PA193 - Secure coding principles and practices

Protecting integrity of modules and external components

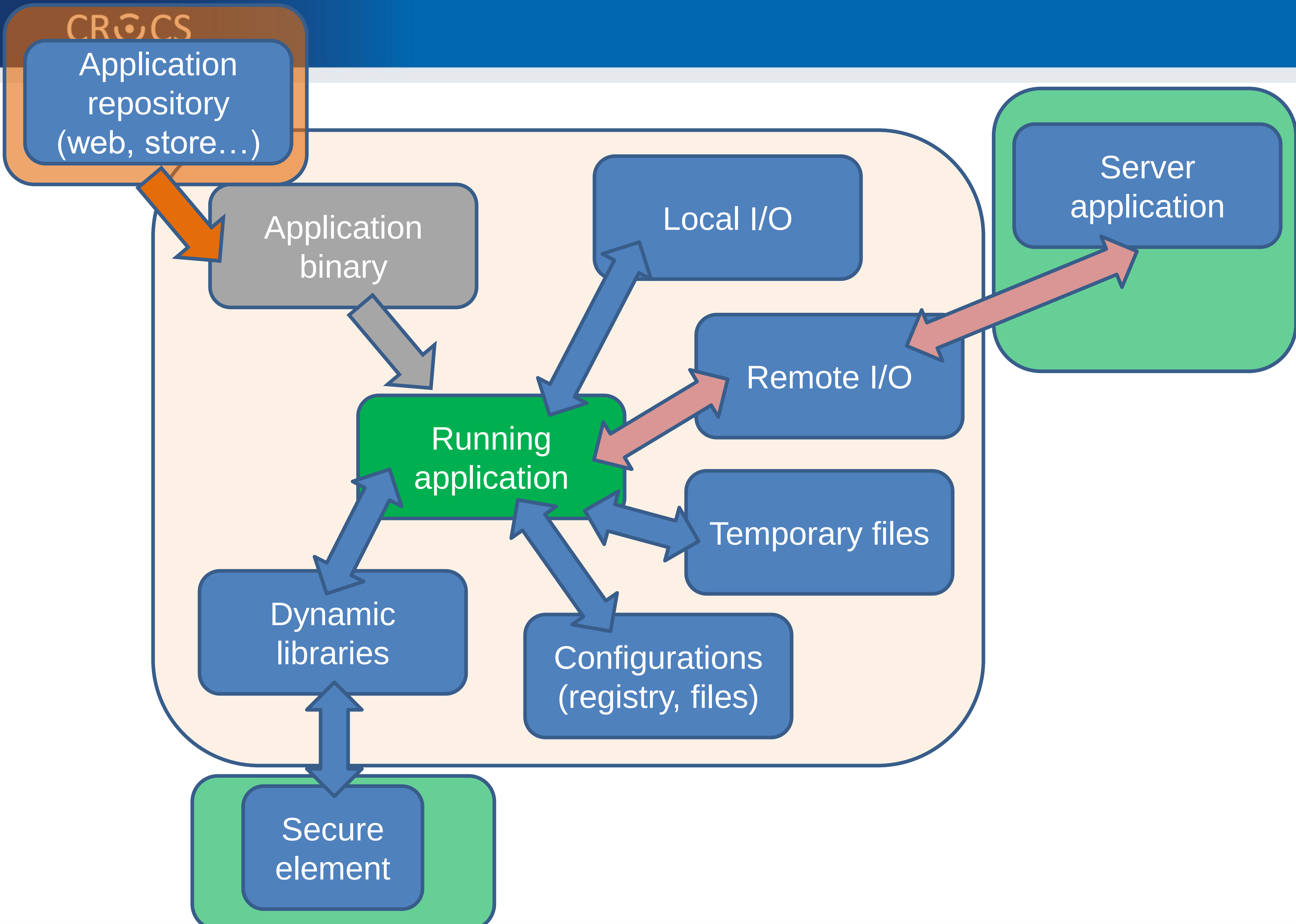
Petr Švenda svenda@fi.muni.cz

Centre for Research on Cryptography and Security

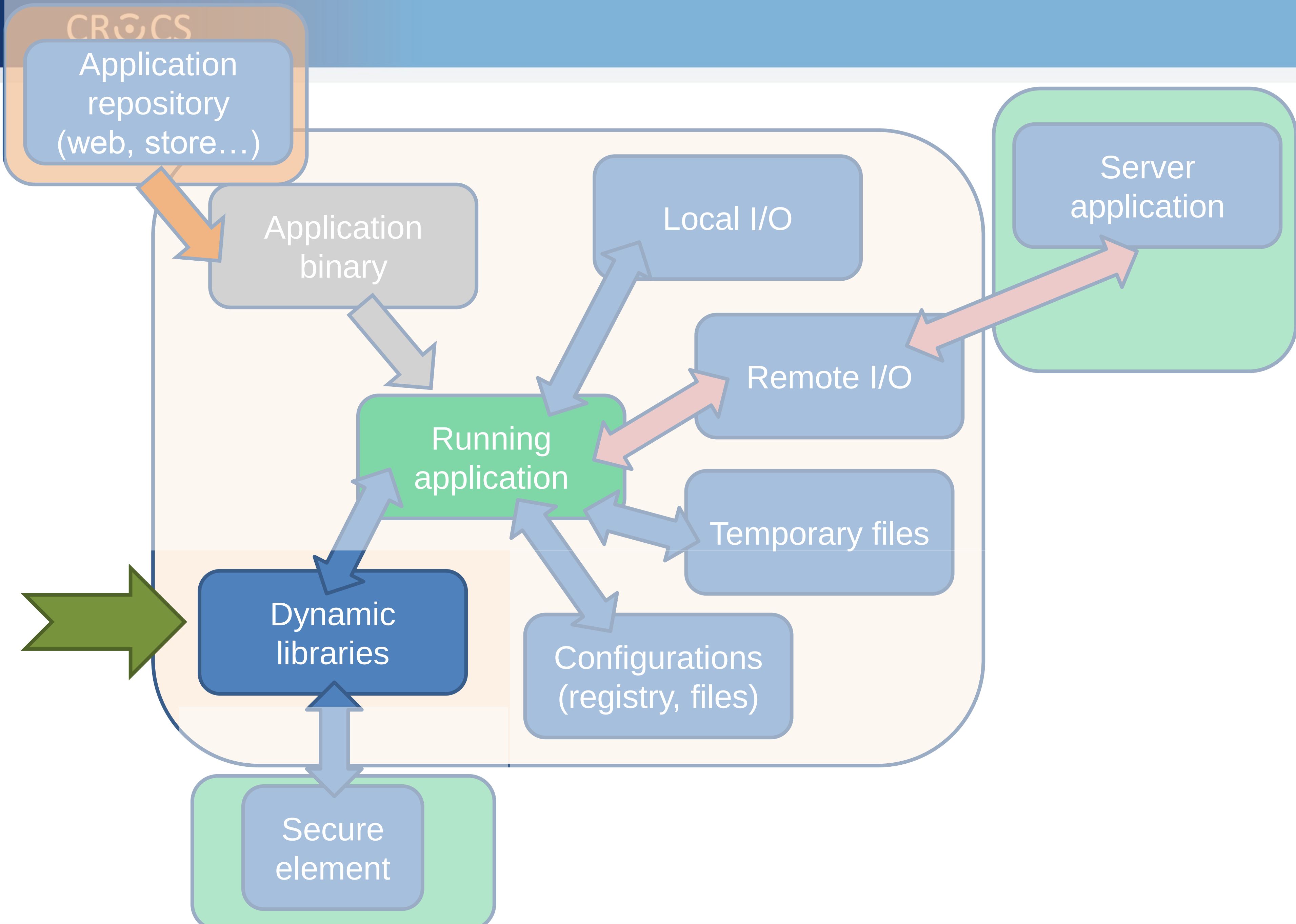
Overview

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

PROBLEM


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

DYNAMIC LIBRARIES



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

Dynamic library usage

- 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*)**
  - **dlopen(path) & dlclose(hLib)**
- Run-time search for specific function
  - **GetProcAddress(*hLib, "function_name"*)**
  - **dlsym(*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 (from Windows XP SP2)

