PA193 - Secure coding principles and practices

Protecting integrity of modules and external components

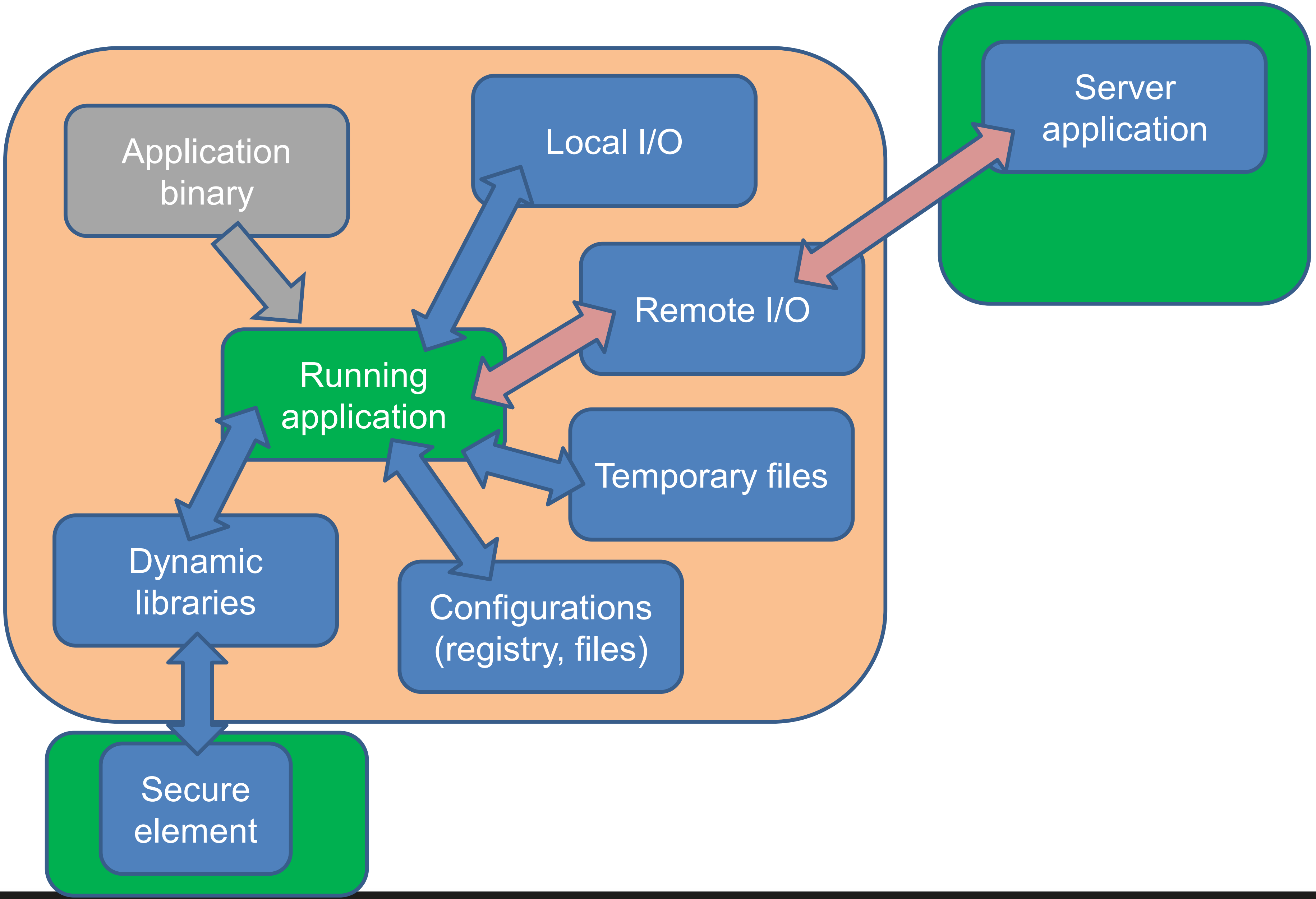
Petr Švenda svenda@fi.muni.cz

Centre for Research on
Cryptography and Security

Overview

- Lecture:
  - dynamic libraries, forging, protection
  - code signing
  - temporary files
  - protections in whitebox attacker model
- Labs
  - security for temporary files

PROBLEM


Application binary
Local I/O
Server application
Remote I/O
Running application
Temporary files
Dynamic libraries
Configurations (registry, files)
Secure element

DYNAMIC LIBRARIES

Dynamic library usage (Windows)

- Static linking
  - *library*.lib added to dependencies
- Run-time dynamic linking
  - controllable run-time search for dynamic library
  - developer can control and respond on (un)available lib
  - LoadLibrary(*path*) & FreeLibrary(*hLib*)
- Run-time search for specific function
  - GetProcAddress(*hLib, "function_name"*)
  - cast to target function prototype (later)

Default order of directory search for DLL

1. The directory from which the application loaded
   - “application directory”
2. The system directory
3. The 16-bit system directory
4. The Windows directory
5. The current directory
6. The directories that are listed in the PATH environment variable

- Safe DLL search mode place current directory to 5.

DLL preloading attack

- Called DLL preloading or binary planting attack

1. Attacker obtains write access to one directory in search list
2. Attacker places malicious DLL here
3. If application will not find DLL in directories searched before, attacker's DLL gets loaded
4. Malicious code is executed with application privileges

How to execute man-in-the-middle for dll

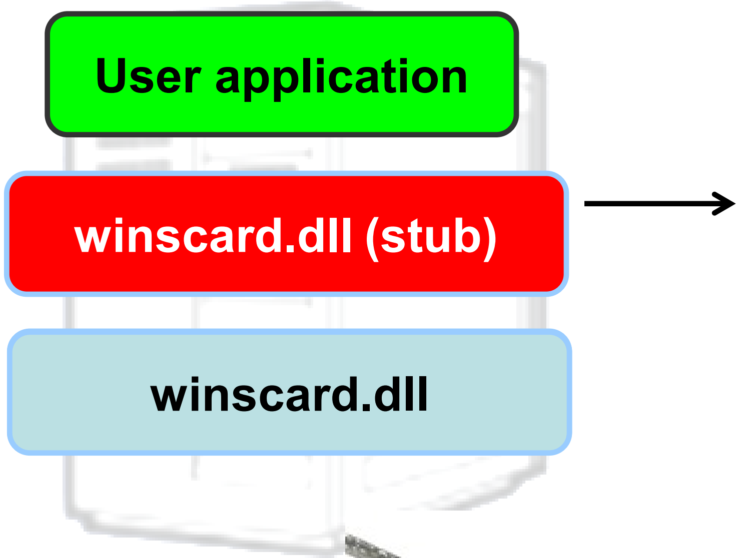
- Application wants to load dynamic library
  - according to specified name, e.g., winscard.dll
  - e.g. via LoadLibrary("winscard.dll") call
- Create dynamic library ("stub") with the same name and the same set of exported functions
- Move stub DLL into directory where application looks first for requested DLL
  - stub is loaded instead of original
  - application will call stub function instead
- When given function from stub is called, pass input arguments to the original DLL and return response
  - or modify, log, delay, block...

Example: APDUPlay

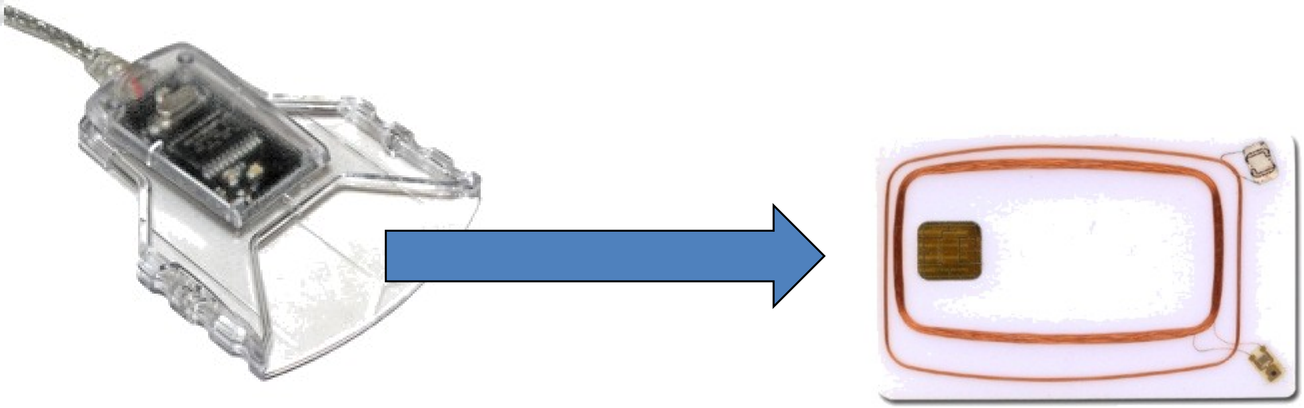
- Dynamic library for interception and manipulation of communication with smart cards
  - winscard.dll, APDU-based communication
  - http://www.fi.muni.cz/~xsvenda/apduinspect.html
- What you can achieve:
  - log input/output APDU commands (including keys, PINs...)
  - manipulate APDUs content according to predefined rules
    - e.g., return OK even when verification fails
    - e.g., simulate presence of reader / smart card
  - reverse engineer protocol used based on communication
  - redirect communication to other computer via socket

Let’s write own winscard.dll (PC/SC)

based on ApduView utility (by Fernandes)



```
[begin]
SCardTransmit (handle 0xEA010001)# apduCounter:0#
totalBytesINCounter:1#
transmitted:00 a4 04 00 0a a0 00 00 00 28 80 10 30 01 ff
responseTime:31#
SCardTransmit result:0x0#
received:6a 81

SCardTransmit (handle 0xEA010001)# apduCounter:1#
totalBytesINCounter:16#
…
```



How to load proper library?

