Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

« *virus don't harm, ignorance does* »
herm1t

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Outline
  - What's a k-ary virus ???
  - Implementation
  - Conclusion

- What's a k-ary virus ?
  - Cohen's general model of computer viruses :
    - every code is made of a single program which contains the whole instructions devoted to its action

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- What's a k-ary virus ?
  - Cohen's general model of computer viruses :
    - every code is made of a single program which contains the whole instructions devoted to its action

Since every virus is supposed to be composed of a single code, antiviral detection itself considers only this model

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Scattered the viral information over differents files
  - make the viral detection far more complex

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Scattered the viral information over differents files
  - make the viral detection far more complex

The k constituting part looks like an innocent file and thus does not trigger any alert

- Definition (Éric Filiol) :
  - *A k-ary virus is a family of k files (some of them may be not executable) whose union constitues a computer virus and performs an offensive action that is equivalent to that of a true virus. Such a code is said sequential (serial mode) if the k constituent parts are acting strictly one after the another. It is said parallel if the k parts executes simultaneously (parallel mode).*

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Two modes :
  - Class I (sequential)
    - codes are executing one after another
  - Class II (parallel)
    - codes are executing at the same time

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- With 3 subclasses
  - A subclass (dependent sequential codes)
    - « Every part refers or contains a reference to the other ones. It is the weakest class in termof detection since successful detection of one part helps to detect the others. »
  - B subclass (independent sequential codes)
    - « No part is referring to another one. Detecting one part does not endanger the other ones. The detected part may be automatically replaced under a different form. »
  - C subclass (weakly dependent sequential codes)
    - « Dependency between codes is partial and directed only. »

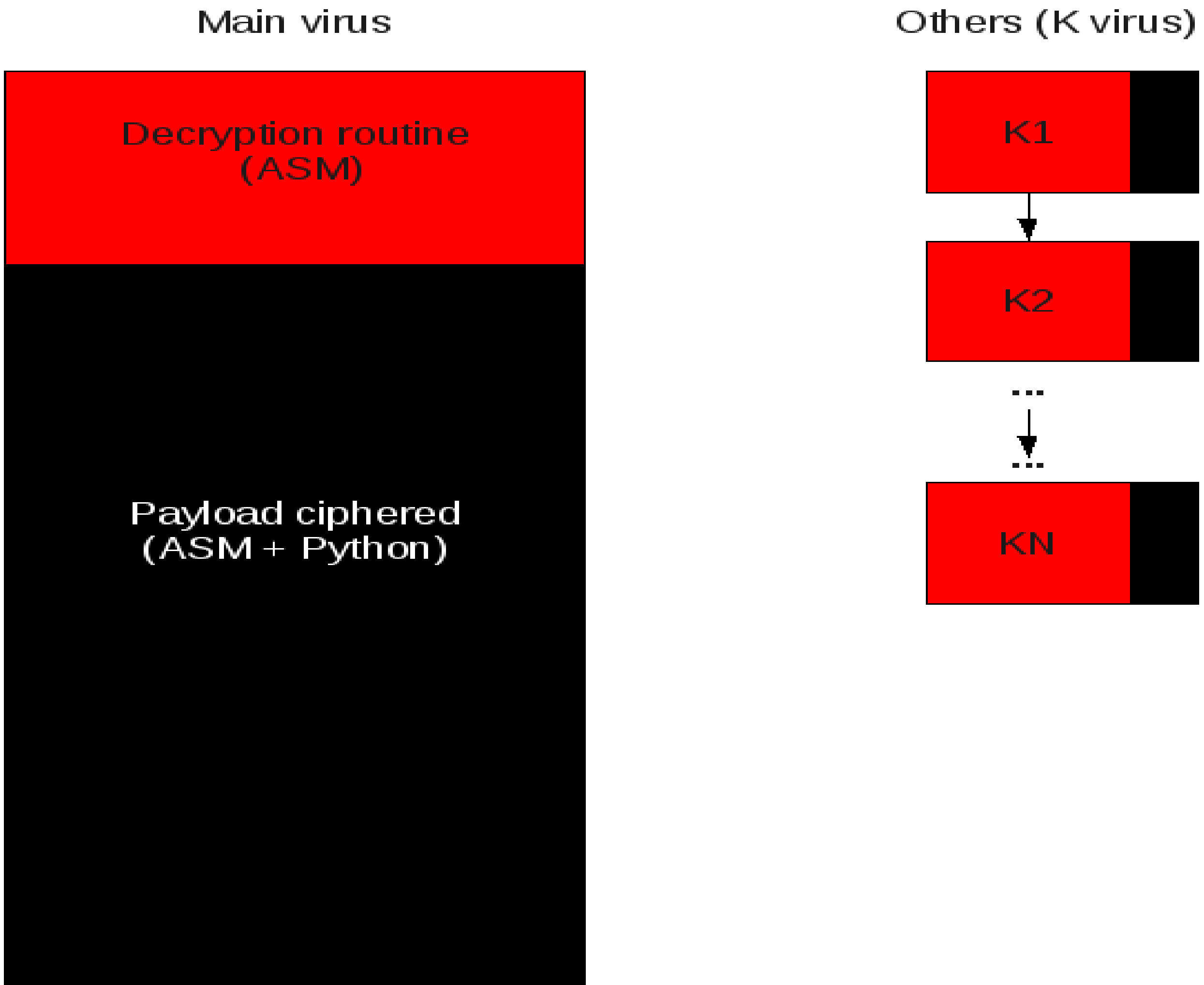
Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Class I (C and B subclasses)
  - The most interesting
- We must make N exploitations to execute the real virus
- Split our virus in differents parts :
  - the first contains the encrypted viral payload
  - the others contain the secret key
- Linux system