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

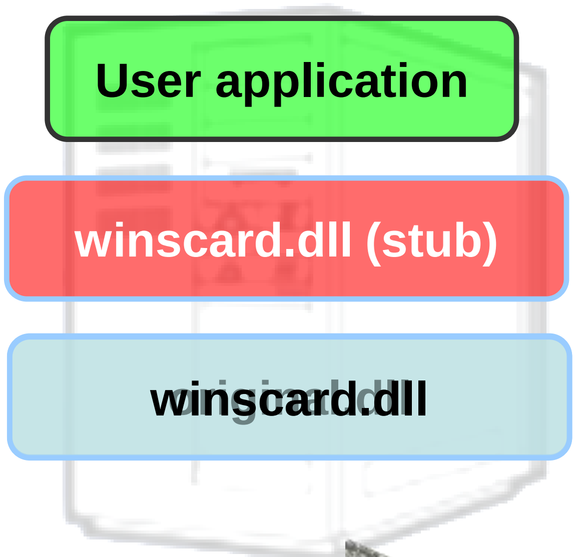
- Application likes to load dynamic library
  - according to specified name, e.g., winscard.dll
  - e.g. via LoadLibrary(“winscard.dll”) call
- Create dynamic library (“stub”) with the same name and the same set of exported functions
- Move stub DLL into directory where application looks first for requested 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

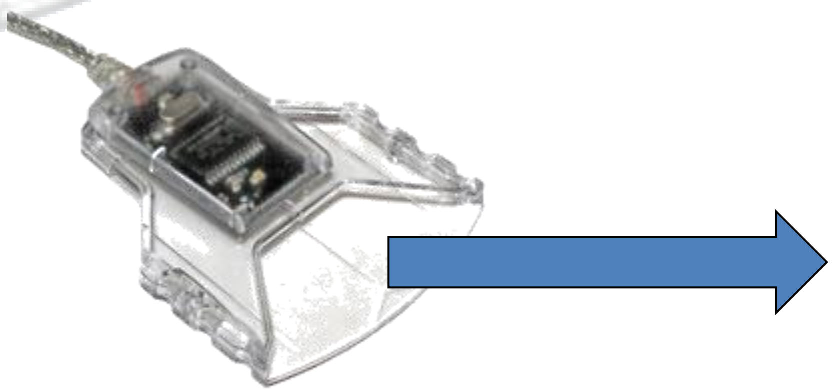
- Dynamic library for interception and manipulation of communication with smart cards
  - winscard.dll, APDU-based communication
  - http://www.fi.muni.cz/~xsvenda/apduinspect.html
- What you can achieve:
  - log input/output APDU commands (including keys, PINs...)
  - manipulate APDUs content according 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

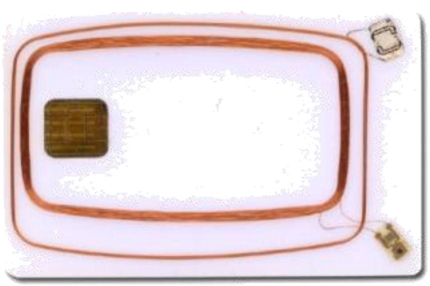
Let’s write own winscard.dll (PC/SC)

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
   - No changes to application binary are required (legacy)
   - 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 target DLL
   - good practice to install application DLLs in its directory
     - will not overwrite other versions of same DLL
   - (is overridden 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">
…
```

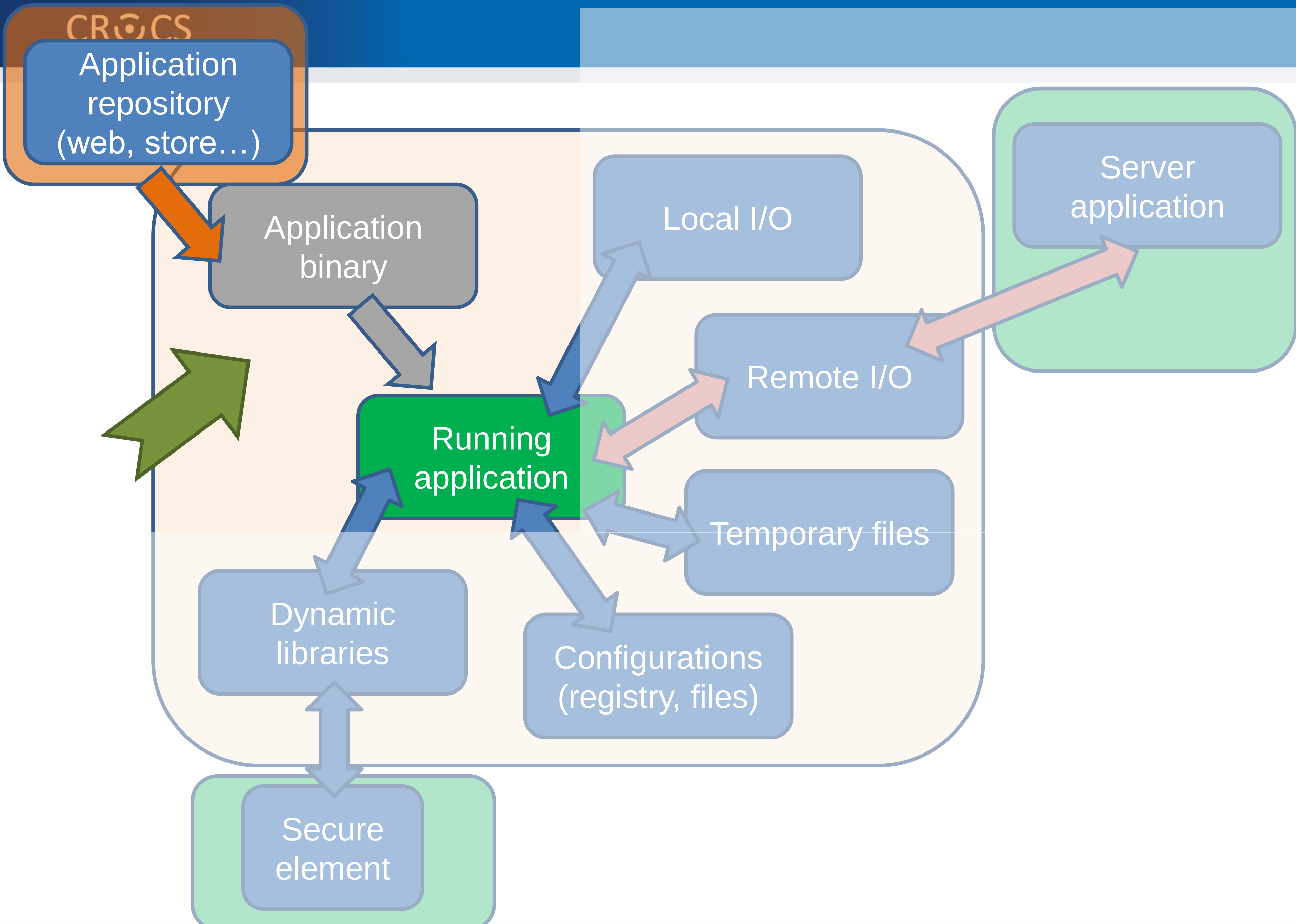
Security implications of dynamic libraries

- 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 (assumption about already verified inputs)

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


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

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, GCHQ Edgehill 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
   – signatures of package **`sign=RSA(private_key,your_package)`**
   – everybody can Verify(public_key, sign)
   – usually most suitable, but may require PKI

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
```