1. Use fully qualified path to load library (LoadLibrary)
2. Dynamic-Link Library Redirection
   - https://tinyurl.com/chy5wum
   - redirection file is created in application directory
   - *App_name*.local (e.g., explorer.exe → explorer.exe.local)
     - (content of file is ignored)
   - application directory is searched first for the target DLL
   - good practice to install application DLLs in its directory
     - will not overwrite other versions of same DLL
   - (will not work if application has manifest)

How to load proper library? (2)

3. Application manifest
   - XML file with various application configurations
   - including versions and hash (SHA-1) of required DLLs
   - when required DLL is loaded, hash is checked
   - https://tinyurl.com/b2dz8u9

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
…
   <file name="bar.dll" hash="ac72753e5bb20446d88a48c8f0aaae769a962338" hashalg="SHA1"/>
   <file name="foo.dll" hash="a7312a1f6cfb46433001e0540458de60adcd5ec5" hashalg="SHA1">
…
```

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 then user inputs
  - feeling that library is my “internal” stuff and should play by „my“ rules
- Library function call can be behind logical access controls

References

- Dynamic-Link Library Security
  - http://msdn.microsoft.com/en-us/library/windows/desktop/ff919712%28v=vs.85%29.aspx
- Assembly manifests
  - http://msdn.microsoft.com/en-us/library/aa374219%28v=vs.85%29.aspx
- Assembly signing example
  - http://msdn.microsoft.com/en-us/library/aa374228%28v=vs.85%29.aspx

CODE SIGNING

Code authenticity

- Why to authenticate binary/source codes?
  - random transmission errors solved by transport layer (CRC)
  - intentional modification on remote code repository
  - intentional modification during transport (MITM)
  - intentional modification locally (malware in user space)
  - NSA Bullrun program...
- Strong authentication often required implicitly
  - relatively restricted platforms like iOS / Android...
  - kernel drivers (no unsigned kernel driver from Vista 64bit)
  - official software repositories…

Possibilities for code signature

1. Non-keyed hash function sign = H(your_package)
   - everyone can compute H(modified_package)
   - where to get “correct” hash value? (usually same webpage ☹)
   - often MD5 algorithm (known collisions, insecure)
   - often need for manual verification (lazy users)
2. Authentication based on symmetric cryptography
   - keyed MAC, sign = HMAC(key, your_package)
   - not suitable for *one to many* distribution (shared key)
3. Authentication based on asymmetric cryptography
   - digital signatures of package sign = RSA(private_key, your_package)
   - everybody can Verify(public_key, sign)
   - most suitable, but may require PKI (trust to public key is critical)

Code signing (GPG/PGP)

- PGP/GPG can be used for code signing
  - same process as message signature
  - signature is usually detached into separate file (*.sig)

```
gpg --output app.sig --detach-sig app
gpg --verify app.sig app
```

- Trust to signing key is needed
  - public key should be obtained from trusted source
  - but usually only publisher website or keyserver ☹
- Can be used to sign packages (e.g., Debian RPM)
  - http://fedoranews.org/tchung/gpg/

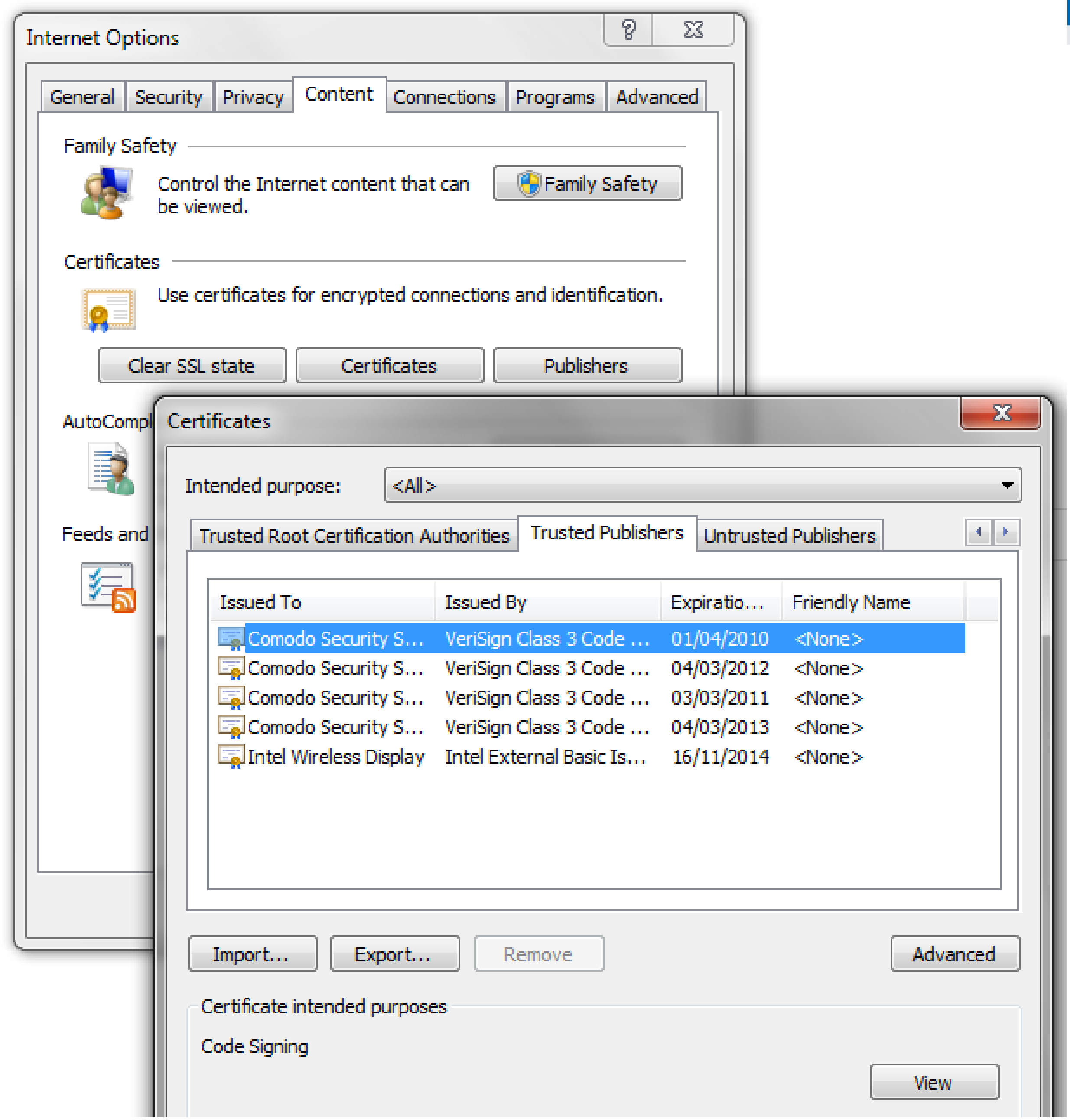
Various code signing managers

- Java certificates (also Android)
  - java-based applications and applets (.jar)
- Microsoft Authenticode
  - Active-X controls, plug-ins, execs (.cab, .cat, .ctl, .ocx, .exe, .dll)
- Adobe Air certificate
  - Adobe Ajax and flex files (.air and .airi)
- Microsoft Office and VBA certificate
  - Microsoft Office macros and Visual Basic applications
- Apple developer program
  - applications for iOS platform
- Difference: local sign vs. additional check on server

Code signing (Microsoft's Authenticode)

1. Publisher obtains Code Signing Digital ID
   - X.509 certificate with public key signed by trusted authority
   - authority's certificate imported in Trusted Publishers
2. Publisher creates code (application)
3. Publisher signs code with its private key
4. Application is distributed along with signature(s)
5. Application signature is verified, user is notified
   - invalid signature of application → confirmation from user
   - invalid signature of driver → no installation
   - (problem with legacy apps, non-compliant developers)

- (RSA 2048bit with SHA-1)



Control the Internet content that can be viewed.

Use certificates for encrypted connections and identification.

Intended purpose: <All>

Issued To	Issued By	Expiratio...	Friendly Name
Comodo Security S...	VeriSign Class 3 Code ...	01/04/2010	<None>
Comodo Security S...	VeriSign Class 3 Code ...	04/03/2012	<None>
Comodo Security S...	VeriSign Class 3 Code ...	03/03/2011	<None>
Comodo Security S...	VeriSign Class 3 Code ...	04/03/2013	<None>
Intel Wireless Display	Intel External Basic Is...	16/11/2014	<None>

Certificate intended purposes

Code Signing

Microsoft WHQL

- Windows Hardware Quality Labs (WHQL)
- Intended for kernel-mode binaries (drivers, dll)
- WHQL-certified binaries can be distributed through the Windows Update program
- Signature stored in catalog file (*.cat)
- Practical Windows Code and Driver Signing
  - http://www.davidegrayson.com/signing/

Microsoft Authenticode – selfsign (testing)

- Process of creating Authenticode selfsign certificate
  - used for testing purposes
  - your certificate imported as Trusted Publisher
  - signing of exe, dll, scripts
- http://msdn.microsoft.com/en-us/library/office/aa140234%28v=office.10%29.aspx

- Why it will not work for other computers?

Signed code == secure code?

- Developer can sign anything
  - additional layer of validation of application needed
    - Microsoft WHQL, Google Play, Apple App Store...
  - but his/her key (and apps) can be revocated
- Trusted authority can be compromised
  - Comodo, DigiNotar...
- Signature must be verified correctly
  - Android Master key vulnerability
  - https://tinyurl.com/kj63ae8, https://tinyurl.com/p5fu3j3
- Signed application can execute unsigned code
  - Apple's Nitro JavaScript engine, https://tinyurl.com/6tpvzpq

TEMPORARY FILES

Why we use temporary files?

- Temporary files are used only during the program run
  - no persistence between runs is typically assumed
- Used to offload (large) data from memory to disk
  - too large to fit into memory of the application
- Communication with other process
  - transferring data through the file system

#include <stdio.h>

Creating temporary files in C/C++

- `FILE* tmpfile (void);`
  - creates new temporary binary file with unique file name and opens it for update ("wb")
  - file is created in TMP directory according to environment settings
  - file is automatically closed at program end (including crash)
- `char* tmpnam (char * str);`
  - return unique file name not used yet (but is not opening file)
  - additional call to fopen() is required
  - if not specified, file is created in current directory
  - Warning: file is not opened in the same time, attacker can open it and manipulate in between – *Race condition*
  - tmpnam generates a different string each time you call it, up to TMP_MAX times (defined in stdio.h as 65,535)

Creating temporary files in C/C++ (2)

- Function alternatives from Secure C library exist
  - secure from the perspective of buffer manipulation
  - not necessarily against various attacks described later
- `errno_t tmpnam_s(char *s, rsize_t maxsize);`
  - returns unique file name (same format as tmpnam)
- `errno_t tmpfile_s(FILE*restrict*restrict streamptr)`
  - creates new temporary binary file with unique file name and opens it for update (“wb”)
  - NOTE: if program crashes, tmp file might NOT be removed (difference to `tmpfile`)

Removing temporary files in C/C++

- _rmtmp()
  - removes all temporary files created by `tmpfile / tmpfile_s`
  - NOTE: `will leave invalid FILE* handle(s)`
- Files created by `tmpfile / tmpfile_s`
  - fclose() will remove the file
  - normal program termination will remove the file
  - abnormal program termination might not remove files
- Temporary files opened by tmpnam() & fopen()
  - not treated by system as temporary files
  - developer is responsible for removal

Problem with temporary files - TOCTOU

