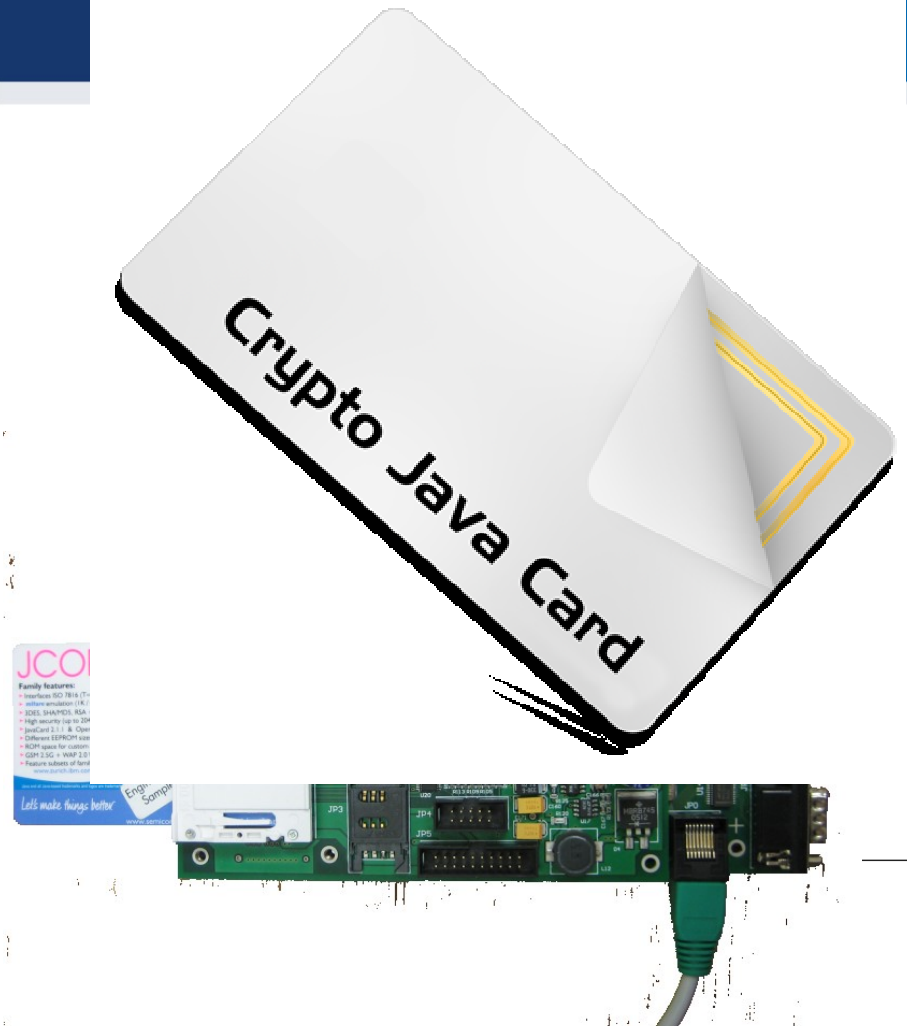
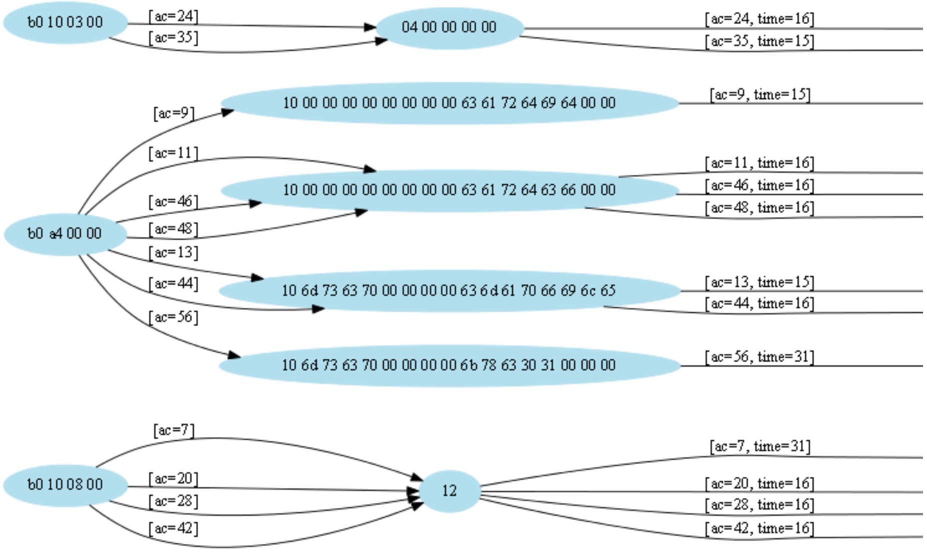
PB173 - Tématický vývoj aplikací v C/C++
Domain specific development in C/C++