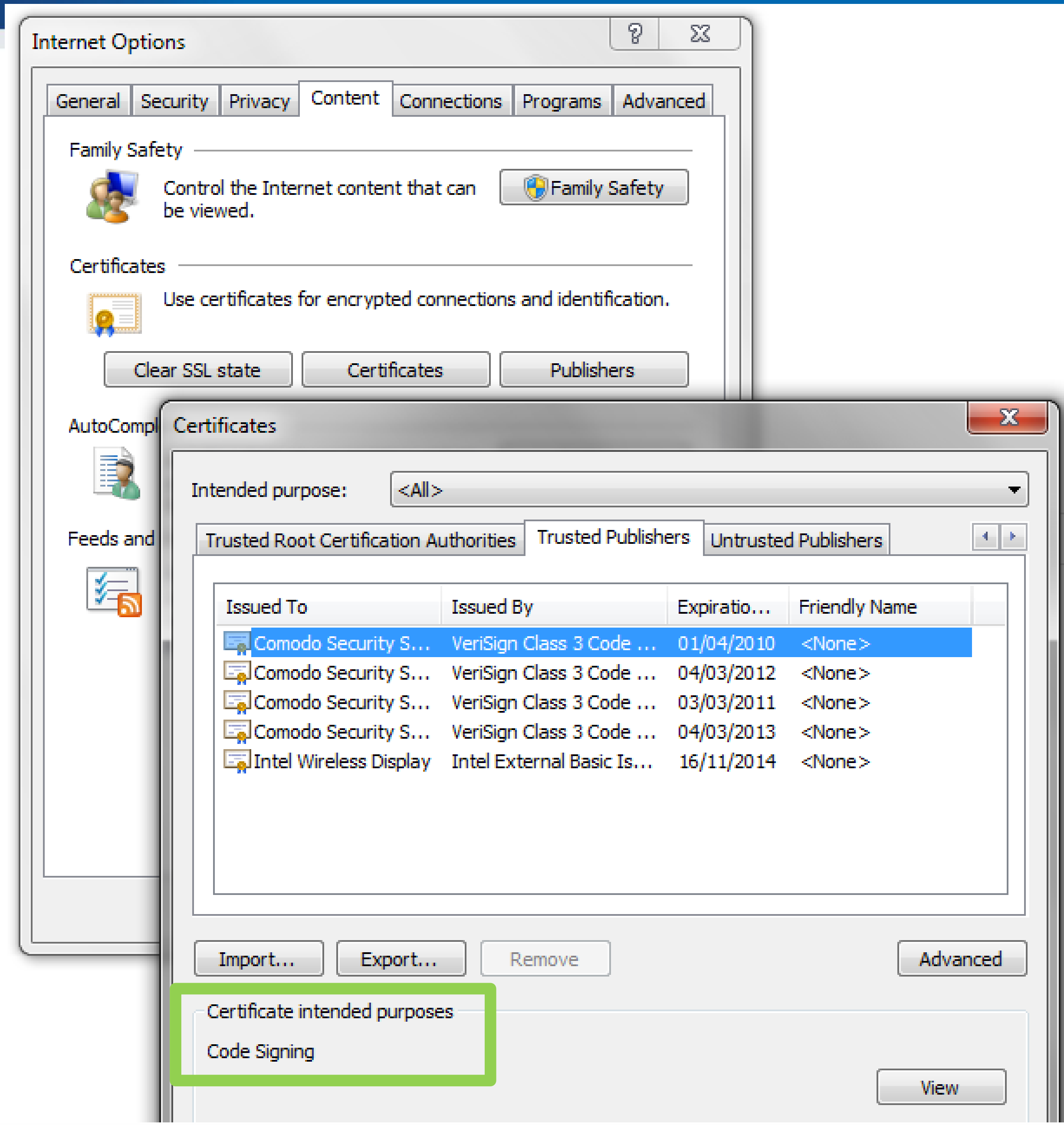
- Trust to signing key is needed
  - public key should be obtained from trusted source
  - but usually only publisher website ☹
- Can be used to sign packages (e.g., Debian RPM)
  - http://fedoranews.org/tchung/gpg/

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
- …

Code signing (Microsoft's Authenticode)

- Publisher obtains Code Signing Digital ID
  - X.509 certificate with public key signed by trusted authority
  - authority's certificate imported in Trusted Publishers
- Publisher creates code (application)
- Publisher signs code with its private key
- Application is distributed along with signature(s)
- Application signature is verified, user is notified
  - invalid signature of application → confirmation from user
  - invalid signature of driver → no installation
- (RSA 2048bit with SHA-1)


Internet Options
General
Security
Privacy
Content
Connections
Programs
Advanced
Family Safety
Control the Internet content that can be viewed.
Family Safety
Certificates
Use certificates for encrypted connections and identification.
Clear SSL state
Certificates
Publishers
AutoCompl
Feeds and
Certificates
Intended purpose:
<All>
Trusted Root Certification Authorities
Trusted Publishers
Untrusted Publishers
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>
Import...
Export...
Remove
Advanced
Certificate intended purposes
Code Signing
View