```
#include <stdio.h>
int main() {
  const size_t BUFFER_SIZE = 1000;
  char filename[BUFFER_SIZE];
  // Get unique file name
  tmpnam_s(filename, BUFFER_SIZE);
  // Test if no such file exists
  FILE *fp = fopen(filename,"r");
  if( !fp ) { // file does not exist
    fp = fopen(filename,"w");
    // use tmp file...
    fclose(fp);
  } else {
    // file exists, go for other name
    fclose(fp);
  }
  return 0;
}
```

Time Of Check

attacker can open filename during this period (TOCTOU)

Time Of Use

Problem with temp. files - predictability

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char* argv[]) {
  const size_t BUFFER_SIZE = 1000;
  const size_t NUM_FILES = 15;
  char buffer[BUFFER_SIZE];
  // Obtain directory for temporary files
  GetTempPath(BUFFER_SIZE, buffer);
  printf("Temporary directory: %s\n", buffer);

  FILE * pFile1[NUM_FILES];
  // Obtain unique file name
  for (size_t i = 0; i < NUM_FILES; i++) {
     tmpnam_s(buffer, BUFFER_SIZE);
     printf("Unique file name: %s\n", buffer);
     fopen_s(&pFile1[i], buffer + 1, "wb");
  }

  return 0;
}
```

```
Temporary directory:
C:\Users\petr\AppData\Local\Temp\
Unique file name: \s4sg.
Unique file name: \s4sg.1
Unique file name: \s4sg.2
Unique file name: \s4sg.3
Unique file name: \s4sg.4
Unique file name: \s4sg.5
Unique file name: \s4sg.6
Unique file name: \s4sg.7
Unique file name: \s4sg.8
Unique file name: \s4sg.9
Unique file name: \s4sg.a
Unique file name: \s4sg.b
Unique file name: \s4sg.c
Unique file name: \s4sg.d
Unique file name: \s4sg.e
```

Problem with temp. files – predictability (2)

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  const size_t NUM_FILES = 15;

  FILE * pFile2[NUM_FILES];
  // Open temporary files
  for (size_t i = 0; i < NUM_FILES; i++) {
    tmpfile_s(&pFile2[i]);
  }
  // Wait - tmp files can be spotted in tmp directory
  getchar();
  // Remove tmp files (only these opened by tmpfile / tmpf
  // Handles FILE* inside pFile2 now have invalid value
  _rmtmp();

