://xkcd.com/221/

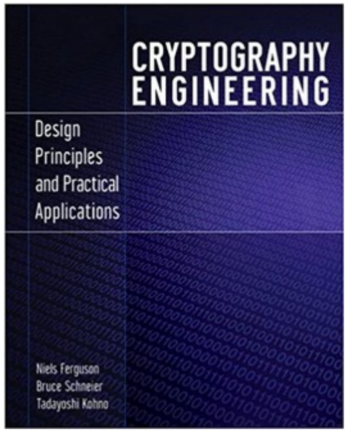
Some (not too old) books

More practically oriented books:

- *Jean-Phillipe Aumasson*
  Serious Cryptography:
  A Practical Introduction
  to Modern Encryption (2017)



- *Ferguson, Schneier, Kohno*
  Cryptography Engineering:
  Design Principles and Practical
  Applications (2010)



Cryptographic libraries Introduction

- Open-source / Proprietary
- Static + embedded / dynamically linked
- Low or high level abstractions
- Multiplatform
- Stable API and ABI
- Security or platform specific features
  - Safe memory use, side-channel resistance, …
  - HW acceleration support, “secure” HW support

Example libs (C and Linux) abstraction from low to high

- **Nettle**
- **libgcrypt**
- **OpenSSL / OpenSSL3**
  - LibreSSL (clone), BoringSSL (Google)
- **NSS**
  - Network Security Services (Mozilla)
- **NaCl** ("salt")
  - more common as **libsodium**

Examples in **gcrypt**, **OpenSSL / OpenSSL3** and **libsodium**

Crypto libraries

- Random Number Generator (RNG) access
- Hash, keyed-hash (HMAC, msg authentication)
- Symmetric ciphers and modes
- Asymmetric ciphers
- Certificate support, ASN.1, ...
- Key exchange, key derivation
- Helpers
  - secure memory
  - safe comparison
  - network / sockets
  - ...