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/

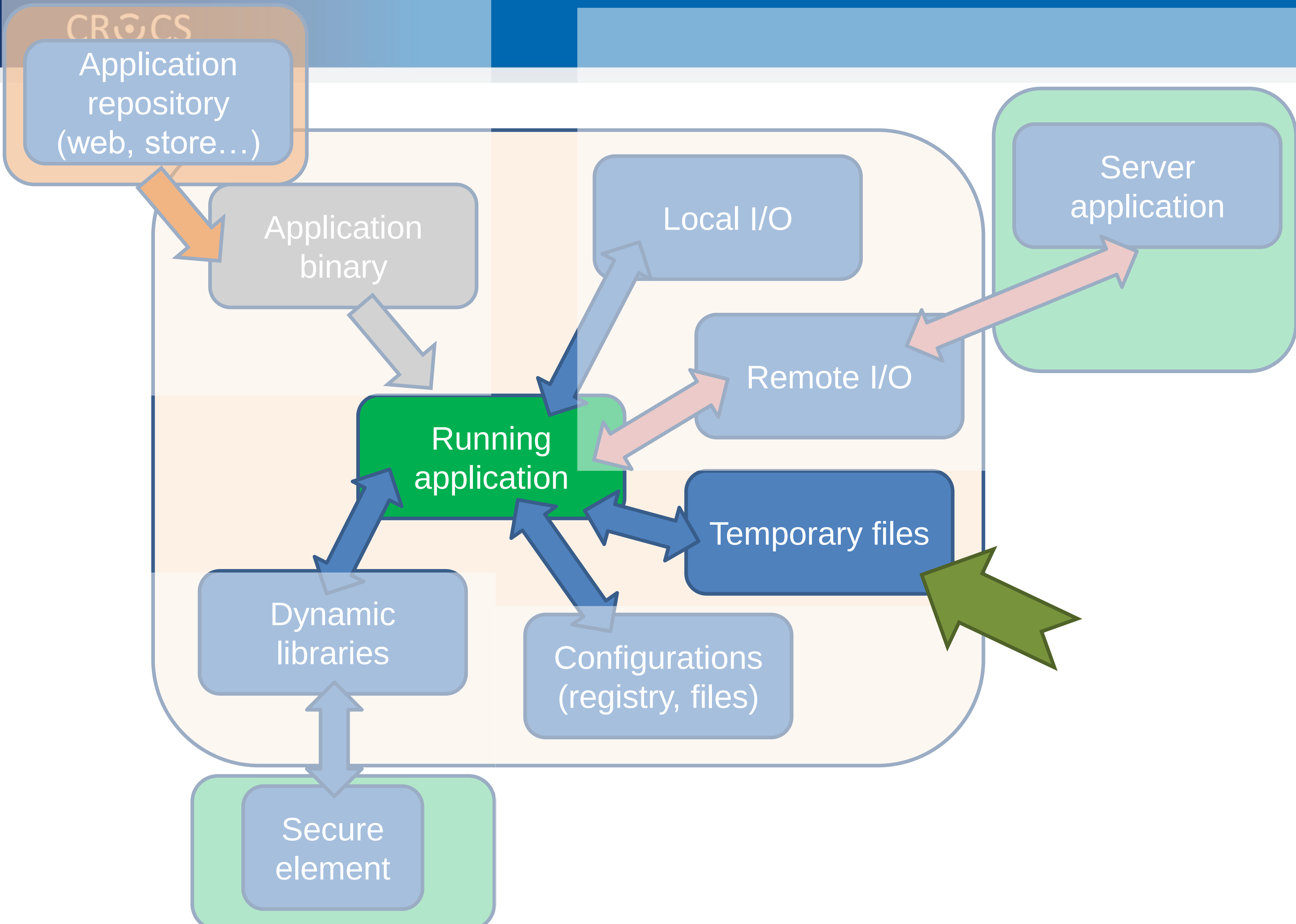
Microsoft Authenticode – selfsign (testing)

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

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


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

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
- **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
  - 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)

- 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** 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 perceived 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

Attack: overwrite protected system files

1. Privileged program will use tmp file in public directory
2. Attacker guess the tmp file name ‘t3oc.7’
3. Attacker creates symbolic link between ‘t3oc.7’ and protected system file (mklink t3oc.7 explorer.exe)
4. Privileged program tries to create tmp file but uses link to system file instead
5. Privileged program has enough privilege to write into system file explorer.exe
6. System file is corrupted

Attack: overwrite user own files

1. Unprivileged program will use tmp file in public directory
2. Attacker guess the tmp file name ‘t3oc.7’
3. Attacker creates symbolic link between ‘t3oc.7’ and other user file ‘user.txt’
4. Unprivileged program tries to create tmp file but uses link to user file instead
5. User file ‘user.txt’ is corrupted

Comparison of temporary file functions

	tmpnam	tmpnam_s	tmpfile	tmpfile_s	mktemp	mkstemp
Unique name	Yes	Yes	Yes	Yes	Yes	Yes
Atomic	No	No	Yes	Yes	No	Yes
Exclusive Access	Possible	Possible	No	If supported by OS	Possible	Yes
Appropriate permissions	Possible	Possible	No	If supported by OS	Possible	Not portably
File Removed	No	No	Yes *	Yes *	No	No
Un-predictable name	Not portably	Yes	Not portably	Yes	Not portably	Not portably

* **If the program terminates abnormally this behavior is implementation-defined.**

Temporary Files Secure Programming (Robert C. Seacord)
https://www.securecoding.cert.org/confluence/download/attachments/3524/07.5+Temporary+Files+v2.pdf

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 use shared directories if possible
4. Don’t store sensitive information in temp files
   - temp files are common attack vector, prevent it
5. Research where are temporary files stored
   - no standard function for that in C/C++
   - Windows: GetTempPath()

Temporary files security checklist (2)

6. Ensure strong uniqueness and non-predictability 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)

7. 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
8. 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)

9. 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)

10. 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)
11. 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

Example: not yet ☺ (LABs)

- obtain temporary directory
- create subdirectory, set proper ACL
- generate long random file name
- open file exclusively with absolute path
- generate random key
- encrypt data before write
- shred content (overwrite) before close
- close after use
- handle abnormal termination (signal, exception)

References

- Temporary Files Secure Programming (Robert C. Seacord)
  - https://www.securecoding.cert.org/confluence/download/attachments/3524/07.5+Temporary+Files+v2.pdf
- 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

