SYNACKTIV
DIGITAL SECURITY

Code Obfuscation 10**2+(2*a+3)%2

When 2018/11/29
Where JSecIn
Who Gaetan Ferry
Why For fun!

me@JSecIN:/ $ whoami

- **Gaetan Ferry**
- **@mabo^W Not on twitter**
- **Security expert @Synacktiv :**
  - Offensive security company : pentest, red team, reverse/exploit…
- **Pentest team peon:**
  - 1 me / 17 pentester / 41 ninjas
  - Breaking things since 2012
  - Web, internal, external, IOT, indus, cloud

WE WANT YOU FOR OUR NINJA ARMY !

INTRODUCTION

Why this presentation?

- **Why not?**
- **Obfuscation is an undervalued domain**
  - Usefulness often discussed
  - Defined as security by obscurity

- **Therefore abandoned**
- **Therefore unknown**

→ We want to redeem obfuscation

What is in this presentation?

- **Objectives of a proper obfuscation**
- **Details of classic obfuscation patterns**
- **Implementation with / for Python**
- **Examples and …**

…demos (pray demo gods)

WHAT IS OBFUSCATION ?

A bit of theory

Let P be the set of all programs and T a set of transformations such as:

$$T_i : P \rightarrow P$$

T_i is an obfuscation transformation if and only if:

- $out(T_i(P_k)) == out(P_k)$
- analysis of $T_i(P_k)$ is harder than analysis of P_k

T_i is considered efficient if the knowledge of $T_i(P_k)$ is equivalent to having a black-box oracle of $P_{k.}$

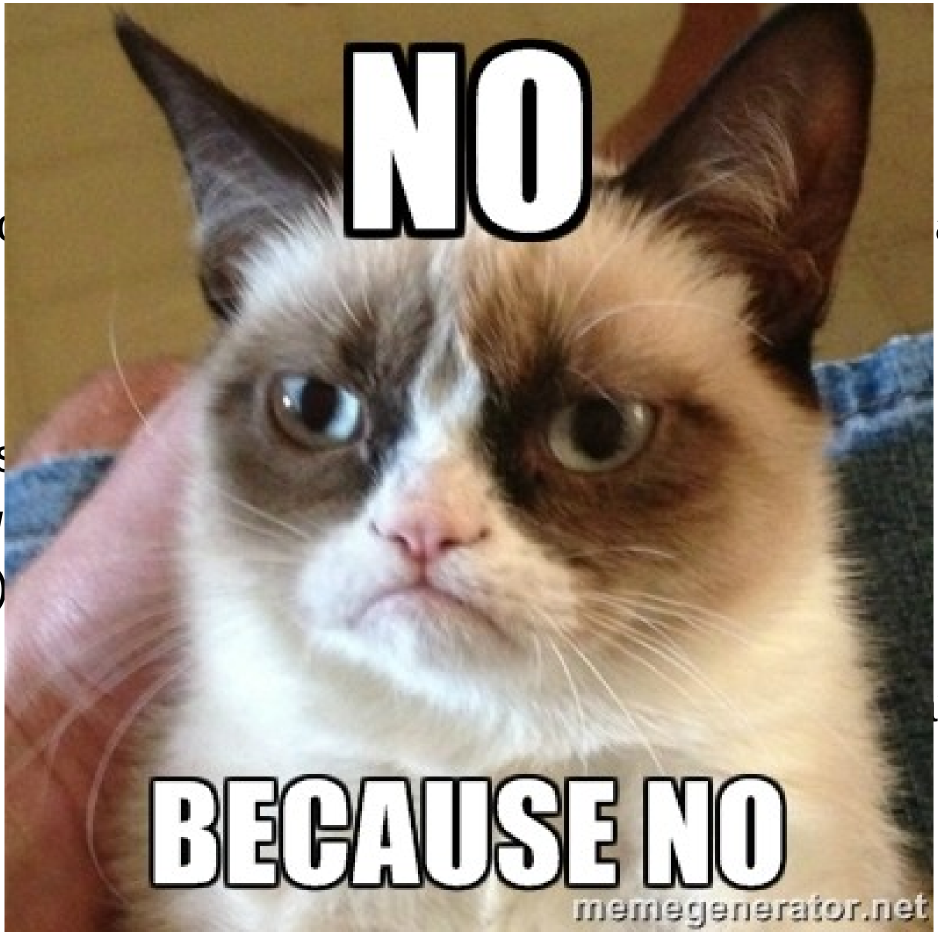
A bit of theory

Let P be the set of all pr as:

$T_i : P \to P$

T_i is an obfuscation trans
- $out(T_i(P_k))$ == ou
- analysis of $T_i(P_k)$

T_i is considered efficient ving a black-box oracle of $P_{k.}$



This is not what we are doing

- **Obfuscation does not stand well theory**
- **Theoretical results are demoralizing**
  - In general cases obfuscation is impossible
  - Some exceptions: point functions