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)



Implementation of K-ary Viruses in Python

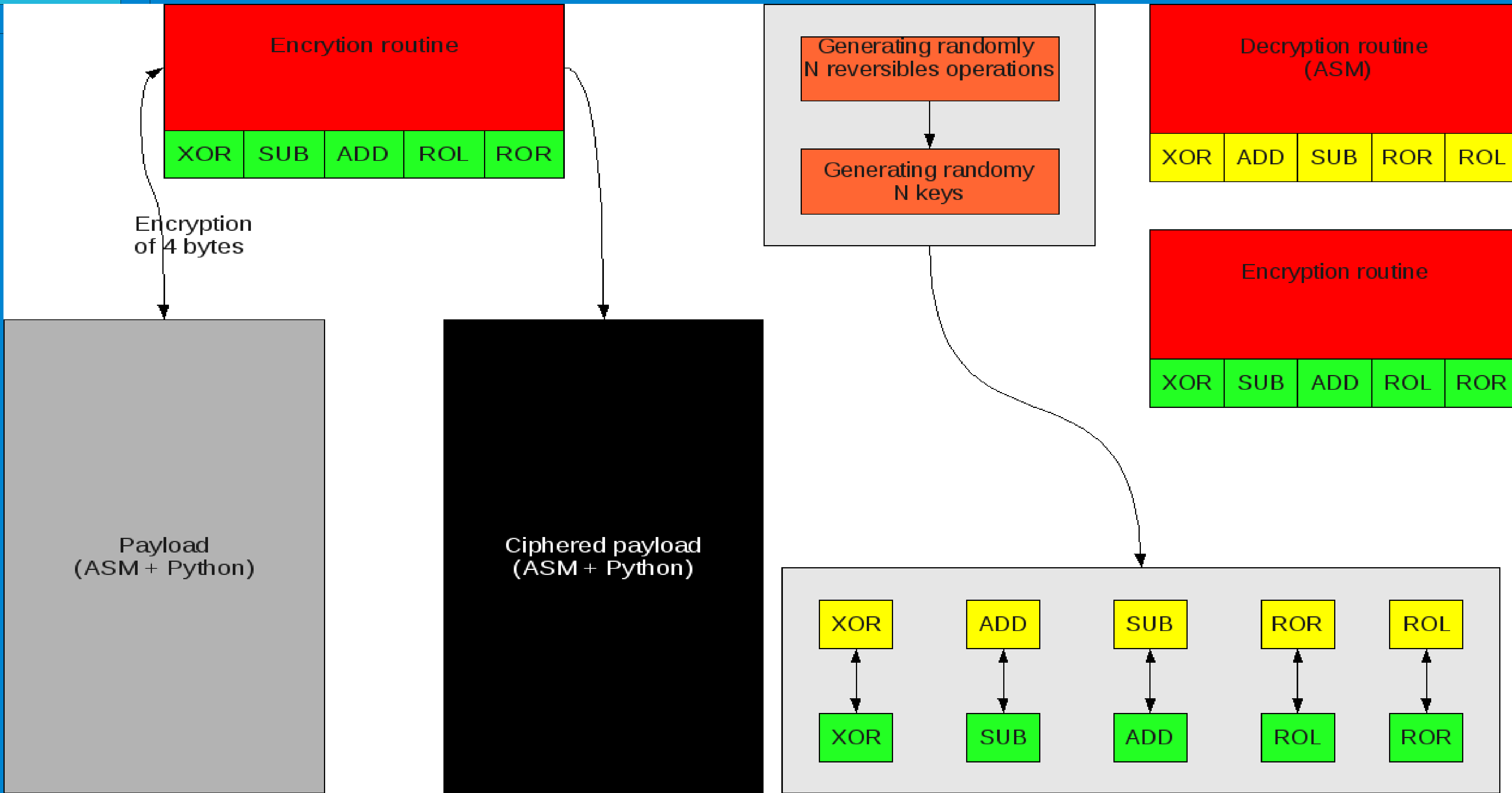
Desnos Anthony (ESIEA SI&S)

- Six steps :
  - Generation of N separate entities, a main entity containing the viral payload (with or without information about secret key), and secondary entities which reconstruct the private key to activate the viral payload,
  - The decryption routine,
  - Loading of the python script through several techniques,
  - Executing of python program, which decrypts with the help of others viruses the final payload,
  - Loading of the decrypted payload which is in memory,
  - The spread of the virus, in particular the generation of a new routine of encryption and decryption, therefore, with a return to stage 1.

- Polymorphic engine
  - CLET Team *(Polymorphic Shellcode Engine Using Spectrum Analysis, Phrack Magazine 61, 2004)*
    - *Generate a ciphered code which is different at each generation, with different keys*
      - *generate N reversible operations with N keys*
        - *examples (simple operations) :*
          - *XOR → XOR*
          - *ADD → SUB*
          - *ROL → ROR*

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)



Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Loading the script
  - Contains a simple script (in python)
    - In this script we have a buffer (or this script can download a buffer ...)
      - which decrypt the final payload
        - when the key is complete

→ How can I execute (stealth) my python script ??