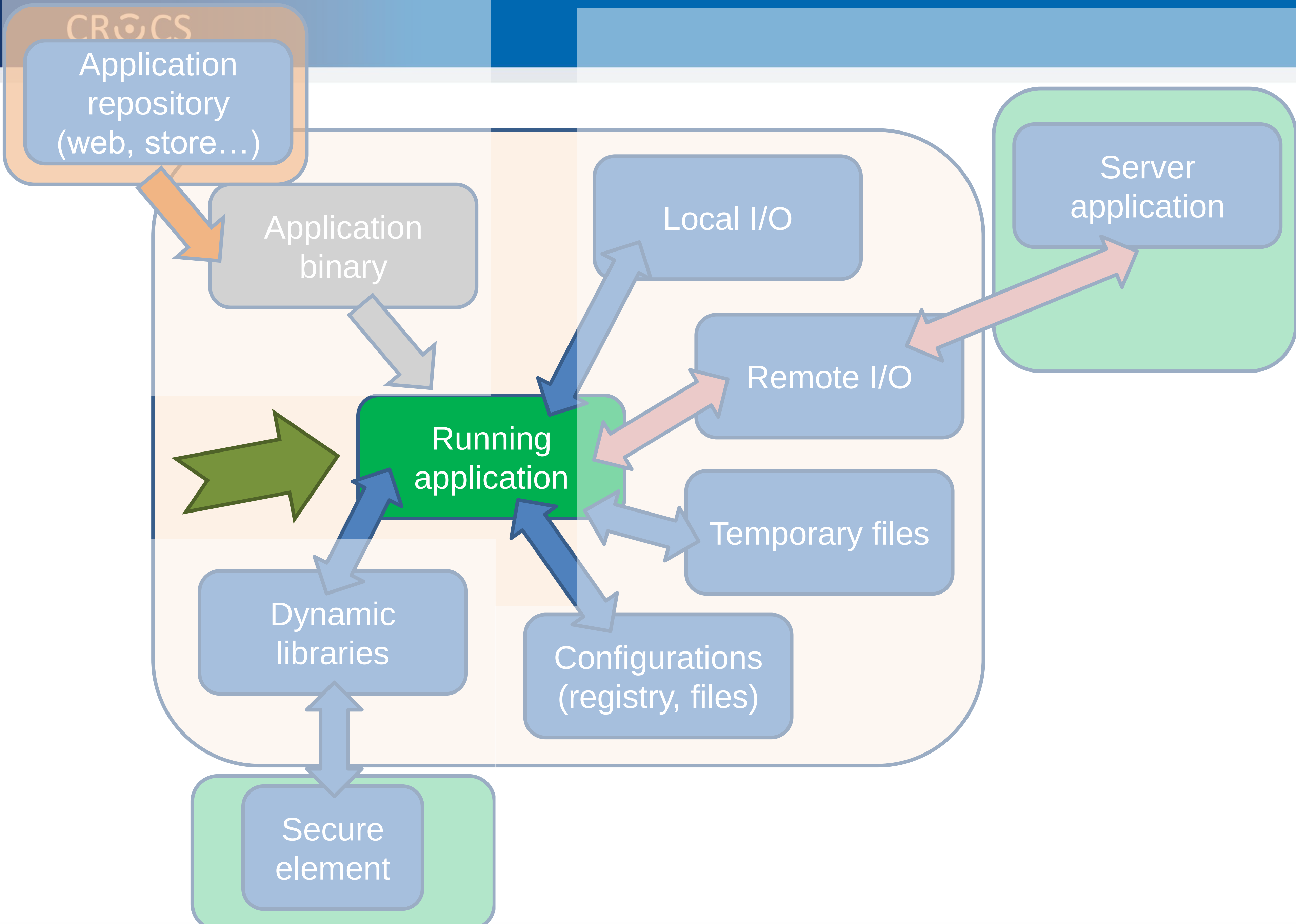
SUMMARY

Summary

Questions

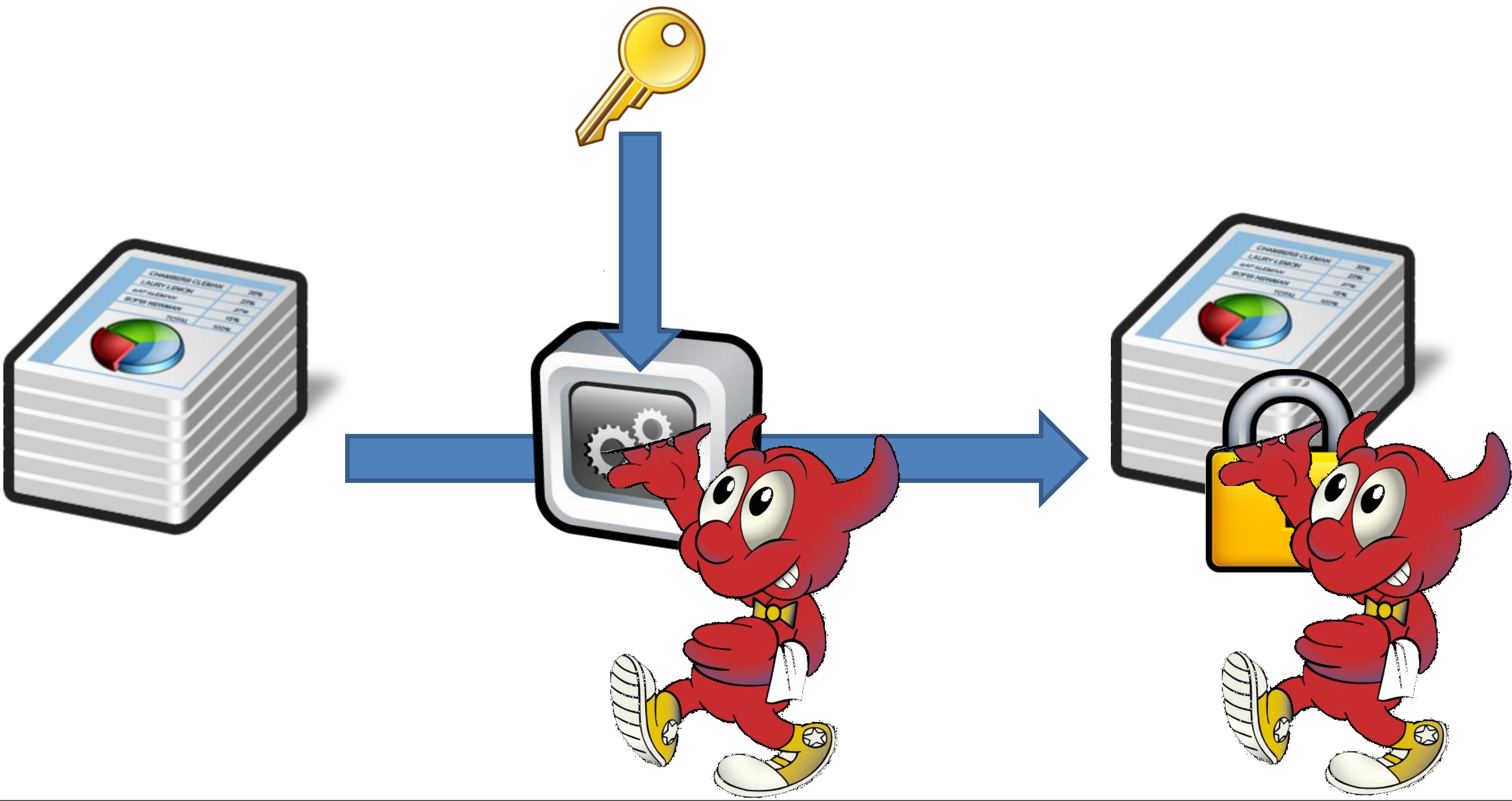
- Dynamic libraries can be forged
  - make DLL preloding harder (manifest)
  - check input from library as untrusted
- Don't use standard C functions for temporary files
  - not use temporary files at all or follow security guidelines
- Try to protect secrets inside binary
  - don't hardcode any secrets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