Skupina: Aplikovaná kryptografie a bezpečné programování

Petr Švenda svenda@fi.muni.cz
Konzultace: A406, Úterý 13:00-13:50

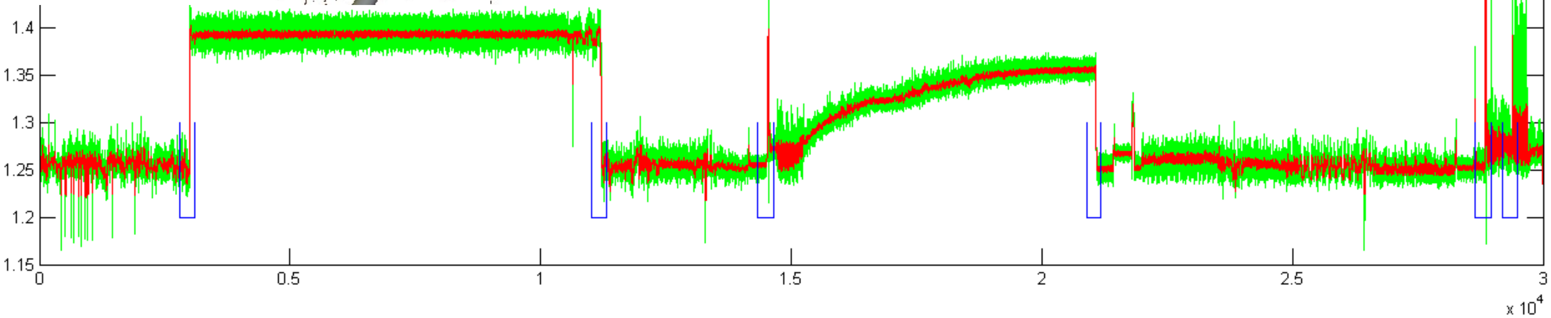
CROCS
Centre for Research on
Cryptography and Security