  return 0;
}
```

```
06/11/2013  15:28  0 t3oc
06/11/2013  15:28  0 t3oc.1
06/11/2013  15:28  0 t3oc.2
06/11/2013  15:28  0 t3oc.3
06/11/2013  15:28  0 t3oc.4
06/11/2013  15:28  0 t3oc.5
06/11/2013  15:28  0 t3oc.6
06/11/2013  15:28  0 t3oc.7
06/11/2013  15:28  0 t3oc.8
06/11/2013  15:28  0 t3oc.9
06/11/2013  15:28  0 t3oc.a
06/11/2013  15:28  0 t3oc.b
06/11/2013  15:28  0 t3oc.c
06/11/2013  15:28  0 t3oc.d
06/11/2013  15:28  0 t3oc.e
```

Problems with creating tmp files (MSVC)

- tmpnam() / tmpnam_s()
  - format as sxxx.#
  - TOCTOU
- tmpfile() / tmpfile_s()
  - unique file name is generated as txxx.# where xxx is digit or character and # is sequential number or character
  - predictability
- Attacker can:
  - predict file name, create own file (TOCTOU)
  - then capture sensitive & forge malformed data

Temporary files in Java

- File tempFile = File.createTempFile(prefix, suffix);
  - Will keep file even when JVM exits
  - Longer name then in C/C++ (by default)
- Ask for delete on JVM exit
  - tempFile.deleteOnExit();
  - But deleted only during "normal" termination
  - *"Deletion will be attempted only for normal termination of the virtual machine, as defined by the Java Language Specification."*
- Similar problems as for C/C++

TEMPORARY FILES – SECURITY CHECKLIST

Temporary files security checklist

1. Avoid temporary files if possible ☺
2. Don’t use standard C function for temporary files
   - mktemp(), tmpnam(), tempname(), tmpfile()...
   - predictable names, race conditions
3. Don’t store sensitive information in temp files
   - temp files are common attack vector, prevent it
4. Research where are temporary files stored
   - no standard function for that in C/C++
   - Windows: GetTempPath()

Temporary files security checklist (2)

5. Ensure strong uniqueness and unpredictability for name of temporary file
   - don’t use tmpnam or tmpfile functions (predictable)
   - generate long random name internally, open it, check
   - use strong random generating function like CAPI’s CryptGenRandom(), OpenSSL’s RAND_bytes()...

Temporary files security checklist (2)

6. Ensure proper permissions for temporary file
   - avoid publically writable directories if possible
   - if publically writable directory is used, create subdirectory and set ACL's (read and write) only for your application
7. Encrypt log file content with random key
   - generate random secret key every time you run your application
   - encrypt data before writing into log file (and decrypt when reading)
   - when program is abnormally terminated, (encrypted) temporary file will stay but random key will is lost
   - attacker cannot supply older temporary version (different key)

Temporary files security checklist (3)

8. Perform secure cleanup
   - overwrite content of temporary file with random data before close
     - even when performing log file encryption (key may leak in memory dump, pagefile etc.)
   - leave no temporary files behind
     - close temporary files as soon as possible
     - call _rmtmp**()** if standard C functions were use for open
   - still possible to leave temporary files during abnormal termination
     - utilize own signal handlers
     - wrap main into big exception handler and cleanup

Temporary files security checklist (4)

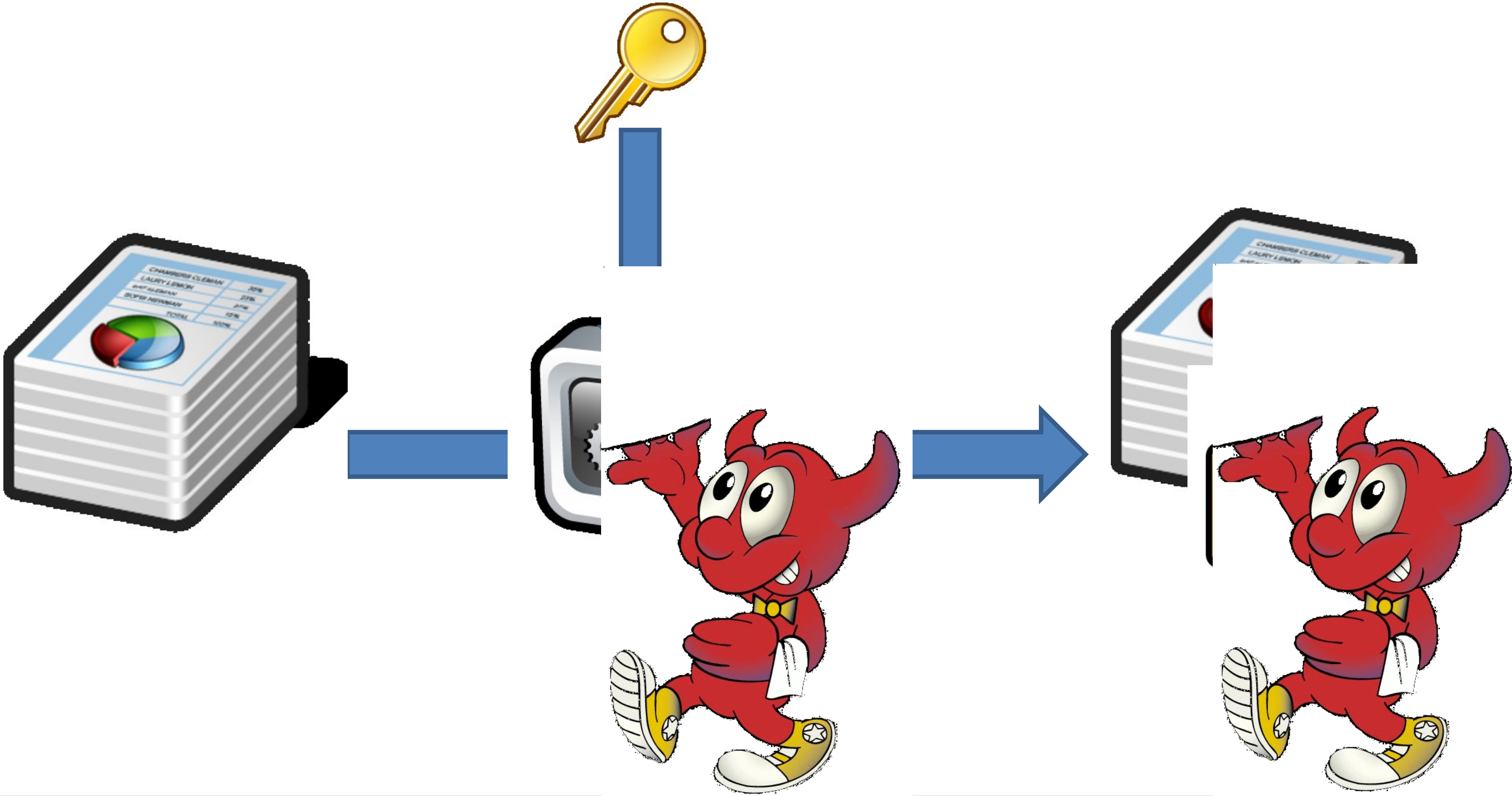
9. Rely on absolute, not on relative paths
   - relative paths will change when application current directory change
   - if user provides directory path for temporary files, sanitize it
   - use file handles (e.g., FILE*) instead of file path (TOCTOU)
10. Open files exclusively and non-existing only
    - C99: fopen(“filename”, “wb”) opens new as well as existing file ☹
    - C11: new exclusive create-and-open mode ("…x") for fopen
    - POSIX: open() with O_CREAT|O_EXCL
    - WIN32 API:CreateFile() with CREATE_NEW

References

- Security Tips for Temporary File Usage in Applications
  - http://www.codeproject.com/Articles/15956/Security-Tips-for-Temporary-File-Usage-in-Applicat
- FIO43-C. Do not create temporary files in shared directories
  - https://www.securecoding.cert.org/confluence/display/seccode/FIO43-C.+Do+not+create+temporary+files+in+shared+directories
- MITRE CWE-377: Insecure temporary files
  - http://cwe.mitre.org/data/definitions/377.html

OBFUSCATION, PROTECTING SOFTWARE MODULES

Standard vs. whitebox attacker model



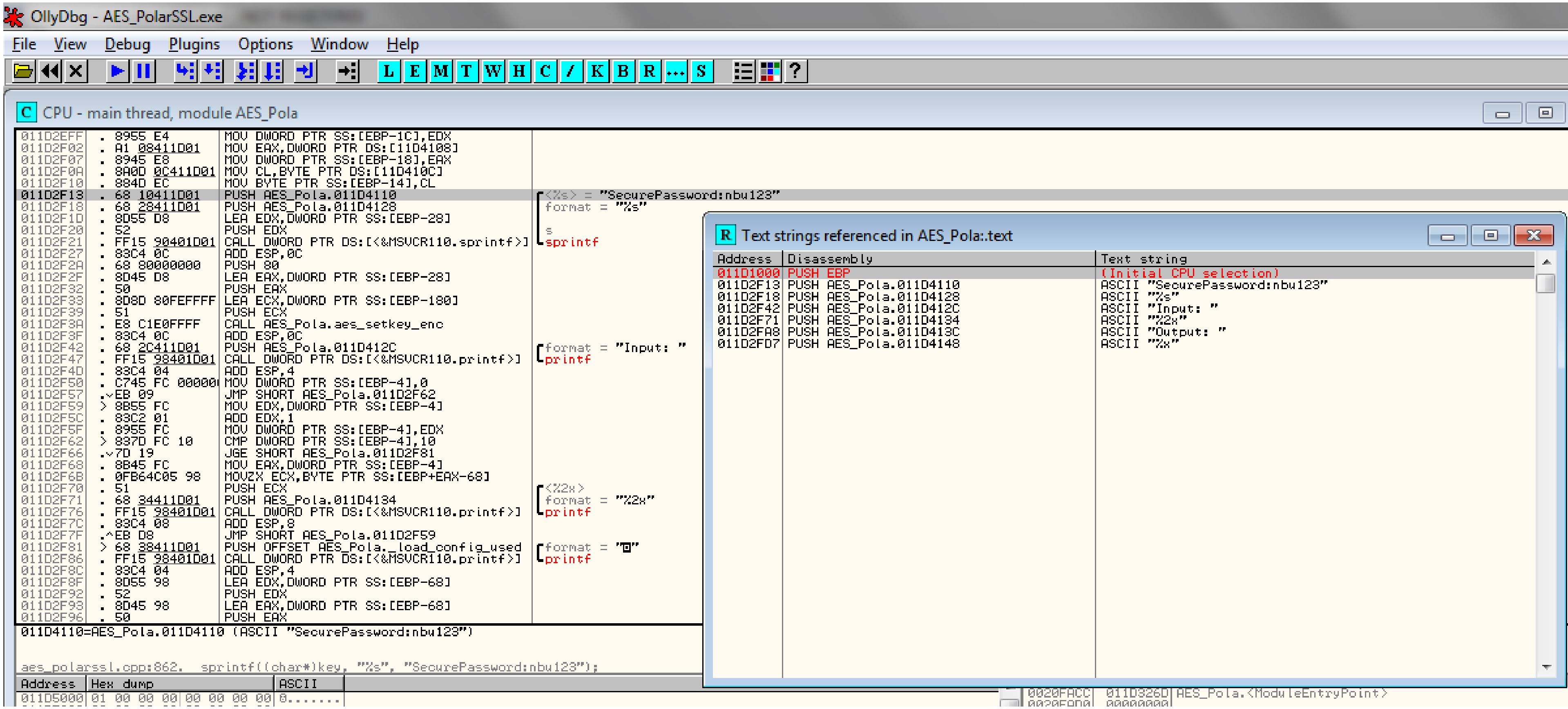
Standard AES API (PolarSSL)

```
/**
 * \brief          AES key schedule (encryption)
 *
 * \param ctx      AES context to be initialized
 * \param key      encryption key
 * \param keysize  must be 128, 192 or 256
 *
 * \return         0 if successful, or POLARSSL_ERR_AES_INVALID_KEY_LENGTH
 */
int aes_setkey_enc(aes_context *ctx, const unsigned char *key, unsigned int keysize);

/**
 * \brief          AES-ECB block encryption/decryption
 *
 * \param ctx      AES context
 * \param mode     AES_ENCRYPT or AES_DECRYPT
 * \param input    16-byte input block
 * \param output   16-byte output block
 *
 * \return         0 if successful
 */
int aes_crypt_ecb( aes_context *ctx,
                   int mode,
                   const unsigned char input[16],
                   unsigned char output[16] );
```

Standard AES - usage

```
void simpleAES() {
    unsigned char key[32];
    unsigned char buf[16];
    aes_context ctx;

    memset( buf, 1, sizeof(buf));
    memset( &ctx, 0, sizeof(ctx));

    // Set the key
    sprintf((char*)key, "%s", "SecurePassword:nbu123");
    aes_setkey_enc( &ctx, key, 128);

    printf("Input:  ");
    for (int i = 0; i < AES_BLOCK_SIZE; i++) printf("%2x", buf[i]);
    printf("\n");