BONUS: PROTECTING SOFTWARE MODULES



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

Standard vs. whitebox attacker model



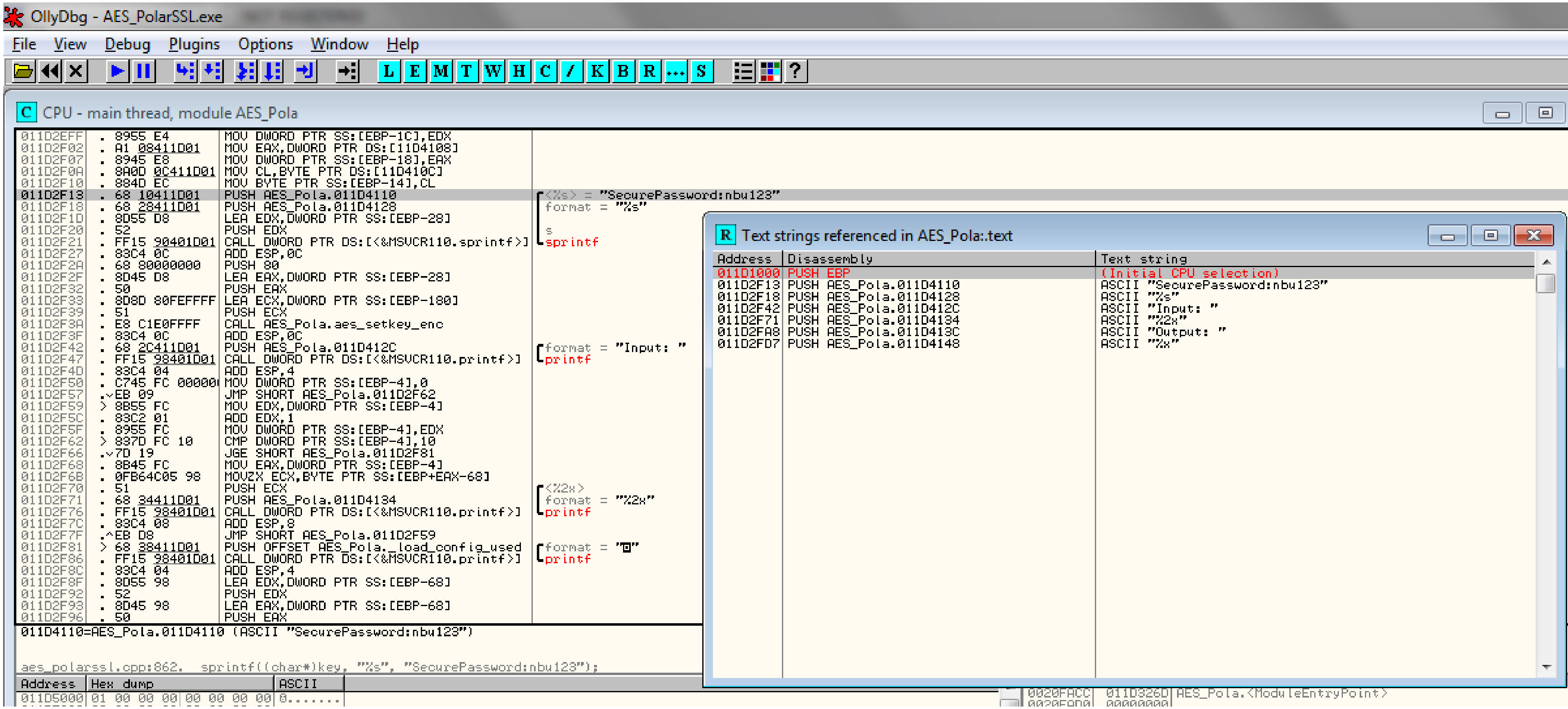
Standard AES API (PolarSSL)

```
/**
 * \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] );
```

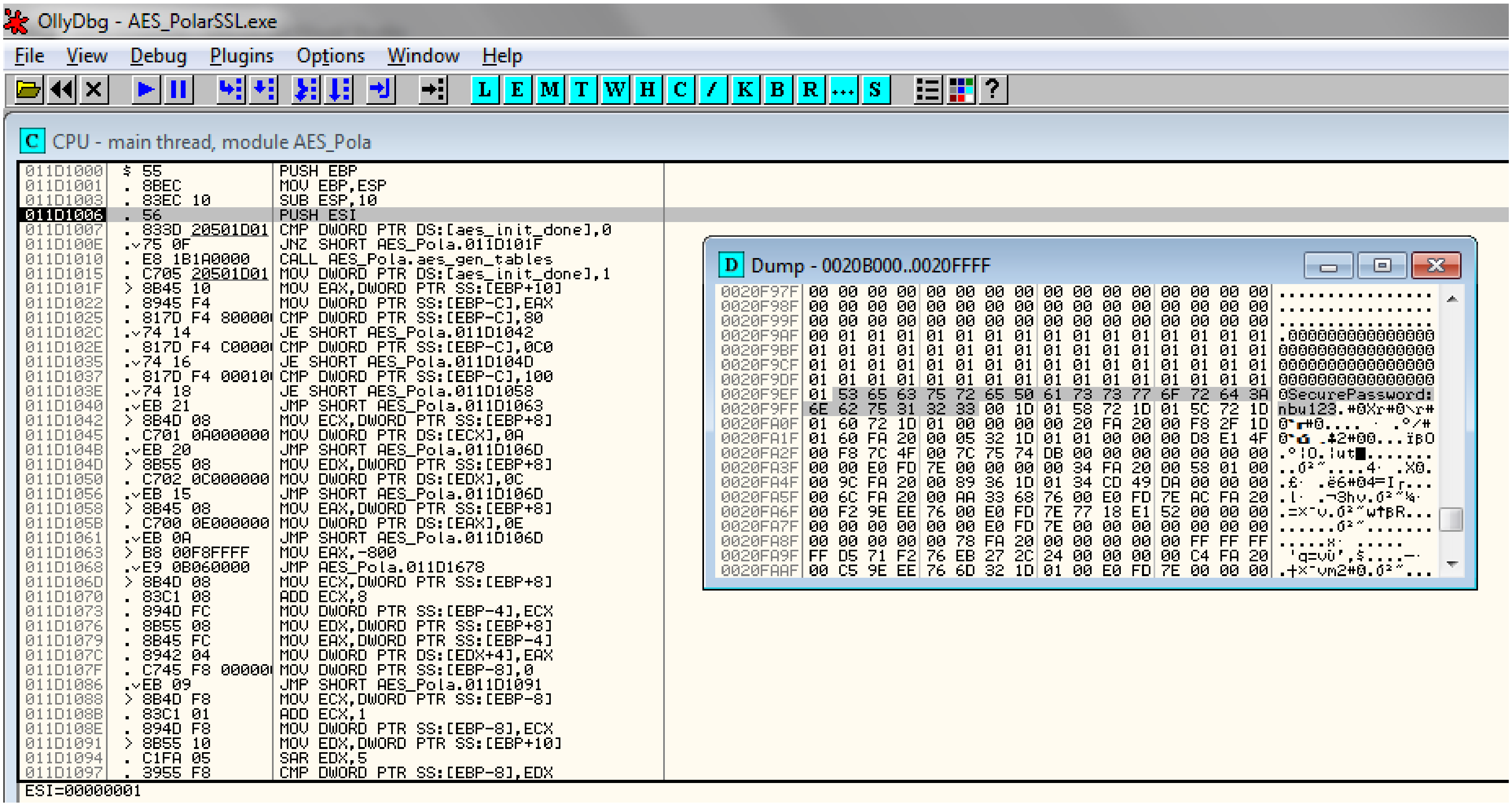
Standard AES - usage

```
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]);
}
```