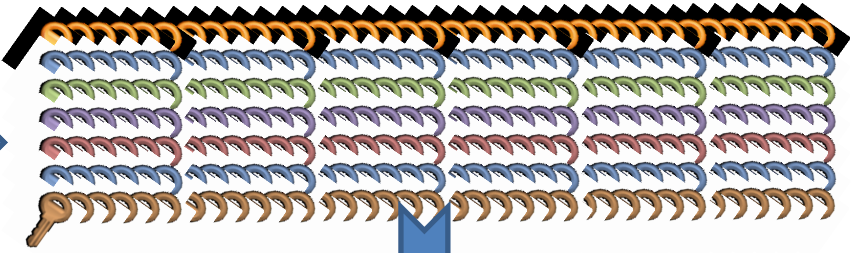
Security programming

Power analysis

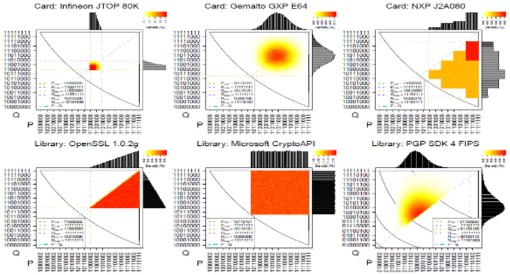


22 sw. libraries
16 smart cards

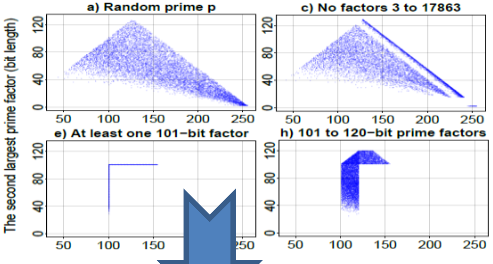
60+ million fresh RSA keypairs



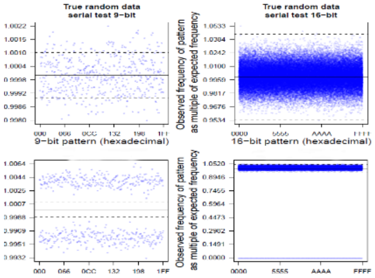
Distribution of primes (MSB)



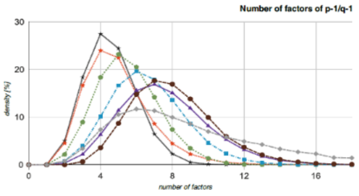
Large factors of p-1 / p+1



Bit stream statistics



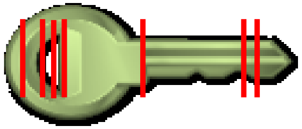
Number of factors



and more…

7 implementation choices observable in public keys

(biased bits of public modulus, “mask”)



Genetic programming

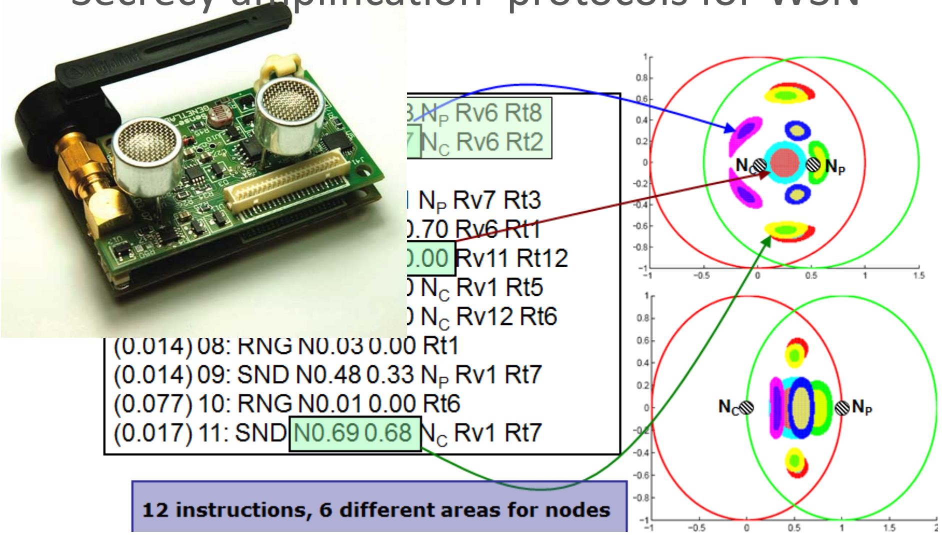
+

⇓

BOINC

Distributed computing

Secrecy amplification protocols for WSN



```
3 NP Rv6 Rt8
7 NC Rv6 Rt2

1 NP Rv7 Rt3
0.70 Rv6 Rt1
0.00 Rv11 Rt12
0 NC Rv1 Rt5
0 NC Rv12 Rt6
(0.014) 08: RNG N0.03 0.00 Rt1
(0.014) 09: SND N0.48 0.33 NP Rv1 Rt7
(0.077) 10: RNG N0.01 0.00 Rt6
(0.017) 11: SND N0.69 0.68 NC Rv1 Rt7
```