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Loading the script
  - It's written in assembly language
    - We can use /dev/shm
      - tmpfs --> ramfs
        - It's a memory file system !

esiea

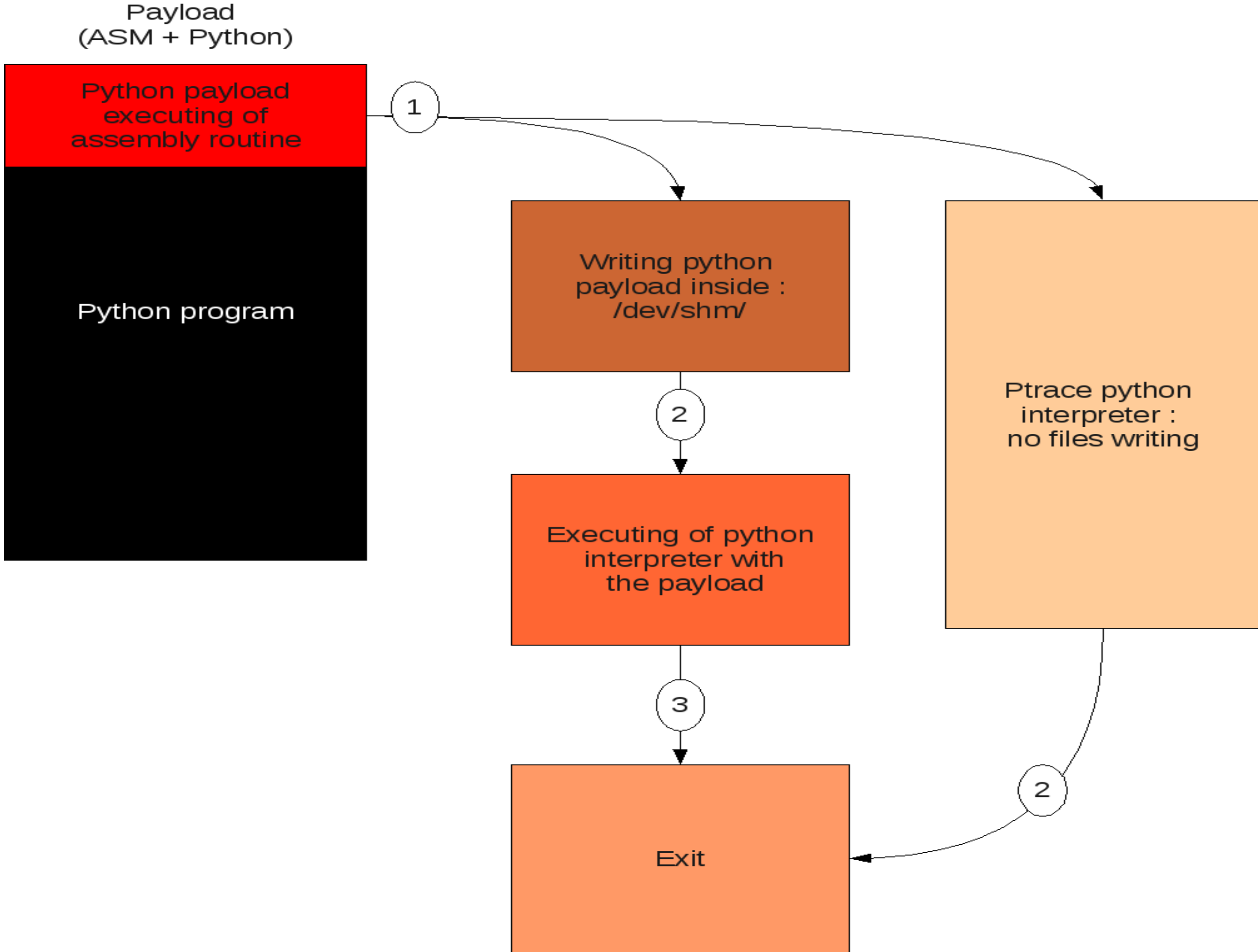
Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Loading the script
  - It's written in assembly language
    - We can use ptrace
      - Hijack open/read/close to load our own code which is in our memory !

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)



Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Remote loading of python code
  - We have download a remote python code which can be :
    - in memory in the same process,
    - in memory in another process,
    - on internet, for example on pastebin.com

→ How can I execute a remote python code ??

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Remote loading of python code
  - a simple python class LoadingRemoteModule
    - which gets the buffer, creates classes and calls functions
      - We can use python module :
        - « new » module : creation of run time internal objects
          - with « module » function

→ mod = new.module (name)

- Remote loading of python code
  - a simple python class LoadingRemoteModule
    - which gets the buffer, creates classes and calls functions
      - We can use python module :
        - « exec » module : which load a string (or an object of type file, or object code) in a context. This context should be the dictionary of our new module.

→ exec source in mod.__dict__

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Remote loading of python code
  - a simple python class LoadingRemoteModule
    - which gets the buffer, creates classes and calls functions
      - We can use python module :
        - Once the module is in the context, it must be load
        - « __import__ » function : returns the module

```
module = __import__( modulename )
```

Desnos Anthony (ESIEA SI&S)

- Remote loading of python code
  - a simple python class LoadingRemoteModule
    - which gets the buffer, creates classes and calls functions
      - « getattr » function : permits from the module to retrieve a class

```
class_ = getattr ( module , classname )
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Remote loading of python code
  - a simple python class LoadingRemoteModule
    - which gets the buffer, creates classes and calls functions
      - « inspect » module :
        - « getargspec » function : to know for a function (thus, the case of constructor) the number of argument, the names and default values.

arg = inspect.getargspec (class.__init__)[0]

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Remote loading of python code
  - a simple python class LoadingRemoteModule
    - which gets the buffer, creates classes and calls functions
      - Then the object is simply constructed with the class returned by getattr and the arguments are in parameters.

```
newinit = [ ]
arg.pop(0)
for i in arg :
        newinit.append( i )
newargs = izip ( newinit, args )
d = {}
for i in newargs :
        d[str ( i[0] )] = i[1]
obj = class(d)
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - several problems for a virus
    - to use a weak encryption,
    - embedded a tested library or its own optimized library (risk of a poor implementation),
    - to use a library on the system.