- **Let's go with a more pragmatic approach**

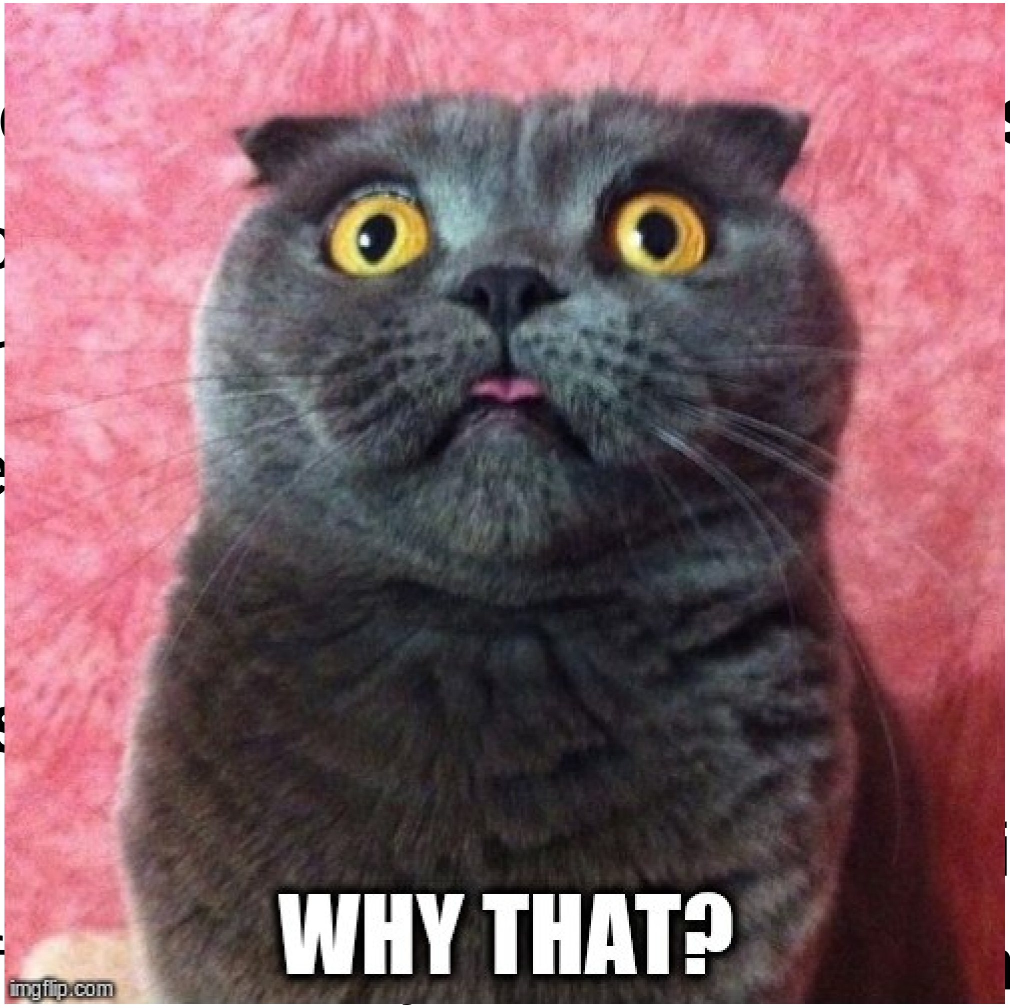
A more pragmatic approach

- **Process of complicating programs**
  - Take a beautiful well written program
  - Transform it in some way
  - Retrieve an obscure ugly program

- **Two rules to follow**
  - Resulting program is semantically equivalent
  - More difficult to analyze and understand

A more pragmatic approach

- **Process** ... **s**
  - Take a b
  - Transfor
  - Retrieve



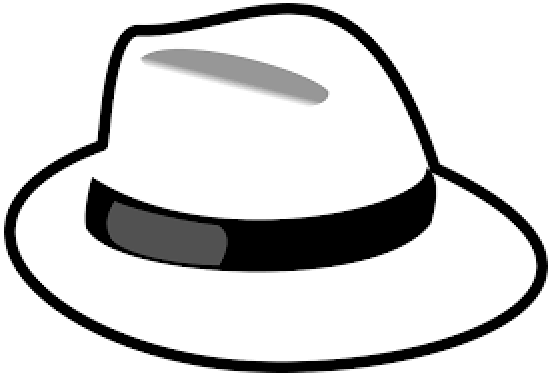
- **Two rules**
  - Resultin ... ivalent
  - More dif ... nd

Why do you want obfuscation?

- Useful for good and bad guys

Why do you want obfuscation?

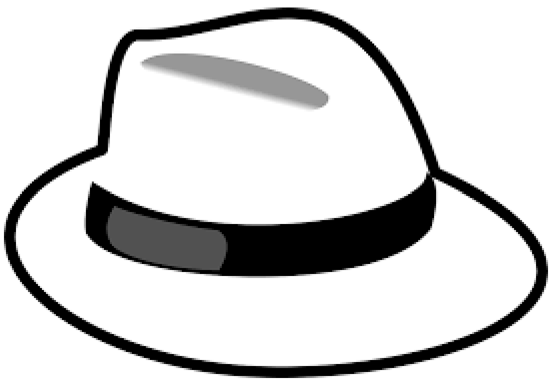
- Useful for good and bad guys



Protect industrial secrets
Discourage hackers who open the thing

Why do you want obfuscation?