12 instructions, 6 different areas for nodes

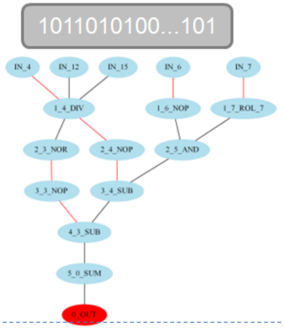
Random distinguisher for crypto fncs

Quantum Random Bit Generator service

ECRYPT

500x 1011010100...101

500x 1001110011...100



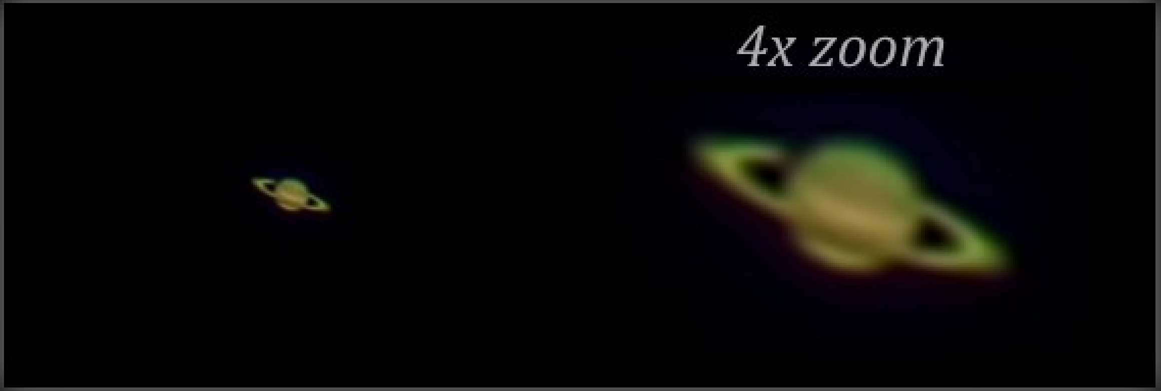
10110111 HW(10110111) > 4 => QRNG

http://astrolight.cz

M45 Pleiades star cluster and reflection nebula

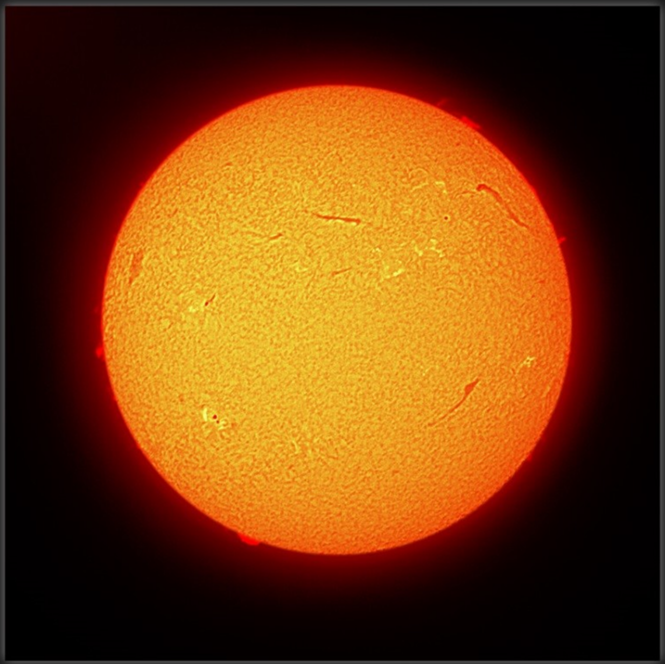
Petr Švenda, http://www.a
Equinox 80EDP 500m
Canon 400D IRmod @

NGC7000 in Cygnus



Saturn 30.6.2012

Petr Švenda, http://astrolight.cz
SW Orion 120/1000mm, stack 900 frames



The Sun 11.12.2011 (H-Alpha)

Petr Švenda, http://astrolight.cz, 11.12.2011
Solarscope Solarview 50mm 0,7Å, Canon 500D, 105 stack

ORGANIZAČNÍ INFORMACE

Co je cílem předmětu

- Získat zkušenosti s implementací většího programu
- Používat vývojové nástroje
- Naučit se dobré programátorské postupy
  - programování obecně
  - ale speciálně v oblasti bezpečnostních aplikací
- Získat praktické postřehy z implementací kryptografických aplikací
  - co nakonec ve firmě vyžadují