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - several problems for a virus
    - to use a weak encryption,
    - embedded a tested library or its own optimized library (risk of a poor implementation),
    - to use a library on the system.

We have made the choice to use a library on the system, and therefore take full advantage of a variable present in a vast majority of Linux machines.

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - several problems for a virus
    - to use a weak encryption,
    - embedded a tested library or its own optimized library (risk of a poor implementation),
    - to use a library on the system.

→ OPENSSL \o/

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Openssl in python ?
    - Not in the default python installation

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Openssl in python ?
    - Not in the default python installation

→ CTYPES \o/

- Cryptographic library
  - Ctypes
    - Load a dynamic library
      - call its functions

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Ctypes
    - Load libssl

```
OPENSSL_FILENAME = find_library ("ssl")
openssl = cdll.LoadLibrary(OPENSSL FILENAME)
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Ctypes
    - RSA : generate new pairs of key

```
openssl.RAND_load_file ("/dev/random" , 2048)
rsa = c_void_p (openssl.RSA_generate_k e y (bits, 0x10001 , None , None ))
rsa_size = openssl.RSA_size (rsa.value)
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Ctypes
    - RSA : Encrypt/Decrypt

```
o = create_string_buffer(rsa_size)
input = create_string_buffer(buffer[i:i+self.rsa_size - 11])
openssl.RSA_public_encrypt(len(input.raw) - 1, addressof(input), addressof(o), rsa.value, 1)
```

```
o = create_string_buffer(rsa_size)
input = create_string_buffer(buffer[i:i+self.rsa_size - 11])
openssl.RSA_public_decrypt(len(input.raw) - 1, addressof(input), addressof(o), rsa.value, 1)
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Ctypes
    - RSA : private key without encryption in the PEM format

```
rsa_private_key = ""
bio = c_void_p(self.openssl.BIO_new(openssl.BIO_s_mem()))
if openssl.PEM_write_bio_RSAPrivateKey(bio.value, rsa.value, None, None) == 1:
        temp = c_char_p()
        bufpriv_len = openssl.BIO_ctrl(bio.value, 3, 0, addressof(temp))
        tmp = temp.value
        rsa_private_key = tmp[0:bufpriv_len]
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Cryptographic library
  - Ctypes
    - AES

```
class AES_KEY(Structure):
    _fields_ = (
        ("rd_key", c_uint * 60),
        ("rounds", c_int),
        )

enc_key = AES_KEY()
dec_key = AES_KEY()

openssl.AES_set_encrypt_key(key, 16 * 8, addressof(enc_key))
openssl.AES_set_decrypt_key(key, 16 * 8, addressof(dec_key))
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

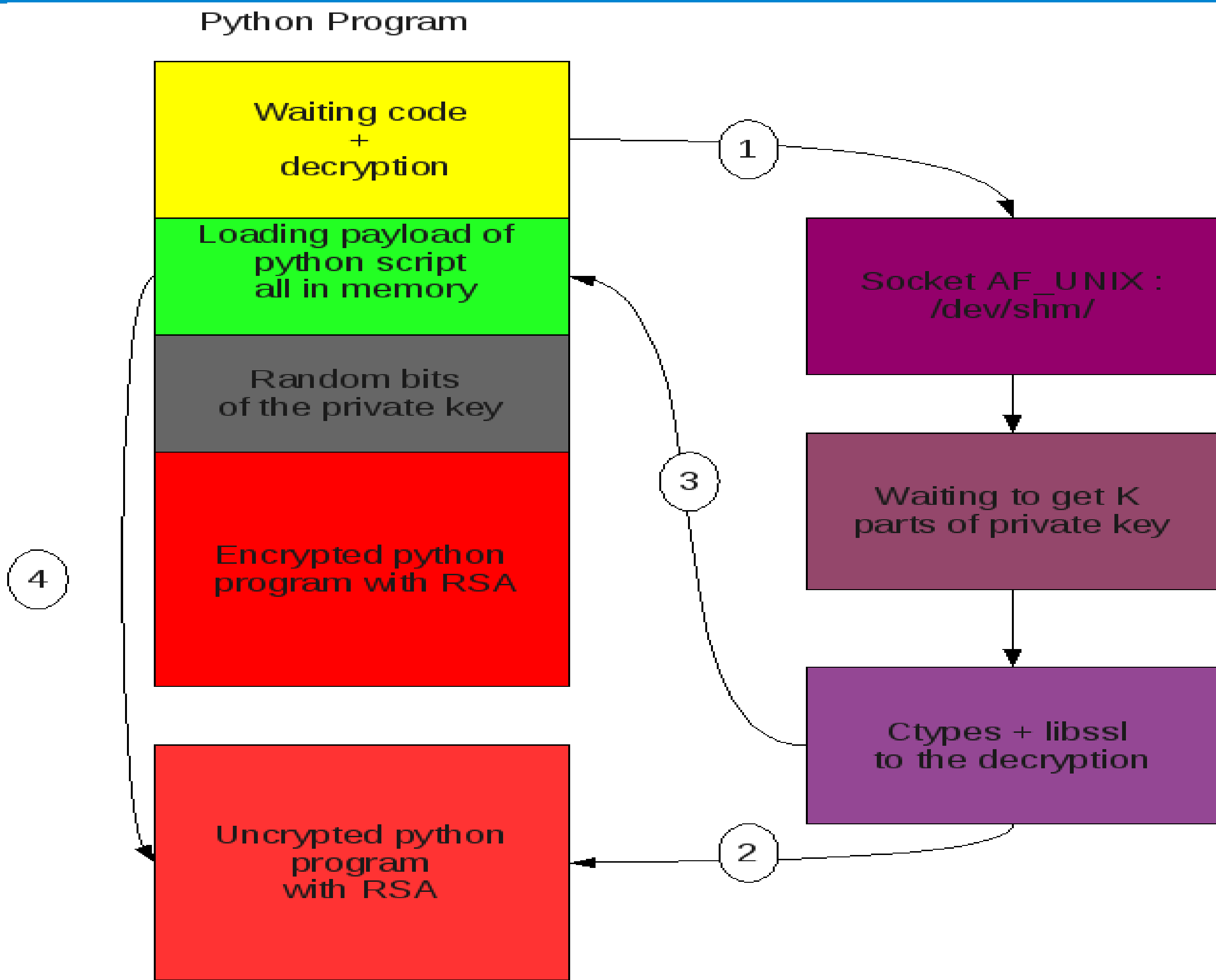
- Cryptographic library
  - Ctypes
    - AES