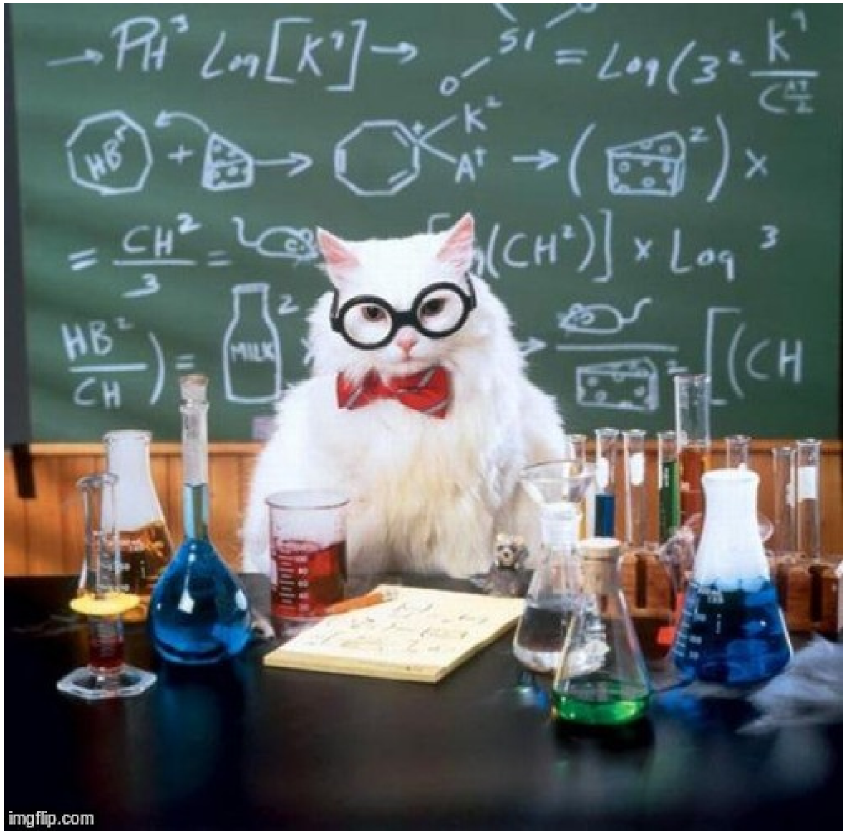
- Useful for good and bad guys



Protect industrial secret
Discourage hackers who open the thing

Bypass sandbox / antivirus detection
Prevent reverse engineering by the good guys

LET'S OBFUSCATE THINGS



How to complicate a program?

- Remove as much information as possible
- Three main directions:
  - Abstractions
  - Data
  - Control flow

- We need to obfuscate each kind

How to complicate a program?

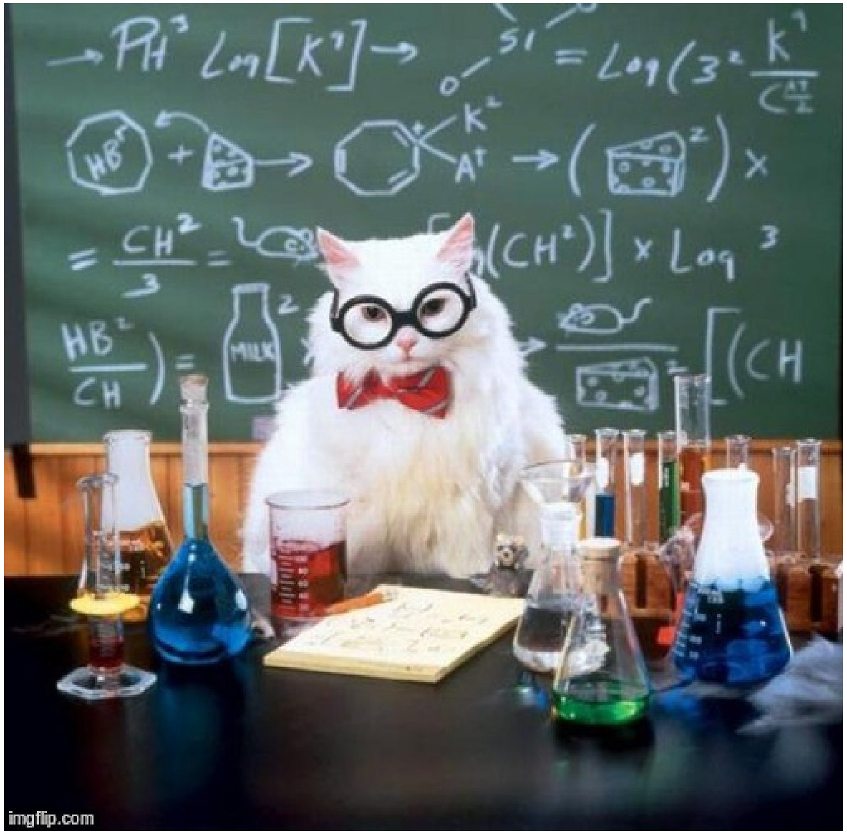
```
import struct
import binascii
import math

lrot = lambda x, n: (x << n) | (x >> (32 - n))

class MD5():

    A, B, C, D = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)

    # r specifies the per-round shift amounts
    r = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
         5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
         4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
         6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

    # Use binary integer part of the sines of integers (Radians) as constants
    k = [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in range(64)]

    def __init__(self, message):
        length = struct.pack('<Q', len(message) * 8)
        while len(message) > 64:
            self._handle(message[:64])
            message = message[64:]
        message += '\x80'
        message += '\x00' * ((56 - len(message) % 64) % 64)
        message += length
        while len(message):
            self._handle(message[:64])
            message = message[64:]
```

Abstractions

Data

Control flow

LET'S OBFUSCATE ABSTRACTIONS



Program abstractions

- Abstractions help understand programs

  → Imagine a program without proper function names or convoluted class hierarchy !
- Giveaway much of the program semantic
  - Division in semantic blocks
  - Role of the blocks
- Sensitive abstractions:
  - Variables
  - Functions
  - Classes

Names obfuscation

- First step of a successful obfuscation
  - Remove meaningful names from the code
  - Replace with random or unrelated ones

- This information is unrecoverable!   \o/
- EZ as 123:
  - Search for all declarations functions, variables, class
  - Replace at each usage location

Names obfuscation

Definition
Usage

```
def power(number, exponent) {
	count = number
	while (exponent > 1) {
		count = count * number
		exponent = exponent - 1
	}
	return count
}

power(2, 10)
```

```
def toast(number, exponent) {
	count = number
	while (exponent > 1) {
		count = count * number
		exponent = exponent - 1
	}
	return count
}

toast(2, 10)
```

```
def toast(bread, butter) {
	salad = bread
	while (butter > 1) {
		salad = salad * bread
		butter = butter - 1
	}
	return salad
}

toast(2, 10)
```

Going further

- Does not seem sufficient
  - Still leaking information
  - Program partitioning unchanged
- We should try to break things
- Ideas:
  - Function inlining
  - Merging / Splitting
- Warning: Beware of introspection calls!