Co není cílem předmětu

- Detailní ovládnutí konkrétní technologie
  - zabrousíme do různých oblastí
- Pokročilé zvládnutí celého vývojového procesu
  - to jednoduše nestihneme
- Vysvětlovat základy kryptografie nebo srovnávat všechny možné varianty řešení problému
  - hlavně se budeme snažit prakticky programovat

Organizační

- Formality výuky
  - každotýdenní dvojhodinovka
  - evidovaná účast, 2 neúčasti bez omluvení OK
- Způsob výuky
  - cca 30 min./týdně úvod do problematiky
  - zbytek vaše programování přímo na hodině
  - z mé strany průběžná konzultace nad vznikajícími problémy
  - default Windows (ale můžete pracovat i na jiné platformě)
- Samostatná práce
  - v týmech, průběžná tvorba většího projektu
  - dodělávání práce z hodiny
  - pravidelné bodované předvádění stavu projektu (každé cvičení)

Organizační (2)

- Používané nástroje
  - IDE, verzovací nástroje (git), Doxygen, debugger, analýza a kontrola kódu (CppCheck, Coverity)
  - GitHub + TravisCI + Coverity
  - Ne vše je striktně dané – ptejte se a použijte svoje oblíbené
- Hodnocení
  - účast
  - průběžná práce (10 bodů týdně)
  - prezentace celého projektu (30 bodů)
  - možné bonusy
  - max. 130 bodů, zisk alespoň 85 bodů na kolokvium

Rozdělení do týmů

- 2-3 osoby
- Společná práce, ale každý prezentuje svůj přínos
  - Iniciální prezentace domácího úkolu na dalším cvičení
  - zapracování připomínek, prezentace a hodnocení na dalším cvičení
- Využití sdíleného repozitáře (GitHub) + CI (Travis)
- Rozdělení provedeme až po 14 dnech
  - Po ustálení zapsaných studentů (ve třetím týdnu)

How good YOU are in English?

Apology for all my mistakes, please.

Short questionnaire

- Do you know difference between symmetric and asymmetric cryptography?
- Do you known difference between block and stream cipher?
- Do you know DES and AES algorithm?
- Do you know ECB and CBC encryption mode?
- Do you know principle of hash functions?
- Do you know MD5, SHA-1/2/3 algorithm?
- Do you known concept of digital signature?
- Do you know what perfect forward secrecy means?

Cryptographic libraries

Cryptographic libraries - overview

1. Why not to implement own crypto algorithm/protocol
2. Adequate complexity of library
3. How to get authentic source code
4. Common libraries: OpenSSL, mbed TLS
5. How to use library

Do NOT implement your own algorithms

- Time consuming (probably already done before)
- Functional problems
- Low performance
- Security problems due to bugs
- Security problems due to missing defence against implementation attacks

Do NOT implement your own protocols

- Do not design algorithms/protocols by yourself
- Try to find existing standards
  - NIST, RSA PKCS, RFC, ISO/ANSI
- Try not to deviate from standards
  - compatibility and compliance
  - no need for (time consuming) specification of detailed your scheme
  - small change can have big security impacts

Use well-known implementations

- Use well-known libraries
  - OpenSSL, PolarSSL, GnuPG, BouncyCastle (Java)
- Or implementation of algorithms from well-established authors (for uncommon algs)
  - Brian Gladman, Eric A. Young …

Complexity matters

- Complexity of library implementation should match your needs
  - usually, you need only one or two algorithms
- Multiprocessor or CPU-independent implementation can be overkill
  - and just increase risk of error
- Do you really need library with object-oriented design?
- Large libraries are not always the most suitable ones
  - OpenSSL is complex and interconnected
  - e.g., AES is extractable much easier from mbedTLS (PolarSSL) than from OpenSSL

Code authenticity

- Source code signature
  - Do you really have original binary/source codes?
  - MD5/SHA1 hash (where to get “correct” hash value?)
  - GPG/PGP
- Generate your own GPG/PGP signature keys
  - sign your code releases (on GitHub)

Which one you like more? Why?