    // Encrypt one block
    aes_crypt_ecb( &ctx, AES_ENCRYPT, buf, buf );
    printf("Output:  ");
    for (int i = 0; i < AES_BLOCK_SIZE; i++) printf("%x", buf[i]);
}
```

OllyDbg – key value is static string



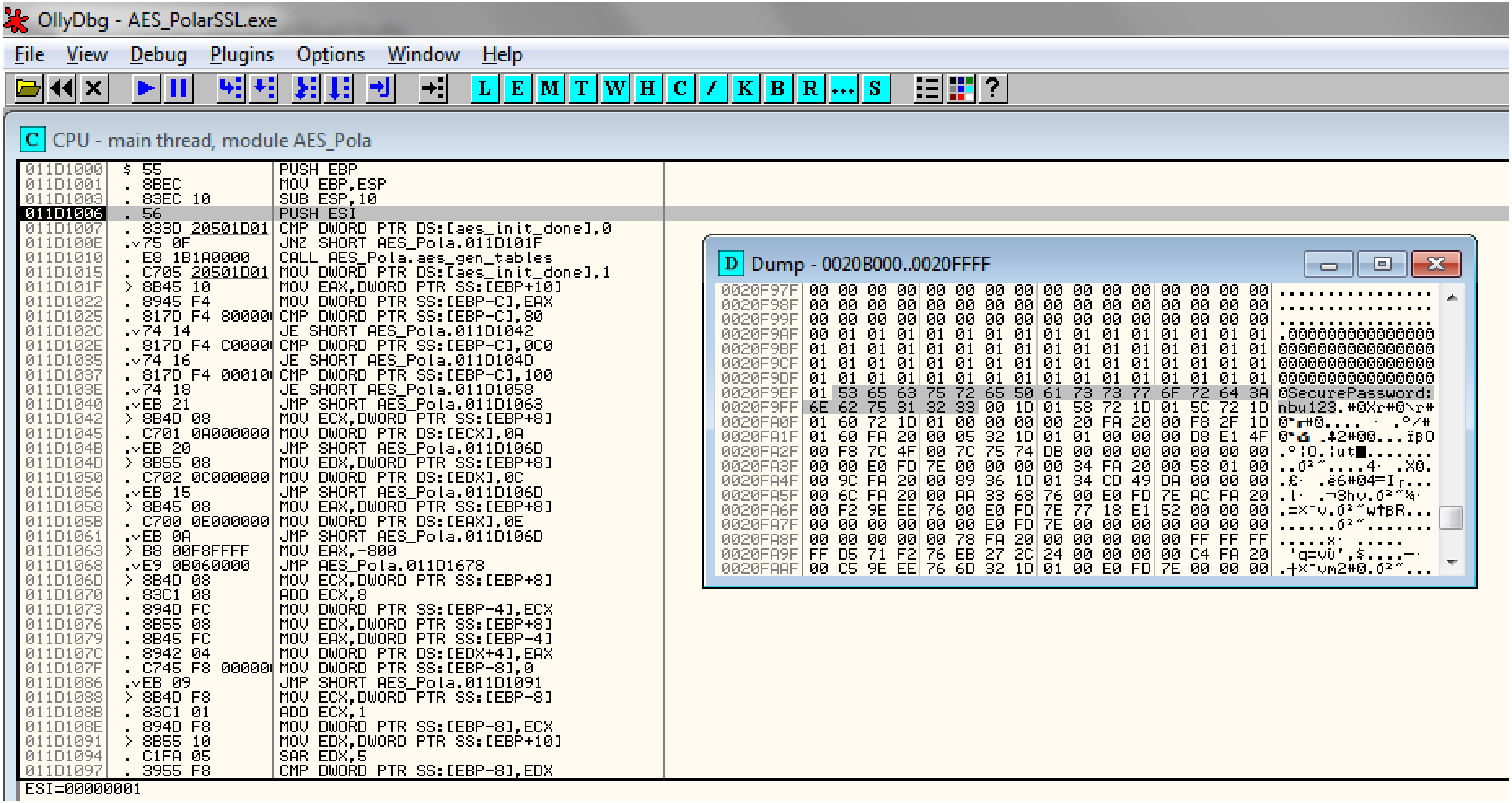
```
011D2EFF  . 8955 E4        MOV DWORD PTR SS:[EBP-1C],EDX
011D2F02  . A1 08411D01    MOV EAX,DWORD PTR DS:[11D4108]
011D2F07  . 8945 E8        MOV DWORD PTR SS:[EBP-18],EAX
011D2F0A  . 8A0D 0C411D01  MOV CL,BYTE PTR DS:[11D410C]
011D2F10  . 884D EC        MOV BYTE PTR SS:[EBP-14],CL
011D2F13  . 68 10411D01    PUSH AES_Pola.011D4110                        [<%s> = "SecurePassword:nbu123"
011D2F18  . 68 28411D01    PUSH AES_Pola.011D4128                        |format = "%s"
011D2F1D  . 8D55 D8        LEA EDX,DWORD PTR SS:[EBP-28]                 |
011D2F20  . 52             PUSH EDX                                      |s
011D2F21  . FF15 90401D01  CALL DWORD PTR DS:[<&MSVCR110.sprintf>]       Lsprintf
011D2F27  . 83C4 0C        ADD ESP,0C
011D2F2A  . 68 80000000    PUSH 80
011D2F2F  . 8D45 D8        LEA EAX,DWORD PTR SS:[EBP-28]
011D2F32  . 50             PUSH EAX
011D2F33  . 8D8D 80FEFFFF  LEA ECX,DWORD PTR SS:[EBP-180]
011D2F39  . 51             PUSH ECX
011D2F3A  . E8 C1E0FFFF    CALL AES_Pola.aes_setkey_enc
011D2F3F  . 83C4 0C        ADD ESP,0C
011D2F42  . 68 2C411D01    PUSH AES_Pola.011D412C                        [format = "Input: "
011D2F47  . FF15 98401D01  CALL DWORD PTR DS:[<&MSVCR110.printf>]        Lprintf
011D2F4D  . 83C4 04        ADD ESP,4
011D2F50  . C745 FC 000000 MOV DWORD PTR SS:[EBP-4],0
011D2F57  .vEB 09          JMP SHORT AES_Pola.011D2F62
011D2F59  > 8B55 FC        MOV EDX,DWORD PTR SS:[EBP-4]
011D2F5C  . 83C2 01        ADD EDX,1
011D2F5F  . 8955 FC        MOV DWORD PTR SS:[EBP-4],EDX
011D2F62  > 837D FC 10     CMP DWORD PTR SS:[EBP-4],10
011D2F66  .v7D 19          JGE SHORT AES_Pola.011D2F81
011D2F68  . 8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
011D2F6B  . 0FB64C05 98    MOVZX ECX,BYTE PTR SS:[EBP+EAX-68]
011D2F70  . 51             PUSH ECX                                      [<%2x>
011D2F71  . 68 34411D01    PUSH AES_Pola.011D4134                        |format = "%2x"
011D2F76  . FF15 98401D01  CALL DWORD PTR DS:[<&MSVCR110.printf>]        Lprintf
011D2F7C  . 83C4 08        ADD ESP,8
011D2F7F  .^EB D8          JMP SHORT AES_Pola.011D2F59
011D2F81  > 68 38411D01    PUSH OFFSET AES_Pola._load_config_used        [format = "□"
011D2F86  . FF15 98401D01  CALL DWORD PTR DS:[<&MSVCR110.printf>]        Lprintf
011D2F8C  . 83C4 04        ADD ESP,4
011D2F8F  . 8D55 98        LEA EDX,DWORD PTR SS:[EBP-68]
011D2F92  . 52             PUSH EDX
011D2F93  . 8D45 98        LEA EAX,DWORD PTR SS:[EBP-68]
011D2F96  . 50             PUSH EAX
011D4110=AES_Pola.011D4110 (ASCII "SecurePassword:nbu123")

aes_polarssl.cpp:862.  sprintf((char*)key, "%s", "SecurePassword:nbu123");
Address  Hex dump                 ASCII
011D5000 01 00 00 00 00 00 00 00  0.......
```

Text strings referenced in AES_Pola:.text

```
Address  Disassembly                Text string
011D1000 PUSH EBP                   (Initial CPU selection)
011D2F13 PUSH AES_Pola.011D4110     ASCII "SecurePassword:nbu123"
011D2F18 PUSH AES_Pola.011D4128     ASCII "%s"
011D2F42 PUSH AES_Pola.011D412C     ASCII "Input: "
011D2F71 PUSH AES_Pola.011D4134     ASCII "%2x"
011D2FA8 PUSH AES_Pola.011D413C     ASCII "Output: "
011D2FD7 PUSH AES_Pola.011D4148     ASCII "%x"
```