Function merging

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}

def mult(a, b) {
    count = 0
    while (b > 0){
        count += a
        b -= 1
    }
    return count
}
```

```
A = 2
B = 3

C = add(A,B)
D = mult(C,A)
```

Function merging

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}

def mult(a, b) {
    count = 0
    while (b > 0){
        count += a
        b -= 1
    }
    return count
}
```

```
def toast(a, b, c, d, e)
{
    if (e) {
        count = a
        while (b > 0){
            count += 1
            b -= 1
        }
        return count
    } else {
        count = 0
        while (d > 0){
            count += c
            d -= 1
        }
        return count
    }
}
```

```
A = 2
B = 3

C = add(A,B)
D = mult(C,A)
```

```
A = 2
B = 3

C = toast(A,B,B,A,true)
D = toast(A,C,C,A,false)
```

Function merging - smarter

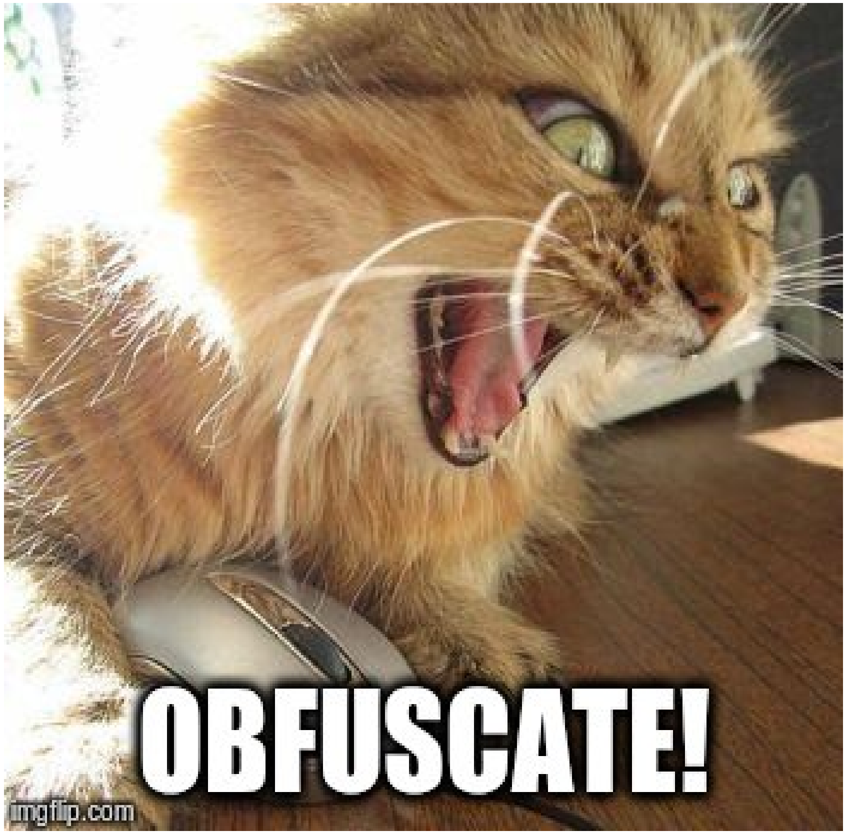
```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}

def mult(a, b) {
    count = 0
    while (b > 0){
        count += a
        b -= 1
    }
    return count
}
```

```
def toast(a, b, e) {
    if (e) {
        count = a
        add = 1
    } else {
        count = 0
        add = a
    }
    while (b > 0){
        count += add
        b -= 1
    }
    return count
}
```

```
A = 2
B = 3

C = add(A,B)
D = mult(C,A)
```

```
A = 2
B = 3

C = toast(A,B,true)
D = toast(C,A,false)
```

(disappointing) DEMO



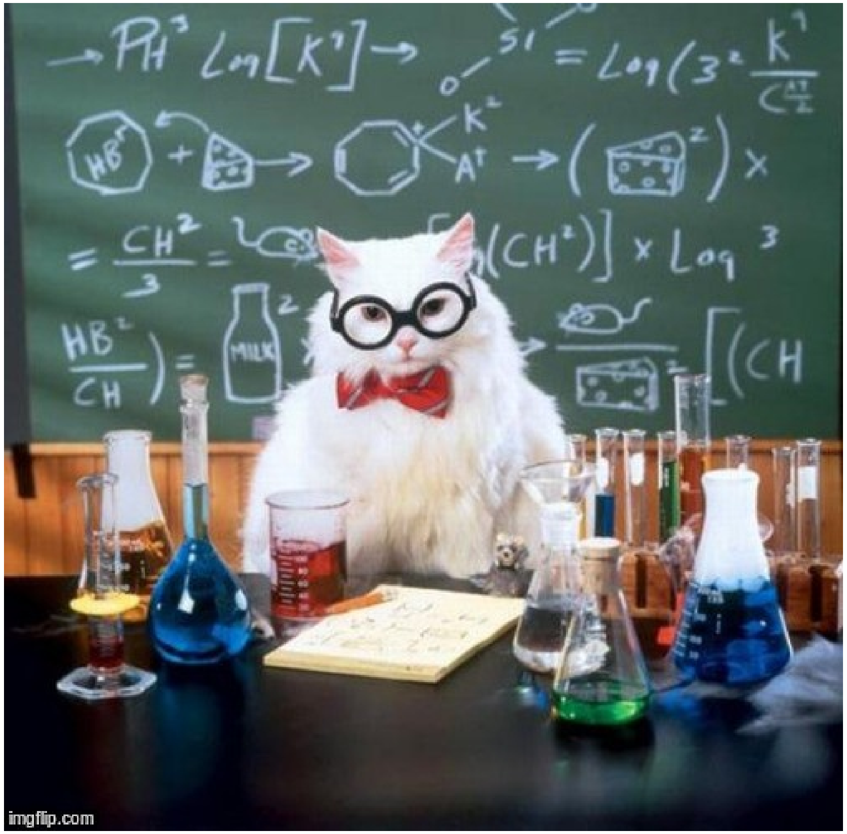
```python
import struct
import binascii
import math

lrot = lambda x, n: (x << n) | (x >> (32 - n))


class MD5():

    A, B, C, D = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)

    # r specifies the per-round shift amounts
    r = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
         5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
         4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
         6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

    # Use binary integer part of the sines of integers (Radians) as co
    k = [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in ra

    def __init__(self, message):
        length = struct.pack('<Q', len(message) * 8)
        while len(message) > 64:
            self._handle(message[:64])
            message = message[64:]
        message += '\x80'
        message += '\x00' * ((56 - len(message) % 64) % 64)
        message += length
        while len(message):
            self._handle(message[:64])
            message = message[64:]

    def _handle(self, chunk):
        w = list(struct.unpack('<' + 'I' * 16, chunk))

        a, b, c, d = self.A, self.B, self.C, self.D

        for i in range(64):
            if i < 16:
                f = (b & c) | ((~b) & d)
                g = i
            elif i < 32:
                f = (d & b) | ((~d) & c)
                g = (5 * i + 1) % 16
            elif i < 48:
                f = b ^ c ^ d
                g = (3 * i + 5) % 16
            else:
                f = c ^ (b | (~d))
                g = (7 * i) % 16
```