```
o = create_string_buffer(16)
openssl.AES_encrypt(buffer[i:i+16], addressof(o), addressof(enc_key))

o2 = create_string_buffer(16)
openssl.AES_decrypt(addressof(o), addressof(o2), addressof(dec_key))
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)



Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Our main problem is to protect our final payload
  - We have encrypted it, but it remains the problem of the storage of the key
    - If the key is contained in the same source code that the virus, then it is very easy for an analyst to find it

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Our main problem is to protect our final payload
  - We have encrypted it, but it remains the problem of the storage of the key
  - If the key is contained in the same source code that the virus, then it is very easy for an analyst to find it

► K -ary viruses can provide an elegant solution to this problem.

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, C subclass
  - weakly dependent sequential codes
  - split our key in equal parts
    - in some cases that could allow an analyst to have all parts of the key

Implementation of K-ary Viruses in Python

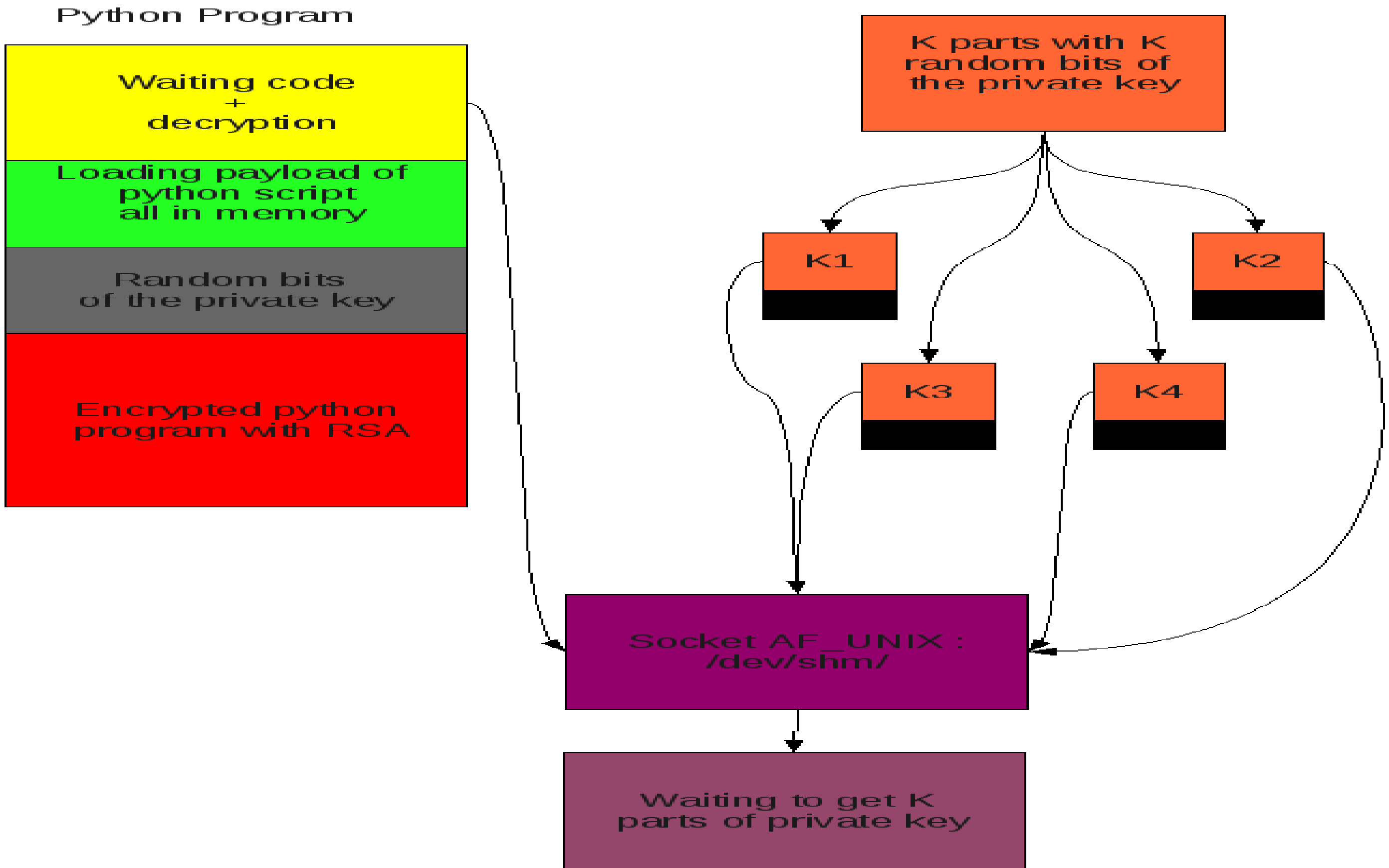
Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, C subclass
  - weakly dependent sequential codes
  - split our key in equal parts but also randomly
    - thus it is impossible for an analyst to retrieve the key without having all different codes