```
0020FACC  011D326D  AES_Pola.<ModuleEntryPoint>
```

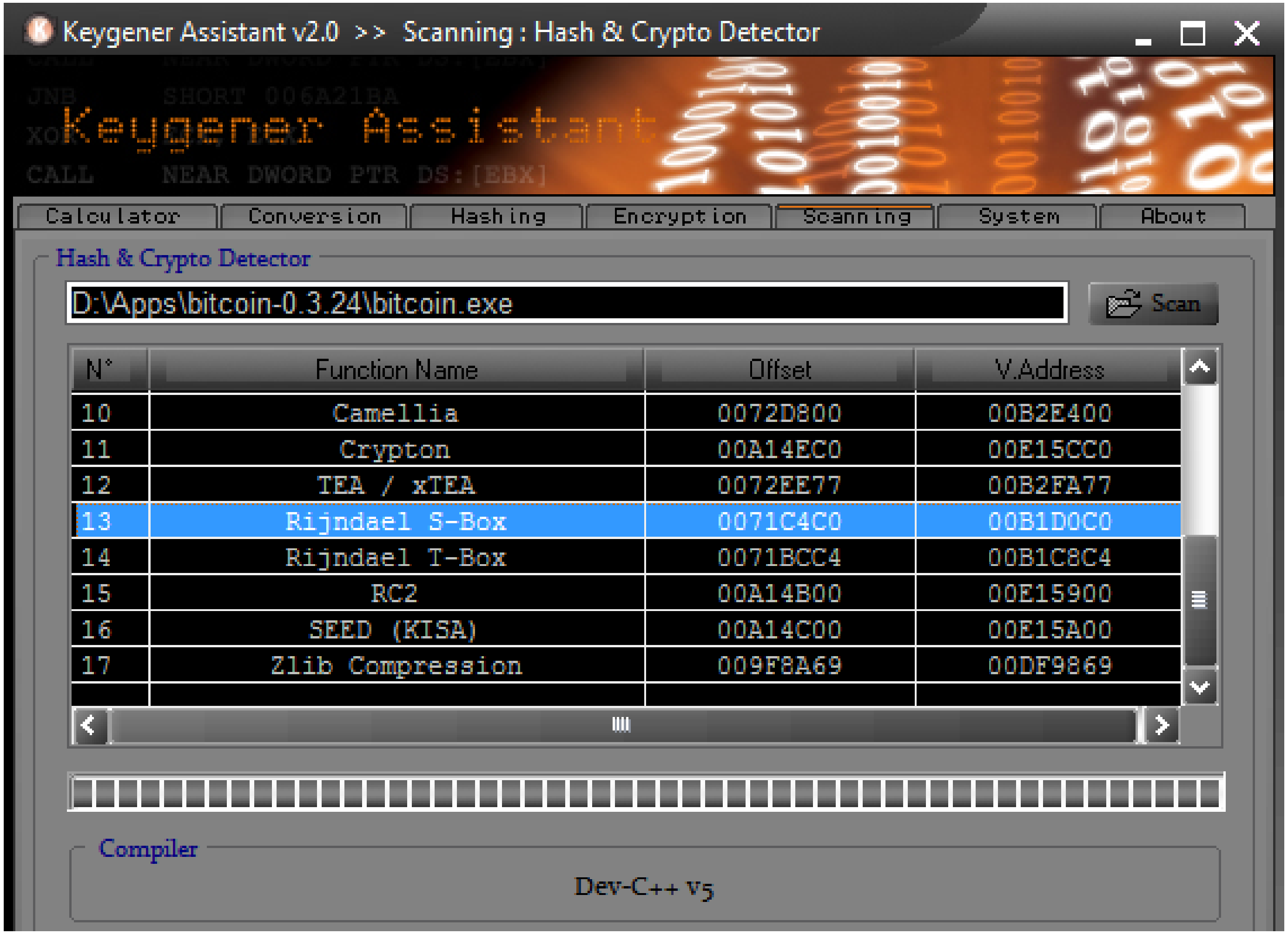
OllyDbg – key is visible in memory



```
011D1000  $ 55            PUSH EBP
011D1001  . 8BEC          MOV EBP,ESP
011D1003  . 83EC 10       SUB ESP,10
011D1006  . 56            PUSH ESI
011D1007  . 833D 20501D01 CMP DWORD PTR DS:[aes_init_done],0
011D100E  .v75 0F         JNZ SHORT AES_Pola.011D101F
011D1010  . E8 1B1A0000   CALL AES_Pola.aes_gen_tables
011D1015  . C705 20501D01 MOV DWORD PTR DS:[aes_init_done],1
011D101F  > 8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
011D1022  . 8945 F4       MOV DWORD PTR SS:[EBP-C],EAX
011D1025  . 817D F4 80000 CMP DWORD PTR SS:[EBP-C],80
011D102C  .v74 14         JE SHORT AES_Pola.011D1042
011D102E  . 817D F4 C0000 CMP DWORD PTR SS:[EBP-C],0C0
011D1035  .v74 16         JE SHORT AES_Pola.011D104D
011D1037  . 817D F4 00010 CMP DWORD PTR SS:[EBP-C],100
011D103E  .v74 18         JE SHORT AES_Pola.011D1058
011D1040  .vEB 21         JMP SHORT AES_Pola.011D1063
011D1042  > 8B4D 08       MOV ECX,DWORD PTR SS:[EBP+8]
011D1045  . C701 0A000000 MOV DWORD PTR DS:[ECX],0A
011D104B  .vEB 20         JMP SHORT AES_Pola.011D106D
011D104D  > 8B55 08       MOV EDX,DWORD PTR SS:[EBP+8]
011D1050  . C702 0C000000 MOV DWORD PTR DS:[EDX],0C
011D1056  .vEB 15         JMP SHORT AES_Pola.011D106D
011D1058  > 8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
011D105B  . C700 0E000000 MOV DWORD PTR DS:[EAX],0E
011D1061  .vEB 0A         JMP SHORT AES_Pola.011D106D
011D1063  > B8 00F8FFFF   MOV EAX,-800
011D1068  .vE9 0B060000   JMP AES_Pola.011D1678
011D106D  > 8B4D 08       MOV ECX,DWORD PTR SS:[EBP+8]
011D1070  . 83C1 08       ADD ECX,8
011D1073  . 894D FC       MOV DWORD PTR SS:[EBP-4],ECX
011D1076  . 8B55 08       MOV EDX,DWORD PTR SS:[EBP+8]
011D1079  . 8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
011D107C  . 8942 04       MOV DWORD PTR DS:[EDX+4],EAX
011D107F  . C745 F8 00000 MOV DWORD PTR SS:[EBP-8],0
011D1086  .vEB 09         JMP SHORT AES_Pola.011D1091
011D1088  > 8B4D F8       MOV ECX,DWORD PTR SS:[EBP-8]
011D108B  . 83C1 01       ADD ECX,1
011D108E  . 894D F8       MOV DWORD PTR SS:[EBP-8],ECX
011D1091  > 8B55 10       MOV EDX,DWORD PTR SS:[EBP+10]
011D1094  . C1FA 05       SAR EDX,5
011D1097  . 3955 F8       CMP DWORD PTR SS:[EBP-8],EDX
ESI=00000001
```