Before

After

```python
import struct
import binascii
import math
_736c50e6a0 = lambda _7d6e525234, n: _7d6e525234 << n | _7d6e525234 >> 32 - n


class _b1034ae5da:
    _79ddf344d7, _142985c1bd, _6bc7e0fdb2, _345efce705 = (1732584193,
        4023233417, 2562383102, 271733878)
    _75429d8422 = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17,
        22, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 4, 11,
        16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 6, 10, 15, 21,
        6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]
    _72c185c371 = [int(math.floor(abs(math.sin(_d2027f99f8 + 1)) * 2 ** 32)
        ) for _d2027f99f8 in range(64)]

    def __init__(self, _06aaffecbe):
        _ce75a64e4a = struct.pack('<Q', len(_06aaffecbe) * 8)
        while len(_06aaffecbe) > 64:
            self._c0bf0e997d(_06aaffecbe[:64])
            _06aaffecbe = _06aaffecbe[64:]
        _06aaffecbe += '\x80'
        _06aaffecbe += '\x00' * ((56 - len(_06aaffecbe) % 64) % 64)
        _06aaffecbe += _ce75a64e4a
        while len(_06aaffecbe):
            self._c0bf0e997d(_06aaffecbe[:64])
            _06aaffecbe = _06aaffecbe[64:]

    def _c0bf0e997d(self, _822cdd0988):
        _0ddb843858 = list(struct.unpack('<' + 'I' * 16, _822cdd0988))
        _f4d23af499, _b40a56f069, _52262f88fe, _3f6c23c57f = (self.
            _79ddf344d7, self._142985c1bd, self._6bc7e0fdb2, self._345efce705)
        for _d2027f99f8 in range(64):
            if _d2027f99f8 < 16:
                _4116aa63c6 = (_b40a56f069 & _52262f88fe | ~_b40a56f069 &
                    _3f6c23c57f)
                _2bfeb87934 = _d2027f99f8
            elif _d2027f99f8 < 32:
                _4116aa63c6 = (_3f6c23c57f & _b40a56f069 | ~_3f6c23c57f &
                    _52262f88fe)
                _2bfeb87934 = (5 * _d2027f99f8 + 1) % 16
            elif _d2027f99f8 < 48:
                _4116aa63c6 = _b40a56f069 ^ _52262f88fe ^ _3f6c23c57f
                _2bfeb87934 = (3 * _d2027f99f8 + 5) % 16
            else:
                _4116aa63c6 = _52262f88fe ^ (_b40a56f069 | ~_3f6c23c57f)
                _2bfeb87934 = 7 * _d2027f99f8 % 16
```

LET'S OBFUSCATE DATA



Program data

- All programs contain data:
  - Numbers, strings, arrays, etc
- Often (always) discloses important information:
  - Status / debug messages
  - Important constants (MD5, AES S-Box, etc)
- We want to hide those nasty values !
  - In our example: integers