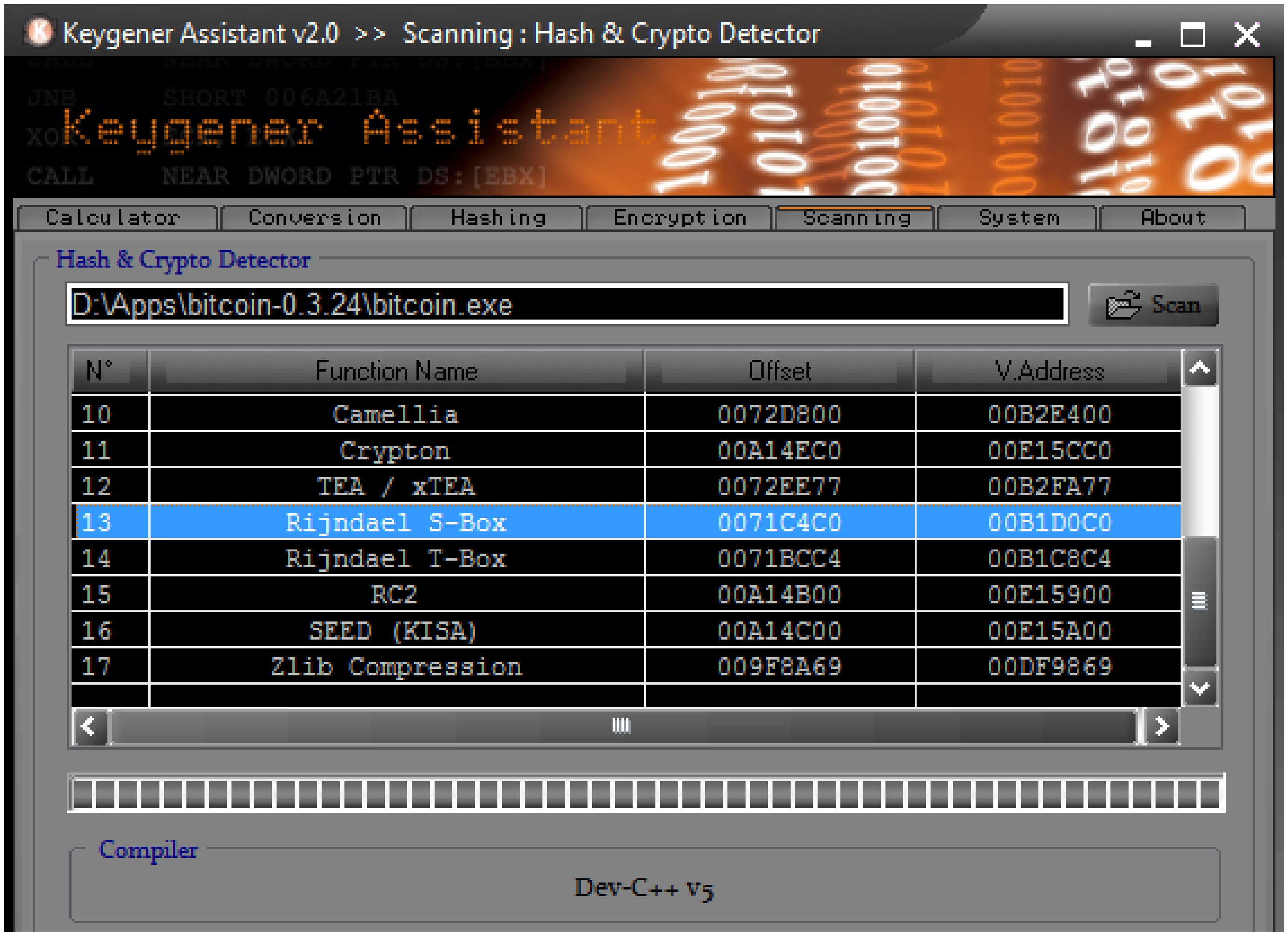
OllyDbg – key value is static string



OllyDbg – key is visible in memory



What if AES usage is somehow hidden?