```
0020F97F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020F98F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020F99F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020F9AF 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .000000000000000
0020F9BF 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 0000000000000000
0020F9CF 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 0000000000000000
0020F9DF 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 0000000000000000
0020F9EF 01 53 65 63 75 72 65 50 61 73 73 77 6F 72 64 3A 0SecurePassword:
0020F9FF 6E 62 75 31 32 33 00 1D 01 58 72 1D 01 5C 72 1D nbu123.#0Xr#0\r#
0020FA0F 01 60 72 1D 01 00 00 00 00 20 FA 20 00 F8 2F 1D 0`r#0....  · .°/#
0020FA1F 01 60 FA 20 00 05 32 1D 01 01 00 00 00 D8 E1 4F 0`  .‡2#00...ïßO
0020FA2F 00 F8 7C 4F 00 7C 75 74 DB 00 00 00 00 00 00 00 .°¦O.¦ut█.......
0020FA3F 00 00 E0 FD 7E 00 00 00 00 34 FA 20 00 58 01 00 ..Ó²˜....4·  .X0.
0020FA4F 00 9C FA 20 00 89 36 1D 01 34 CD 49 DA 00 00 00 .£·  .ë6#04=I┌...
0020FA5F 00 6C FA 20 00 AA 33 68 76 00 E0 FD 7E AC FA 20 .l·  .¬3hv.Ó²˜¼·
0020FA6F 00 F2 9E EE 76 00 E0 FD 7E 77 18 E1 52 00 00 00 .=×¯v.Ó²˜w↑ßR...
0020FA7F 00 00 00 00 00 00 E0 FD 7E 00 00 00 00 00 00 00 ......Ó²˜.......
0020FA8F 00 00 00 00 00 78 FA 20 00 00 00 00 00 FF FF FF .....x·  .....
0020FA9F FF D5 71 F2 76 EB 27 2C 24 00 00 00 00 C4 FA 20  'q=vÙ',$....─·
0020FAAF 00 C5 9E EE 76 6D 32 1D 01 00 E0 FD 7E 00 00 00 .┼×¯vm2#0.Ó²˜...
```

What if AES usage is somehow hidden?



D:\Apps\bitcoin-0.3.24\bitcoin.exe

N°	Function Name	Offset	V.Address
10	Camellia	0072D800	00B2E400
11	Crypton	00A14EC0	00E15CC0
12	TEA / xTEA	0072EE77	00B2FA77
13	Rijndael S-Box	0071C4C0	00B1D0C0
14	Rijndael T-Box	0071BCC4	00B1C8C4
15	RC2	00A14B00	00E15900
16	SEED (KISA)	00A14C00	00E15A00
17	Zlib Compression	009F8A69	00DF9869

Compiler: Dev-C++ v5

Whitebox attacker model

- The attacker is able to:
  - inspect and disassemble binary (static strings, code...)
  - observe/modify all executed instructions (OllyDbg...)
  - observe/modify used memory (OllyDbg, memory dump...)
- How to still protect value of cryptographic key?
- Who might be white-box attacker?
  - Mathematician (for fun)
  - Security researcher / Malware analyst (for work)
  - DRM cracker (for fun&profit)
  - ...

Classical obfuscation and its limits

- Time-limited protection
- Obfuscation is mostly based on obscurity
  - add bogus jumps
  - reorder related memory blocks
  - transform code into equivalent one, but less readable
  - pack binary into randomized virtual machine
  - ...
- Barak’s (im)possibility result (2001)
  - family of functions that will always leak some information
  - but practical implementation may exists for others

Computation with Encrypted Data and Encrypted Function

CEF&CED

Scenario

- We’d like to compute function F over data D
  - secret algorithm F or sensitive data D (or both)
- Solution with trusted environment
  - my trusted PC, trusted server, trusted cloud…
- Problem: can be cloud or client really trusted?
  - server hack, DRM, malware...
- Attacker model
  - controls execution environment (debugging)
  - sees all instructions and data executed

CEF

- Computation with Encrypted Function (CEF)
  - A provides function F in form of P(F)
  - P can be executed on B’s machine with B’s data D as P(D)
  - B will not learn function F during computation

CED

- Computation with Encrypted Data (CED)
  - B provides encrypted data D as E(D) to A
  - A is able to compute its F as F(E(D)) to produce E(F(D))
  - A will not learn D

CED via homomorphism

1. Convert your function into circuit with additions (`xor`) and multiplications (`and`) only
2. Compute addition and/or multiplication “securely”
   - an attacker can compute E(D1+D2) = E(D1)+E(D2)
   - but will learn neither D1 nor D2
3. Execute whole circuit over encrypted data

- Partial homomorphic scheme
  - either addition or multiplication is possible, but not both
- Fully homomorphic scheme
  - both addition and multiplication (unlimited)

Partial homomorphic schemes

- Example with RSA (*multiplication*)
  - $E(d_1).E(d_2) = d_1^e . d_2^e \bmod m = (d_1 d_2)^e \bmod m = E(d_1 d_2)$
- Example Goldwasser-Micali (*addition*)
  - $E(d_1).E(d_2) = x^{d1} r_1^2 \cdot X^{d2} r_2^2 = x^{d1+d2}(r_1 r_2)^2 = E(d_1 \oplus d_2)$
- Limited to polynomial and rational functions
- Limited to only one type of operation (*mult* or *add*)
  - or one type and very limited number of other type
- Slow – based on modular mult or exponentiation
  - every operation equivalent to whole RSA operation

Fully homomorphic scheme (FHE)

- Holy grail - idea proposed in 1978 (Rivest et al.)
  - both addition and multiplication securely
- But no scheme until 2009 (Gentry)!
  - based on lattices over integers
  - noisy FHE usable only to few operations
  - combined with repair operation

Fully homomorphic scheme - usages

- Outsourced cloud computing and storage (FHE search)
  - Private Database Queries
  - using Somewhat Homomorphic Encryption http://researcher.ibm.com/researcher/files/us-shaih/privateQueries.pdf
  - protection of the query content
- Secure voting protocols (yes/no + sum)
- Protection of proprietary info - MRI machines
  - very expensive algorithm analyzing MR data, HW protected
  - central processing restricted due to processing of private patient data
- Read more about current state of FHE
  - http://www.americanscientist.org/issues/id.15906,y.2012,no.5,content.true,page.2,css.print/issue.aspx

Fully homomorphic scheme - practicality

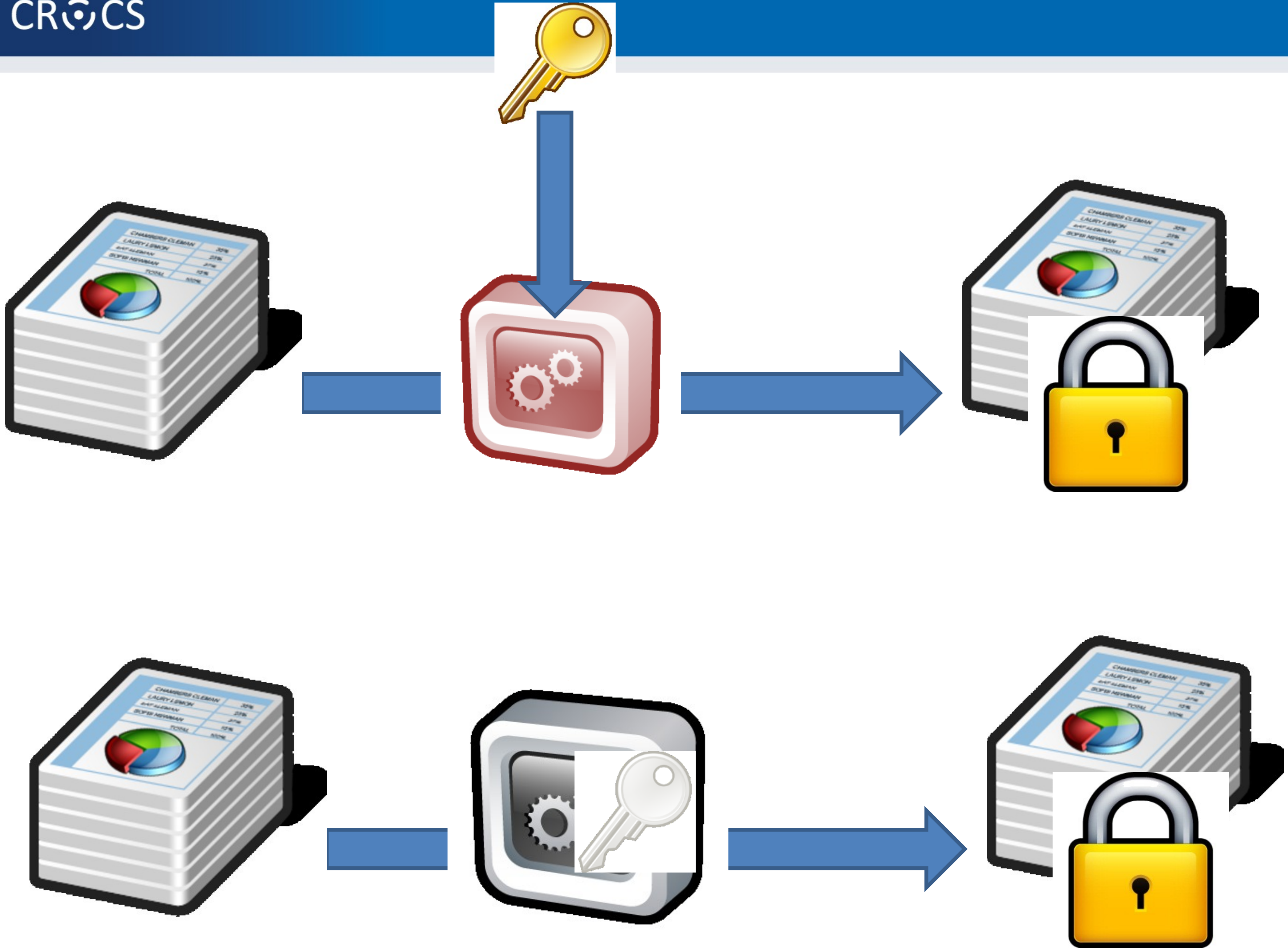
- Not very practical (yet ☺) (Gentry, 2009)