ARM mbed TLS

```
/**
 * \brief        Output = HMAC-SHA-512( hmac key, input buffer )
 *
 * \param key      HMAC secret key
 * \param keylen   length of the HMAC key
 * \param input    buffer holding the  data
 * \param ilen     length of the input data
 * \param output   HMAC-SHA-384/512 result
 * \param is384    0 = use SHA512, 1 = use SHA384
 */
void sha512_hmac( const unsigned char *key, size_t keylen,
                  const unsigned char *input, size_t ilen,
                  unsigned char output[64], int is384 );
```

OPENSSL

```
unsigned char *HMAC(const EVP_MD *evp_md, const void *key, int key_len,
                    const unsigned char *d, size_t n, unsigned char *md,
                    unsigned int *md_len);
```

Common libraries – OpenSSL

- Pros:
  - Very rich library
    - lots of algorithms, protocols, paddings
    - not “just” SSL
  - well tested functionally & security over time!
  - significant amount of existing examples on web
- Cons:
  - API is complex and sometimes harder to understand
  - (started as Eric Young’s personal attempt to learn BigInts ☺)
  - relatively low-level functions (can be pros!)
  - code is significantly interconnected
    - not suitable for extraction of single algorithm
  - poor official documentation

Common libraries – mbed TLS

- (Formerly PolarSSL)
- Pros:
  - API is simple and clear
  - Easy to extract single algorithm
  - Now widely used, reasonably tested
- Cons:
  - fewer supported algorithms and standards
  - dual licensing, but not BSD-like license

How to use library

1. Extract code and compile alone
   - some work with extraction
   - small, clean and self-containing result
2. Compile against whole library
   - usually easy to do
   - but dependence on possibly unused code
3. Link statically against dynamic library
   - dll/so must be always present to run program

How to use library (2)

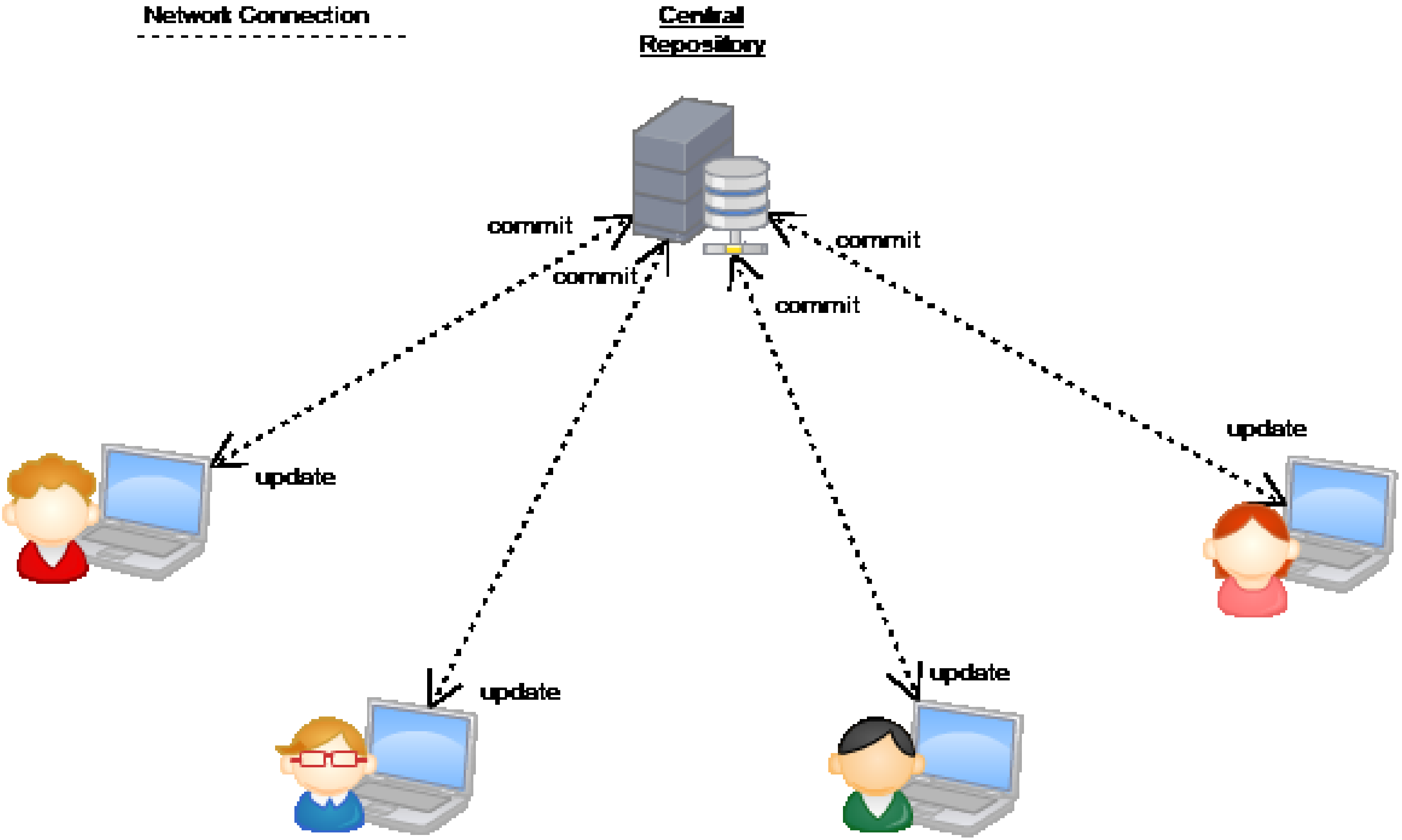
4. Link dynamically against dynamic library
   - try to open dll file and obtain function handle
5. Link against service provider functions
   - Cryptography Service Providers in particular
   - API for listing of available service providers (CryptEnumProviders)
   - standardized functions provided by providers http://msdn.microsoft.com/en-us/library/aa380252%28v=VS.85%29.aspx#service_provider_functions