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

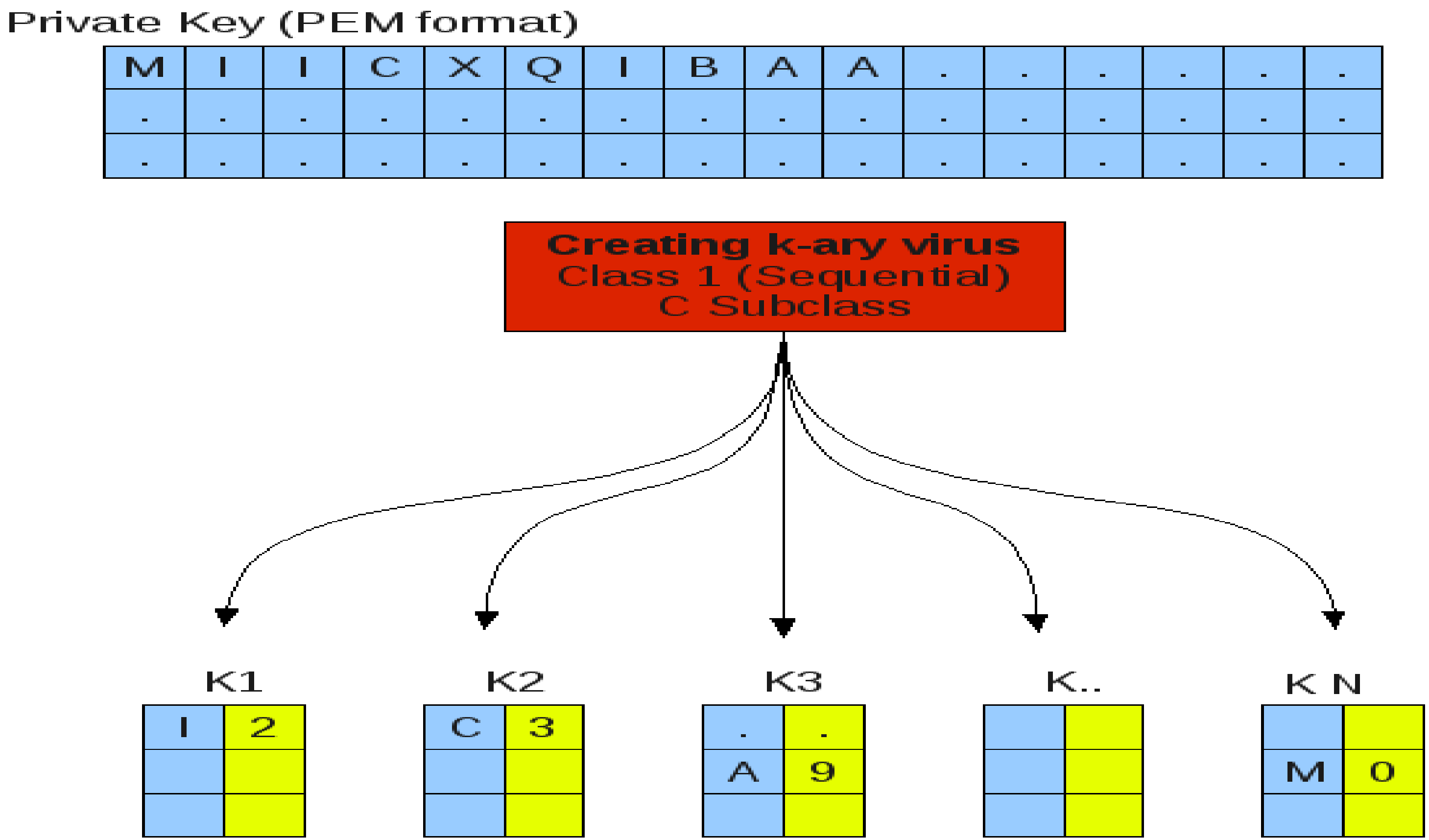
- K-ary virus in sequential mode, C subclass



Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, C subclass



Implementation of K-ary Viruses in Python

- K-ary virus in sequential mode, B subclass
  - independent sequential codes
  - the previous subclass has a big flaw, all codes must arrived in the target to start the final payload
    - packets drop
    - missed exploits

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - independent sequential codes
  - the previous subclass has a big flaw, all codes must arrived in the target to start the final payload
    - packets drop
    - missed exploits

it is possible that a code can't arrive and therefore that the spread doesn't continue !

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - not dependent and can regenerate themselves
    - so if there was a threshold on the different codes generated for the reconstruction of the key without that the totality reaches the destination, or the activation of the final charge after a given time
      - it would continue the spread

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - not dependent and can regenerate themselves
    - so if there was a threshold on the different codes generated for the reconstruction of the key without that the totality reaches the destination, or the activation of the final charge after a given time
      - it would continue the spread

→ secret share schemes

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - secret share schemes
    - the main goal is to divide a data D into n pieces D1 ....Dn in the following manner between different participants
      - knowledge of any k or more Di pieces makes D easily computable,
      - knowledge of any k - 1 or fewer Di pieces leaves D completely undetermined.

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Shamir's secret sharing
    - 2 points are sufficient to define a line,
    - 3 points are sufficient to define a parabola,
    - 4 points are sufficient to define a cubic curve,
    - ...