  - 2.7GB key & 2h computation for every repair operation
  - repair needed every ~10 multiplication
- FHE-AES implementation (Gentry, 2012)
  - standard PC ⇒ 37 minutes/block (but 256GB RAM)

Computation with Encrypted Data and Encrypted Function

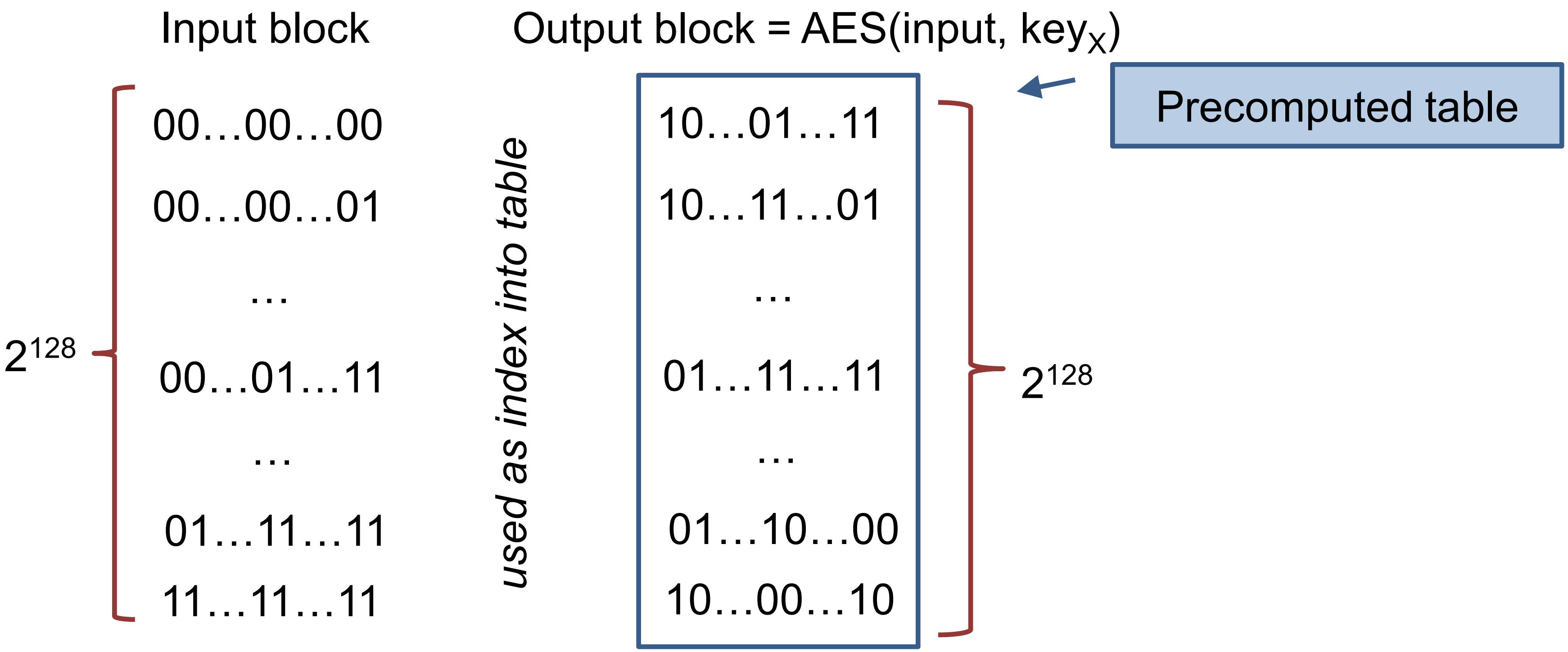
WHITEBOX CRYPTOGRAPHY

White-box attack resistant cryptography

- Problem limited from every cipher to symmetric cryptography cipher only
  - protects used cryptographic key (and data)
- Special implementation fully compatible with standard AES/DES… 2002 (Chow et al.)
  - series of lookups into pre-computed tables
- Implementation of AES which takes only data
  - key is already embedded inside
  - hard for an attacker to extract embedded key



Impractical solution



Input block

Output block = AES(input, key$_X$)

Precomputed table

2^{128}

00…00…00

00…00…01

…

00…01…11

…

01…11…11

11…11…11

used as index into table

10…01…11

10…11…01

…

01…11…11

…

01…10…00

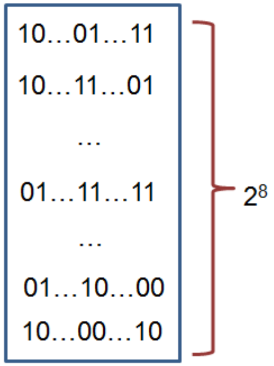
10…00…10

2^{128}

- Secure, but 2^{128} x 16B memory storage

WBACR AES – some techniques

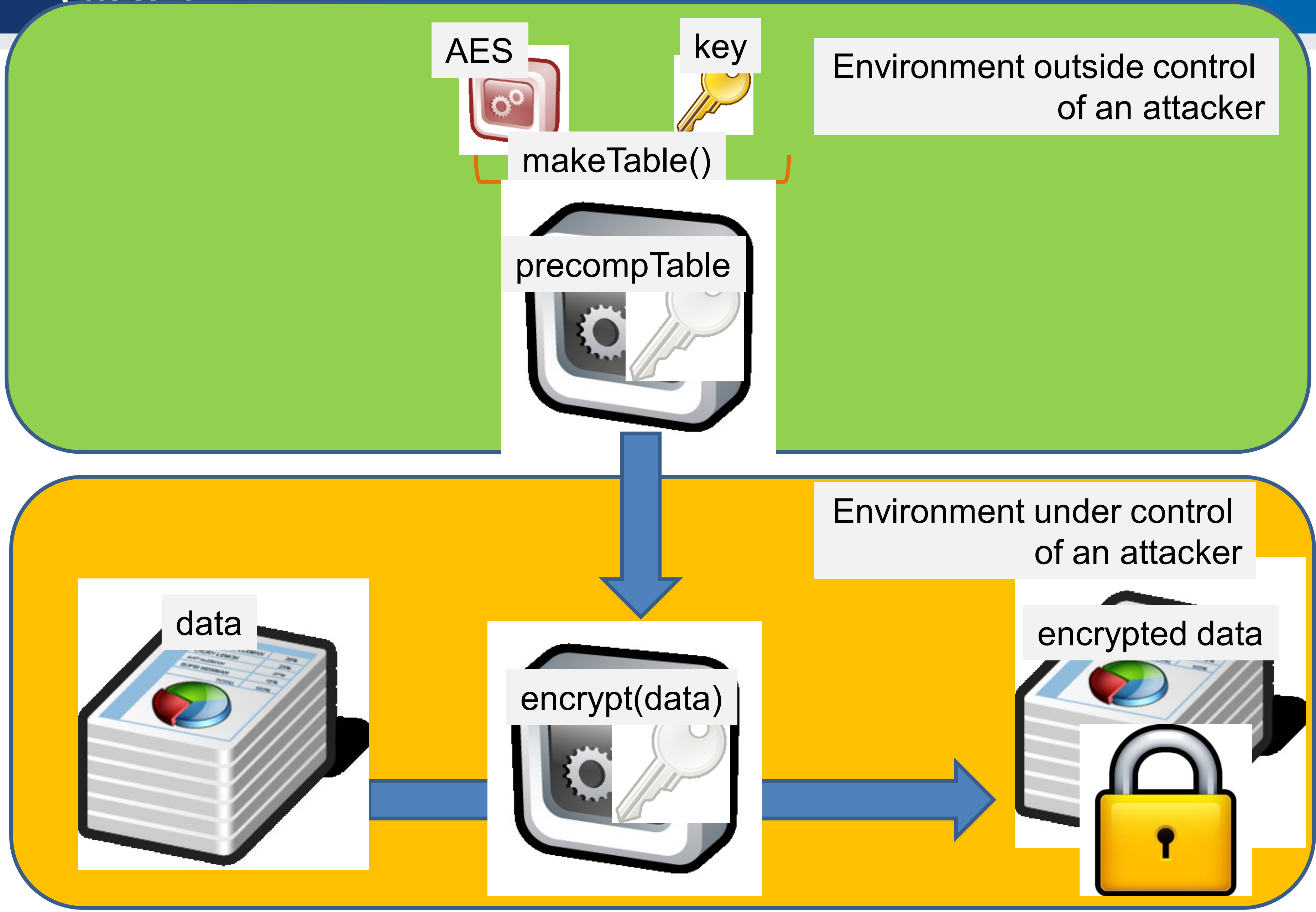
- Pre-compute table for all possible inputs
  - practical for one 16bits or two 8bits arguments table with up to 2^{16} rows (~64KB)
  - **`AddRoundKey: data ⊕ key`**
    - 8bit argument data, key fixed



- Pack several operations together
  - **`AddRoundKey+SubBytes: T[i] = S[i ⊕ key];`**
- Protect intermediate values by random bijections
  - removed automatically by next lookup
  - **X = F^{-1}(F(X))**
  - **T[i] = S[F^{-1}(i) ⊕ key];**

Whitebox cryptography lifecycle

- [Secure environment]
  1. Generate required key (random, database...)
  2. Generate WAES tables (in secure environment)
- [Potential insecure environment]
  3. Compile WAES tables into target application
- [Insecure environment (User PC)]
  4. Run application and use WAES as usual (with fixed key)



AES

key

Environment outside control of an attacker

makeTable()

precompTable

Environment under control of an attacker

data

encrypt(data)

encrypted data

Resulting implementation

- More difficult to detect that crypto was used
  - no fixed constants in the code
  - precomputed tables change with every generation
  - even two tables for same key are different
  - (but can still be found)
- Resistant even when precomputed tables are found
  - when debugged, only table lookups are seen
  - key value is never manipulated in plaintext
  - transformation techniques should provide protection to key embedded inside tables

WBACR AES - pros

- Performance is practically usable
  - implementation size ~800KB (tables)
  - speed ~MBs/sec (~6.5MB/s vs. 220MB/s)
- Hard to extract embedded key
  - Complexity semi-formally guaranteed
  - (if the scheme is secure)
- One can simulate asymmetric cryptography!
  - implementation contains only encryption part of AES
  - until attacker extracts key, decryption is not possible

WBACR AES - cons

- Implementation can be used as oracle (black box)
  - attacker can supply inputs and obtain outputs
  - even if she cannot extract the key
  - (can be partially solved by I/O encodings)
- Problem of secure input/output
  - protected is only AES, not code around
- Key is fixed and cannot be easily changed
- Successful cryptanalysis for several schemes
  - several former schemes broken
  - new techniques proposed
- Fault induction attacks (2015, Riscure)!

List of proposals and attacks

- (2002) First WB AES implementation by Chow et. al. [Chow02]
  - IO bijections, linear mixing bijections, external coding
  - broken by BGE cryptanalysis [Bill04]
    - algebraic attack, recovering symmetric key by modelling round function by system of algebraic equations
- (2006) White Box Cryptography: A New Attempt [Bri06]
  - attempt to randomize whitebox primitives, perturbation & random equations added, S-boxes are enc. keys. 4 AES ciphers, major voting for result
  - broken by Mulder et. al. [Mul10]
    - removes perturbations and random equations, attacking on final round removing perturbations, structural decomposition. 2^{17} steps
- (2009) A Secure Implementation of White-box AES [Xia09]
  - broken by Mulder et. al. [Mul12]
    - linear equivalence algorithm used (backward AES-128 compatibility => linear protection has to be inverted in next round), 2^{32} steps
- (2011) Protecting white-box AES with dual ciphers [Kar11]
  - broken by our work [Kli13]
    - protection shown to be ineffective

More resources

- Overviews, links
  - http://whiteboxcrypto.com/research.php
  - https://minotaur.fi.muni.cz:8443/~xsvenda/docuwiki/doku.php?id=public:mobilecrypto
- Crackme challenges
  - http://www.phrack.org/issues.html?issue=68&id=8
- Whitebox crypto in DRM
  - http://whiteboxcrypto.com/files/2012_MISC_DRM.pdf

Whitebox transform IS used in the wild

- Proprietary DRM systems
  - details are usually not published
  - AES-based functions, keyed hash functions, RSA, ECC...
  - interconnection with surrounding code
- Chow at al. (2002) proposal made at Cloakware
  - firmware protection solution
- Apple's FairPlay & Brahms attack
  - http://whiteboxcrypto.com/files/2012_MISC_DRM.pdf
- ...

SUMMARY

Summary

Questions

- Dynamic libraries can be forged
  - make DLL preloding harder (manifest)
  - check input from library as untrusted
- Don’t use standard C functions for temporary files
  - not use temporary files at all or follow security guidelines
- Try to protect secrets inside binary
  - don’t hardcode any secrets
  - offload sensitive computation to secure environment (server, smart card, HSM)
  - use whitebox-attacker protection techniques