Program data

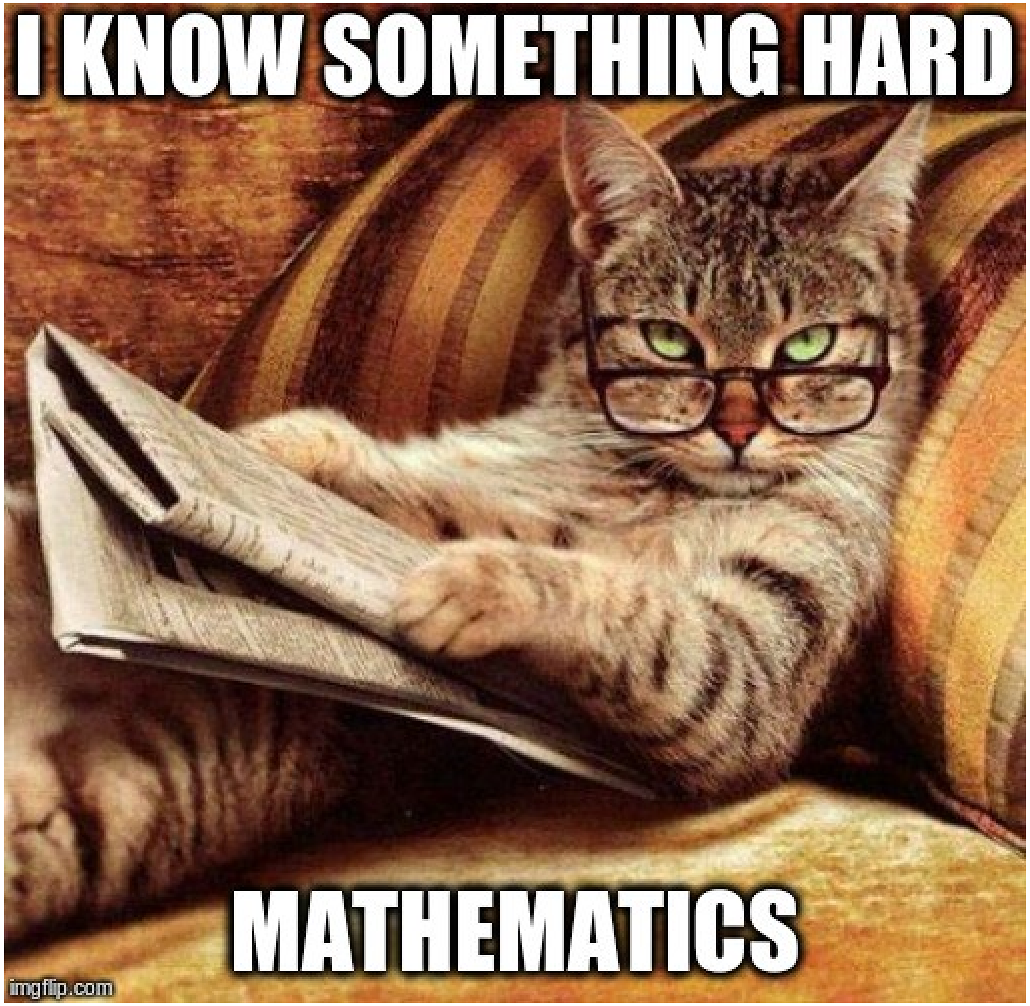
- All programs contain data:
  - Numbers, strings, arrays, etc
- Often (always) discloses important information:
  - Status / debug messages
  - Important constants (MD5, AES S-Box, etc)
- We want to hide those nasty values !
  - In our example: integers

USE OPAQUE PREDICATES !

Opaque predicates and values

- One of the core concepts of obfuscation
- We want to build expressions for which:
  - Value is known at obfuscation time
  - At run time value is hard to determine
- When value is a boolean it's a predicate

Opaque predicates – naive idea



Opaque predicates – naive idea

- Open a mathematics course book
- Ctrl + F “demonstrate that“
- Profit

- Examples:
  - $(n^2 + n) \% 2 = 0$
  - If n is odd : $n^2 \% 8 = 1$
  - $(3^{(2n + 2)} + 1) \% 8 = 2$

Opaque predicates – naive idea

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

Opaque predicates – naive idea

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

$(n^2 + n) \% 2 = 0$

$n^2 \% 8 = 1$

Opaque predicates – naive idea

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

$$(n^2 + n) \% 2 = 0$$

$$n^2 \% 8 = 1$$

```
def add(a, b){
    count = a
    n = rand()
    while (b > (n² + n) % 2){
        count += n² % 8
        b -= n² % 8
    }
    return count
}
```

Opaque predicates – naive idea

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

$(n^2 + n) \% 2 = 0$

$n^2 \% 8 = 1$

```
def add(a, b){
    count = a              ???
    n = rand() * 2 + 1
    while (b > (n² + n) % 2){
        count += n² % 8
        b -= n² % 8
    }
    return count
}
```

Opaque predicates – naive idea

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

$(n^2 + n) \% 2 = 0$

$n^2 \% 8 = 1$

```
def add(a, b){
    count = a                 ???
    n = rand() * 2 + 1
    while (b > (n² + n) % 2){
        count += n² % 8
        b -= n² % 8
    }
    return count
}
```

```
def add(a, b){
    count = a
    while (b > (a² + a) % 2){
        n = count*2+1
        count += (n)² % 8
        b -= n² % 8
    }
    return count
}
```

Opaque predicates – naive idea

- Problem:

  Smart cat is smart! Smart cat knows mathematics!

- Attacking those predicates is easy:
  - Build a collection of mathematics results
  - Pattern match known relations
  - Replace
- We can do better

Array aliasing

- Let's build our own mathematical results
  - Create an array
  - Decide properties
  - Initialize the array respecting the properties
- Then use the properties like previously

Array aliasing

- Example:

```
A = [17, 53, 3, 5, 1, 8, 25, 33, 4, 1]
```

== 3 mod 5

== 1 mod 4

- 3 == A[1]%A[4]   A[2] == A[5]%A[4]
- 1 == A[5]%A[8]   A[9] == A[0]%A[8]

Array aliasing

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

$0 == A[5]\%A[3] - A[1]\%A[3]$

$1 == A[4]\%A[8]$

$1 == A[0]\%A[8]$

Array aliasing

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
```

```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
```

0 == A[5]%A[3] - A[1]%A[3]

1 == A[4]%A[8]

1 == A[0]%A[8]

Array aliasing

- Still insufficient:
  - Global array is static
  - Attacker can globally replace values

- We need to bring indecision in!
  - Idea: change the array during the program’s execution
  - Hard! (e.g. How to know the state in function bodies?)

Array aliasing

- Still insufficient:
  - Global array is static
  - Attacker can globally replace values

- We need to bring indecision in
  - Idea: change the array during the program’s execution
  - Hard! (e.g. How to know the state in function bodies?)
  - But not if you keep the properties

Array aliasing

- Example:

== 3 mod 5   == 1 mod 4

```
A = [17,53,3,5,1,8,25,33,4,1]
```

+ A[3]   + A[8]

- 3 == A[1]%A[4]   A[2] == A[5]%A[4]
- 1 == A[5]%A[8]   A[9] == A[0]%A[8]

Array aliasing

- Example:

== 3 mod 5   == 1 mod 4

A = [17, 53, 3, 5, 1, 8, 25, 33, 4, 1]

+ A[3]   + A[8]

A = [25, 58, 3, 5, 5, 33, 17, 8, 4, 1]

- 3 == A[1]%A[4]   A[2] == A[5]%A[4]
- 1 == A[5]%A[8]   A[9] == A[0]%A[8]

OK

Array aliasing

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
add(2,3)
```