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Shamir's secret sharing
    - take k points to define a polynomial of degree k – 1
    - To build the polynomial, choose at random (k - 1) coefficients $a_1, \dots, a_{(k-1)}$, and let be a_0 the secret :

$$f(x) = a_0 + a_1 x + a_2 x2 + a_3 x3 + \dots + a_{(k-1)} x^{(k-1)}$$

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Shamir's secret sharing
    - Every participant (in our case, every virus) is given from a point X of this system, a pair (X, f (X)) (where each X must be different). When k participants are present, the secret can be found, otherwise it is impossible to recover it.

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Shamir's secret sharing
    - Our secret is our private key, a simple solution to handle our key is to transform it into PEM format, and convert it into a big integer
      - Another solution isn't to share the private key but the password which encrypt the key, this reduces the computing time and the data exchanges.
    - Python

```
def str2long(s):
    """Convert a string to a long integer."""
    if type(s) not in (types.StringType, types.UnicodeType):
        raise ValueError, 'the input must be a string'
    l = 0L
    for i in s:
        l <<= 8
        l |= ord(i)
```

esiea

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Neville-Aitken's algorithm
    - Once a virus arrived with its pair (X, f (X)), we must be able to find the secret (our a0). To do this we can use Neville-Aitken 's algorithm to find a coefficient, that allows to calculate any degree of the polynomial :

$$p_{(i,i)}(x) = y_i, 0 \le i \le n, \ p_{(i,j)}(x) = \frac{((x - x_j)\, p_{(i,j-1)}(x) + (x_i - x)\, p_{(i+1,j(x))})}{(x_i - x_j)}, 0 \le i < j \le n.$$

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Neville-Aitken's algorithm
    - In this case, we want the coefficient of degree 0 (which is the key or the password):

$$p_{(i,i)}(x)=y_i,\ 0\leq i\leq n,\ p_{(i,j)}(x)=\frac{((0-x_j)\,p_{(i,j-1)}(x)+(x_i-0)\,p_{(i+1,j(x))})}{(x_i-x_j)},\ 0\leq i<j\leq n.$$

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass
  - Neville-Aitken's algorithm
    - This algorithm has a space and time complexity both in O(n^2), and can be implemented easily in python

```
def interpolate(x0, y0, x1, y1, x) :
    return (y0*(x-x1) - y1*(x-x0)) / (x0 - x1);

def solveSystem(xs, ys):
    for i in range(1, len(xs)) :
        for k in range(0, len(xs) - i) :
            ys[k] = interpolate(xs[k], ys[k], xs[k+i], ys[k+1], 0)

    return ys[0]
```

esiea

Implementation of K-ary Viruses in Python

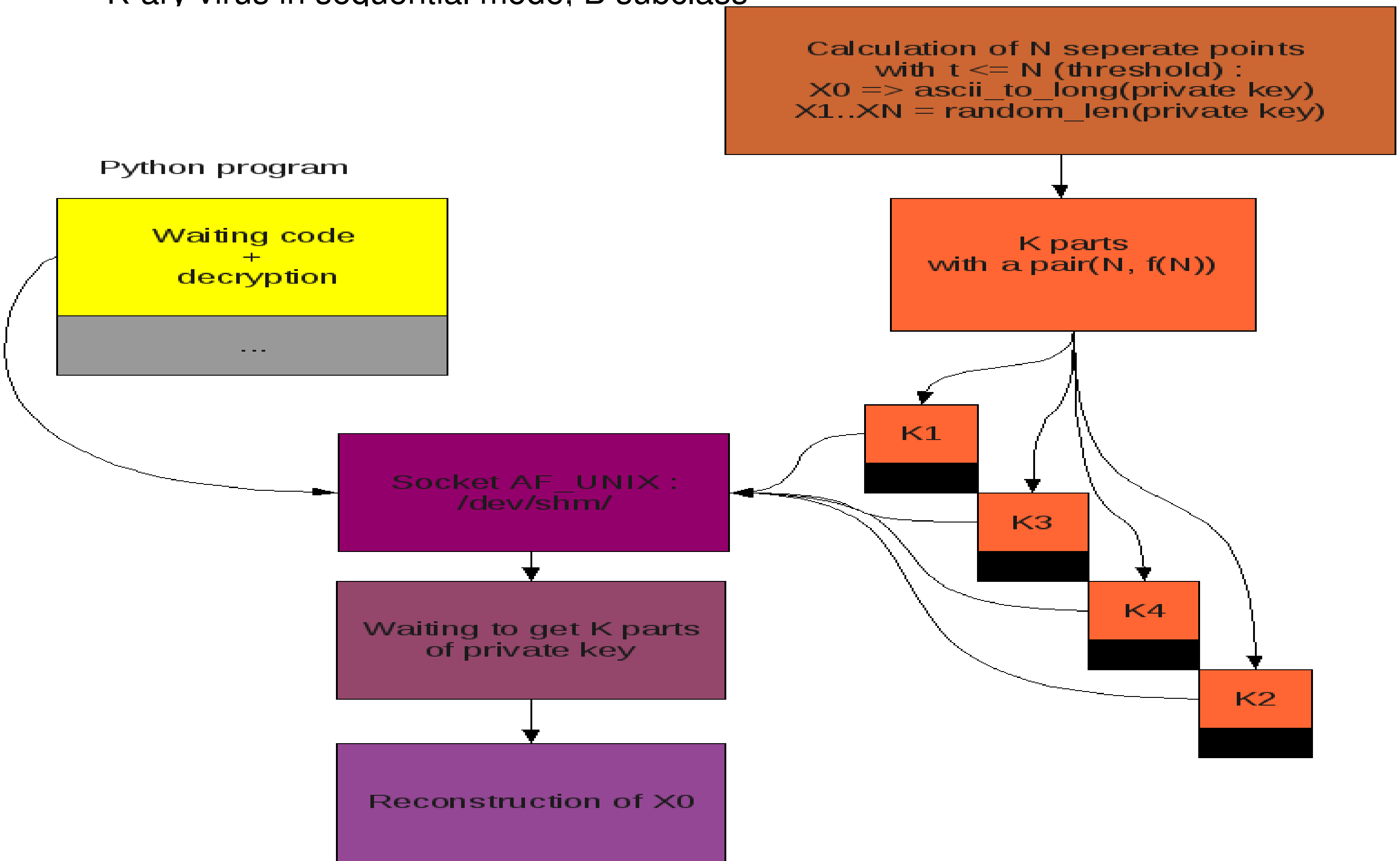
Desnos Anthony (ESIEA SI&S)

```
./shamir.py toto
SECRET toto => TO LONG 1953461359
HASH SECRET
31f7a65e315586ac198bd798b6629ce4903d0899476d5741a9f32e2e521b6a66
f(x) = 1953461359  + 1082694448 x^1 + 100363181 x^2
POINT[1] = 3136518988
POINT[2] = 4520302979
POINT[3] = 6104813332
POINT[4] = 7890050047
POINT[5] = 9876013124
POINT[6] = 12062702563
Running Neville's algorithm :  Found x[0]
SECRET = toto
HASH = 31f7a65e315586ac198bd798b6629ce4903d0899476d5741a9f32e2e521b6a66
```

Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

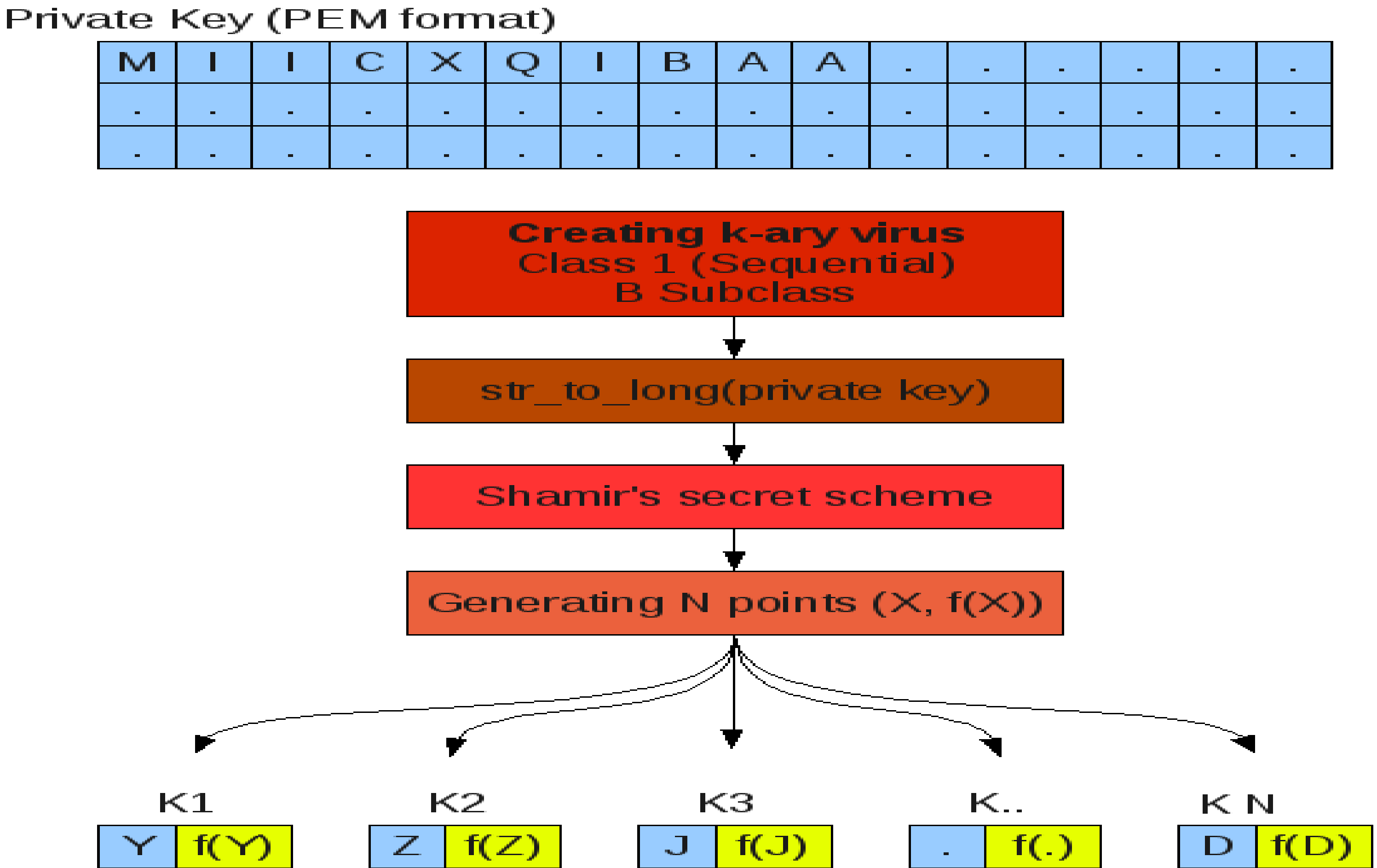
- K-ary virus in sequential mode, B subclass



Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- K-ary virus in sequential mode, B subclass



Implementation of K-ary Viruses in Python

Desnos Anthony (ESIEA SI&S)

- Conclusion
  - K-ary viruses provide an interesting solution to share the key in a virus
  - K-ary viruses are a profound change in the way of analysis from the point of view of anti-virus

Many thanks for your attention!
Have you any question... ?

Happy Hacking !

Thanks to Hack.lu