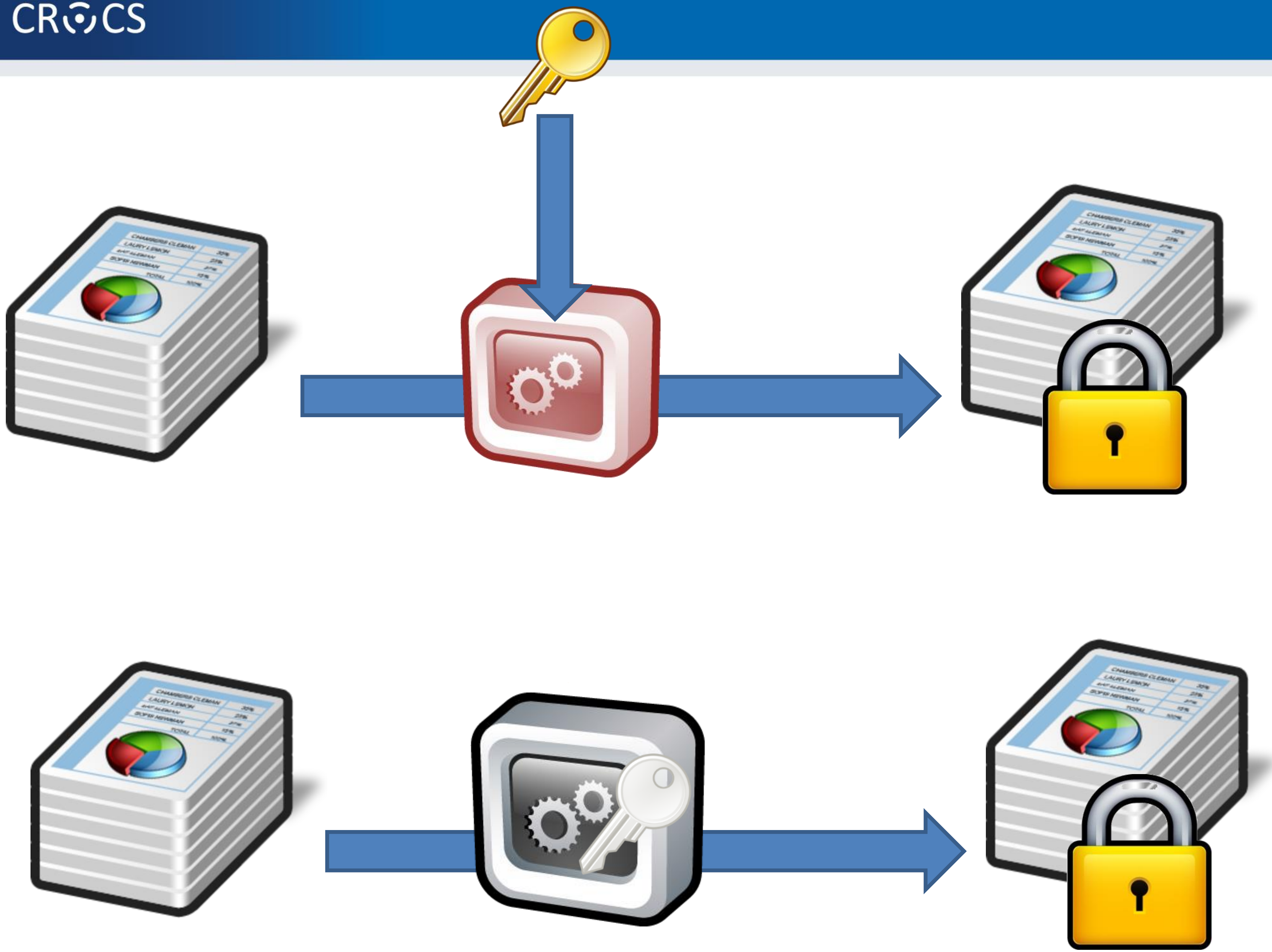


Whitebox attacker model

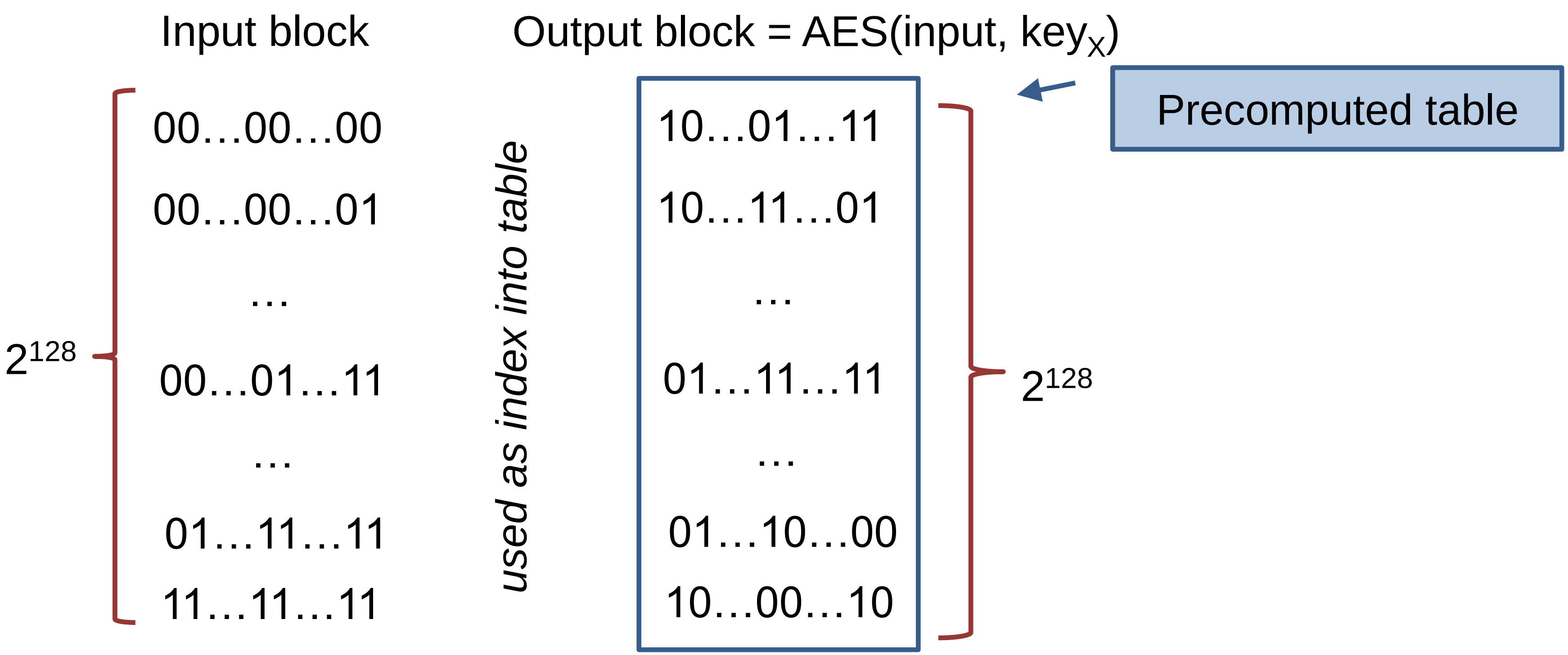
- The attacker is able to:
  - inspect and disassemble binary (static strings, code...)
  - observe/modify all executed instructions (OllyDbg...)
  - observe/modify used memory (OllyDbg, memory dump...)
- How to still protect value of cryptographic key?
- Who might be whitebox attacker?
  - Mathematician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

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



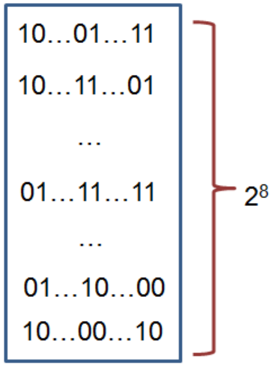
Impractical solution



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

WBACR AES – some techniques

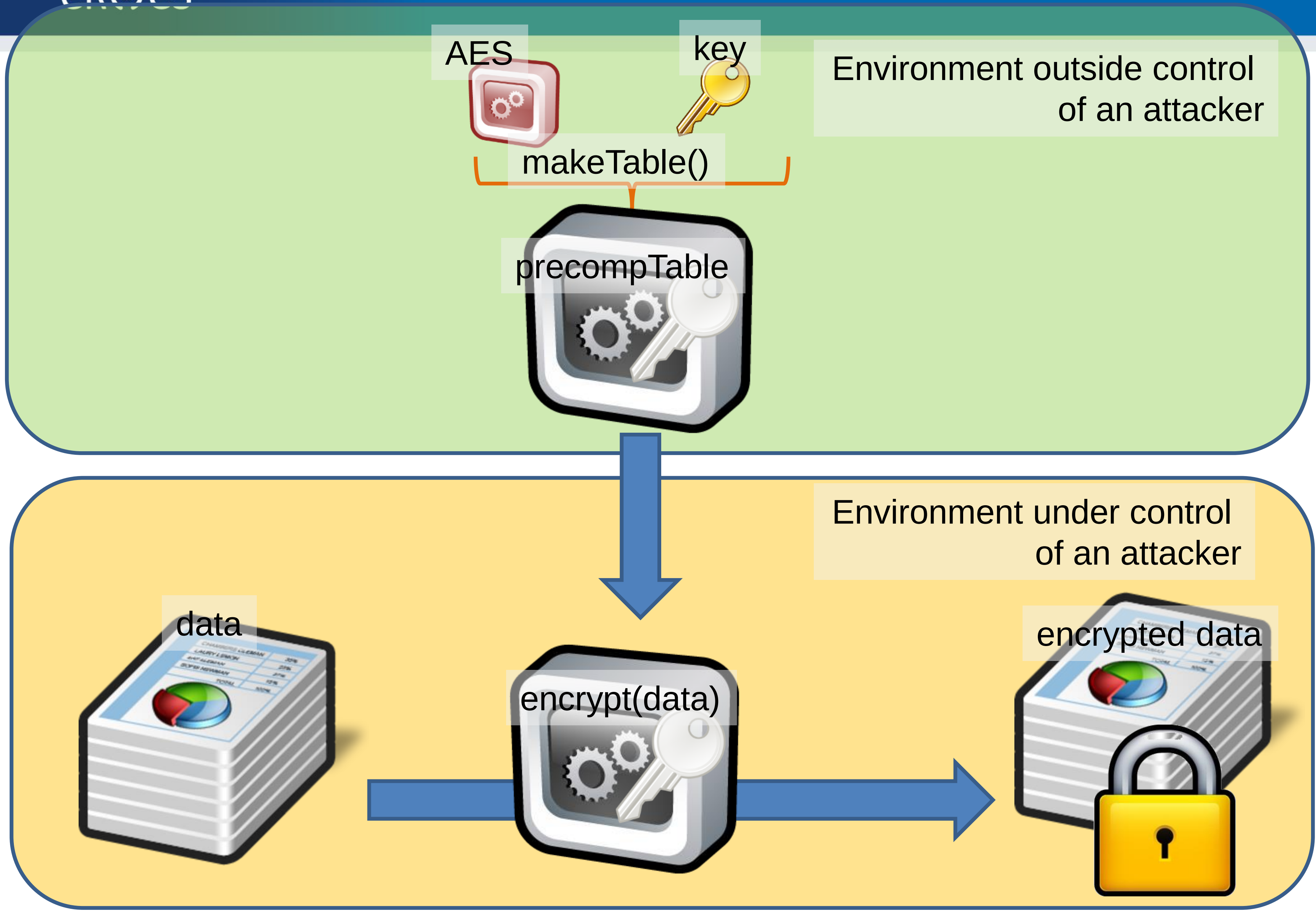
- Pre-compute table for all possible inputs
  - practical for one 16bits or two 8bits arguments table with up to 2^{16} rows (~64KB)
  - **`AddRoundKey: data ⊕ key`**
    - 8bit argument data, key fixed



- 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

- 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

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
- ...