0 == A[5]%A[3] - A[1]%A[3]

1 == A[4]%A[8]

1 == A[0]%A[8]

```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
add(2,3)
```

Array aliasing

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
add(2,3)
```

0 == A[5]%A[3] - A[1]%A[3]
1 == A[4]%A[8]
1 == A[0]%A[8]

```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
add(2,3)
```

```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    A[0]=A[4]
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        A[4] += A[0]%A[8]
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
A[5]=(A[1]+A[7])%A[4]+A[7]
add(2,3)
```

Array aliasing

```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    A[0]=A[4]
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        A[4] += A[0]%A[8]
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
A[5]=(A[1]+A[7])%A[3]+A[7]
c = add(2,3)
A[5]=(A[1]+A[7])%A[3]+A[7]
D = add(2,3)
C == D ???
```

- Results
  - Data now changes at each run
  - Function add change
  - Guessing the value of add(2,3) now requires analyzing more than just the add function
  - Result might change at each call

Array aliasing

```
A = [17,53,3,5,1,8,25,
def add(a, b){
    A[0]=A[4]
    count = a
    while (b > A[5]%A[
        A[4] += A[0]%A
        count += A[4]%A
        b -= A[0]%A[8]
    return count
}
A[5]=(A[1]+A[7])%A[3]+
c = add(2,3)
A[5]=(A[1]+A[7])%A[3]+
D = add(2,3)
C == D ???
```

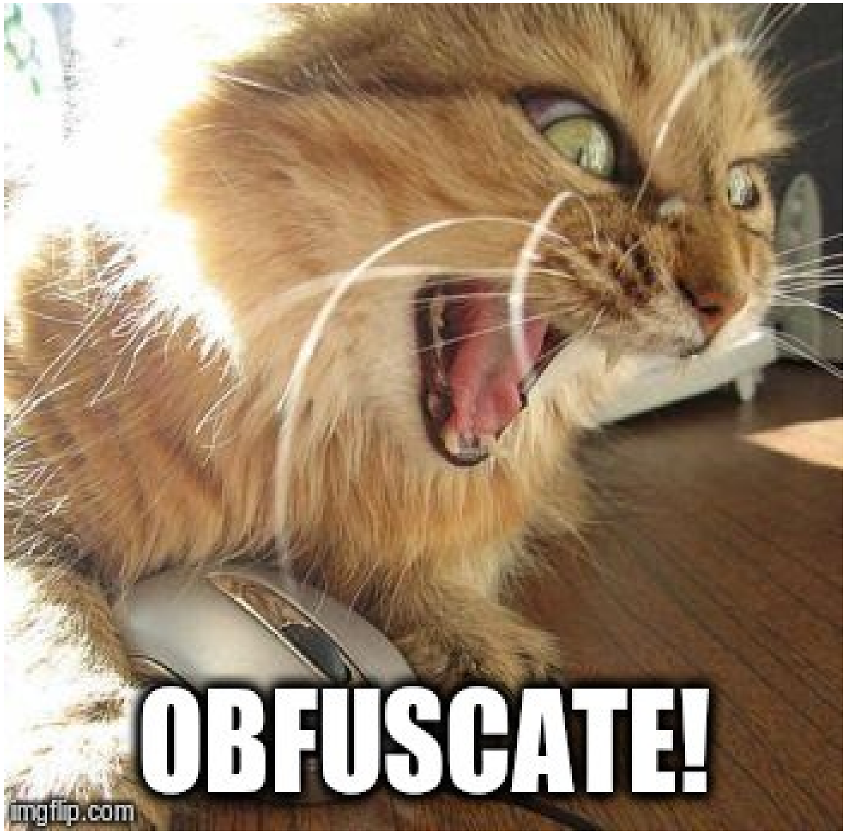
hange at each run

ld change

he value of add(2,3)
e analyzing more
e add function

nt change at each

DEMO



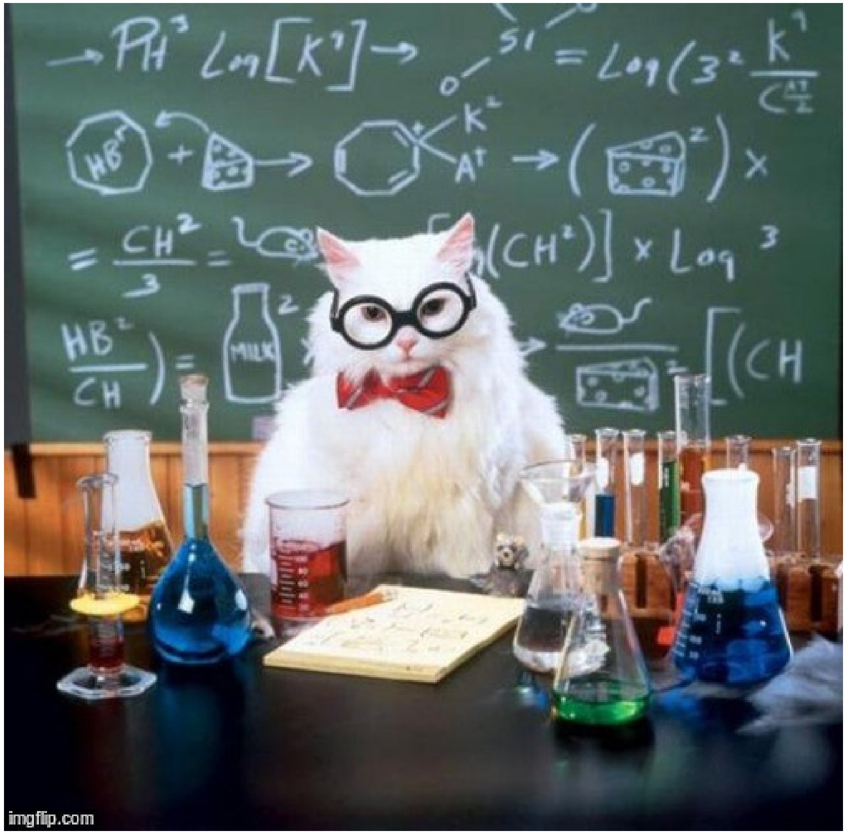
After