Security implications of (dynamic) libraries

- Library can be forged and exchanged
- Library-in-the-middle attack easy
  - data flow logging
  - input/output manipulation
- Library outputs can be less checked than user inputs
  - feeling that library is my “internal” stuff and should play by „my“ rules
- Library function call can be behind logical access controls
- Library can contain bugs
  - Serious development also needs 3rd party libraries control process

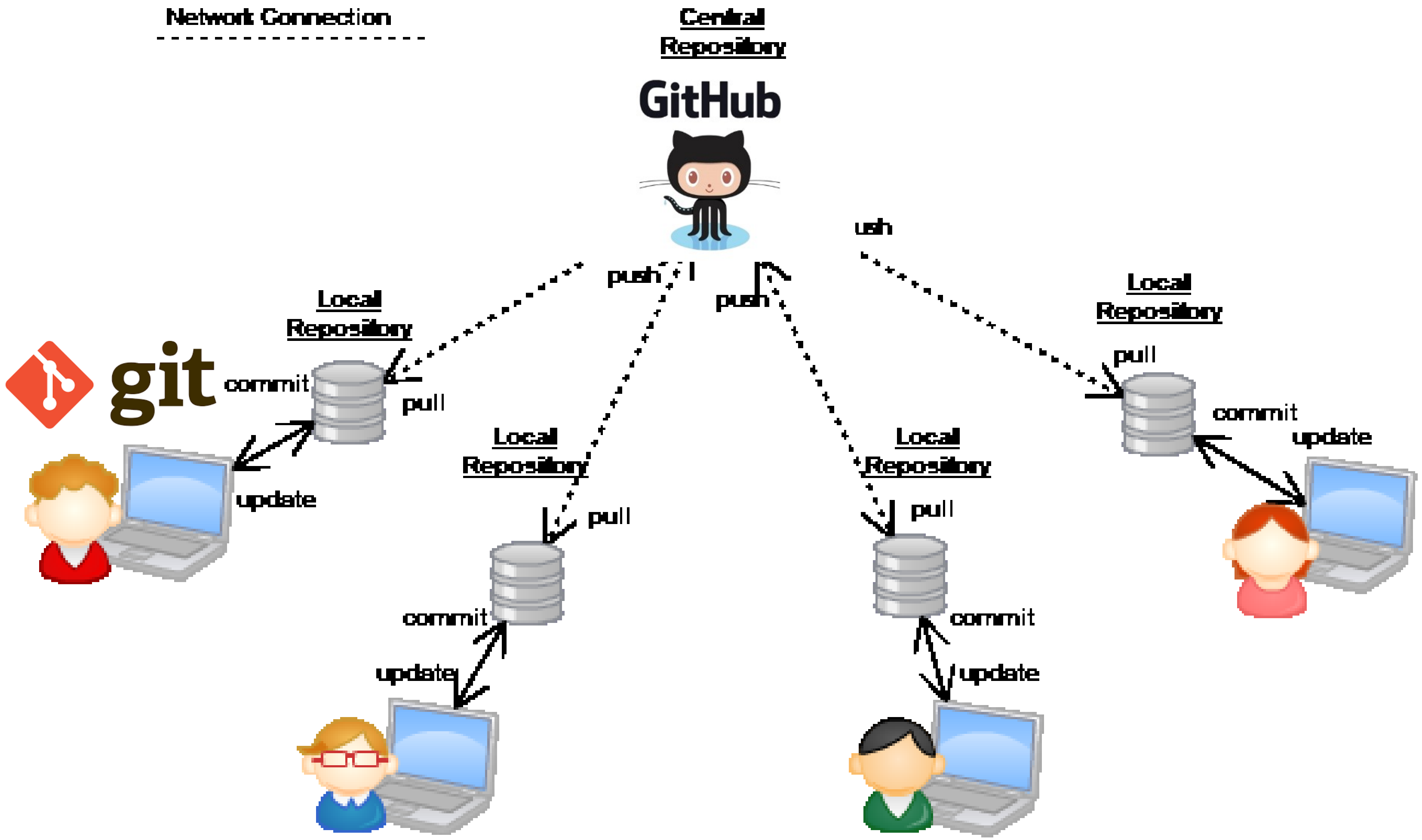
Practical assignment

SVN style – central repository



tps://programmers.stackexchange.com/questions/35074/im-a-subversion-geek-why-should-i-consider-or-not-consider-mercurial-or-git-or

GIT-style – distributed repository



ps://programmers.stackexchange.com/questions/35074/im-a-subversion-geek-why-should-i-consider-or-not-consider-mercurial-or-git-or

Setup your GitHub repository

1. Setup your GitHub account and repository
   - E.g., PB173 test
   - .gitignore C++
   - License MIT
2. Create first milestone (Issues→Set milestone→Create…)
3. Create first issue (Labels, Milestone, Assignee)
   - “Setup initial repo files”
4. Install git locally (GitHub client, TortoiseGit…)
5. Git Clone (your repository)
   - Into local directory

Use your GitHub repository

- Create small project (your favourite IDE)
  - Commit, Push
- Try to modify some files locally
  - Commit, Push
- Try to modify some files in repo via web interface
  - Simulated parallel modification by other developer
  - Git Pull / Sync
- Close your first issue ☺

Practical assignment

- Download *mbed TLS* (formerly PolarSSL) library
  - and check signature (gpg --verify)
- Write small project (mbed TLS based)
  - read, encrypt and hash supplied file, write into out file
  - read, verify hash and decrypt file
  - use AES-128 in CBC mode and SHA2-512
  - use PKCS#7 padding method for encryption (RFC 3852)
- Start with New Project+mbedTLS+AES

Questions

Submissions, deadlines

- Commit into your GitHub repository (frequently ☺)
- Upload application source codes as single zip file into IS
  - Use GitHub’s download ZIP feature
  - Homework vault (Crypto - 1. homework (AES+SHA2))
- DEADLINE: 27.2. 12:00 (first part)
  - application capable to read, encrypt, decrypt, hash
  - Text file containing description how you did PGP signature verification (whole process including import of public keys etc.)
  - 0-5 points assigned
- DEADLINE 5.3. 12:00 (second part)
  - addition of unit tests
  - 0-5 points assigned