```python
_93fc941387 = [10, 9, 19, 19, 9, 64, 99, 55, 99, 1, 49, 55, 99, 91, 49]
_bbed80d741 = _93fc941387[11]
_93fc941387[11] = _93fc941387[3]
_93fc941387[3] = _bbed80d741
import struct
import binascii
import math
lrot = lambda x, n: x << n | x >> 32 * (_93fc941387[11] % _93fc941387[1]) - n


class MD5:
    A, B, C, D = 7 * (_93fc941387[12] % _93fc941387[0]) ** 0 + 2 * (_93fc941387
        [8] % _93fc941387[0]) + 6 * (_93fc941387[6] % _93fc941387[0]) ** 2 + (
        _93fc941387[8] % _93fc941387[0]) ** 3 + 4 * (_93fc941387[2] %
        _93fc941387[0]) ** 4 + (_93fc941387[14] % _93fc941387[0]) ** 5 + 2 * (
        _93fc941387[2] % _93fc941387[0]) ** 6 + 2 * (_93fc941387[2] %
        _93fc941387[0]) ** 7 + 4 * (_93fc941387[6] % _93fc941387[0]
        ) ** 8 + 4 * (_93fc941387[12] % _93fc941387[0]) ** 9, 2 * (_93fc941387
        [4] % _93fc941387[0]) ** 0 + 3 * (_93fc941387[4] % _93fc941387[0]
        ) + 6 * (_93fc941387[8] % _93fc941387[0]) ** 2 + 2 * (_93fc941387[4
        ] % _93fc941387[0]) ** 3 + 7 * (_93fc941387[12] % _93fc941387[0]
        ) ** 4 + 3 * (_93fc941387[4] % _93fc941387[0]) ** 5 + (_93fc941387[
        4] % _93fc941387[0]) ** 6 + 4 * (_93fc941387[14] % _93fc941387[0]
        ) ** 7 + 3 * (_93fc941387[6] % _93fc941387[0]) ** 8 + (_93fc941387[
        2] % _93fc941387[0]) ** 9 + (_93fc941387[10] % _93fc941387[0]
        ) ** 10, 2562383102 * (_93fc941387[11] % _93fc941387[1]), 271733878 * (
        _93fc941387[7] % _93fc941387[1])
    r = [7 * (_93fc941387[5] % _93fc941387[1]), 3 * (_93fc941387[12] %
        _93fc941387[0]) ** 0 + _93fc941387[4] % _93fc941387[0], 17 * (
        _93fc941387[3] % _93fc941387[1]), 22 * (_93fc941387[11] %
        _93fc941387[1]), 7 * (_93fc941387[7] % _93fc941387[1]), 3 * (
        _93fc941387[4] % _93fc941387[0]) ** 0 + _93fc941387[12] %
        _93fc941387[0], 17 * (_93fc941387[7] % _93fc941387[1]), 4 * (
        _93fc941387[2] % _93fc941387[0]) ** 0 + 2 * (_93fc941387[14] %
        _93fc941387[0]), 7 * (_93fc941387[12] % _93fc941387[0]) ** 0, 3 * (
        _93fc941387[2] % _93fc941387[0]) ** 0 + _93fc941387[8] %
        _93fc941387[0], 8 * (_93fc941387[8] % _93fc941387[0]) ** 0 +
        _93fc941387[14] % _93fc941387[0], 4 * (_93fc941387[2] % _93fc941387
        [0]) ** 0 + 2 * (_93fc941387[2] % _93fc941387[0]), 7 * (_93fc941387
        [9] % _93fc941387[1]), 12 * (_93fc941387[9] % _93fc941387[1]), 17 *
        (_93fc941387[11] % _93fc941387[1]), 4 * (_93fc941387[6] %
        _93fc941387[0]) ** 0 + 2 * (_93fc941387[6] % _93fc941387[0]), 5 * (
        _93fc941387[2] % _93fc941387[0]) ** 0, 9 * (_93fc941387[3] %
        _93fc941387[1]), 5 * (_93fc941387[14] % _93fc941387[0]) ** 0 +
        _93fc941387[14] % _93fc941387[0], 20 * (_93fc941387[9] %
        _93fc941387[1]), 5 * (_93fc941387[10] % _93fc941387[0]) ** 0, 9 * (
        _93fc941387[11] % _93fc941387[1]), 5 * (_93fc941387[4] %
        _93fc941387[0]) ** 0 + _93fc941387[4] % _93fc941387[0], 20 * (
        _93fc941387[13] % _93fc941387[1]), 5 * (_93fc941387[14] %
        _93fc941387[0]) ** 0, 9 * (_93fc941387[9] % _93fc941387[1]), 14 * (
```

```python
import struct
import binascii
import math

lrot = lambda x, n: (x << n) | (x >> (32 - n))


class MD5():

    A, B, C, D = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)

    # r specifies the per-round shift amounts
    r = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
         5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
         4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
         6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

    # Use binary integer part of the sines of integers (Radians) as consta
    k = [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in range(

    def __init__(self, message):
        length = struct.pack('<Q', len(message) * 8)
        while len(message) > 64:
            self._handle(message[:64])
            message = message[64:]
        message += '\x80'
        message += '\x00' * ((56 - len(message) % 64) % 64)
        message += length
        while len(message):
            self._handle(message[:64])
            message = message[64:]

    def _handle(self, chunk):
        w = list(struct.unpack('<' + 'I' * 16, chunk))

        a, b, c, d = self.A, self.B, self.C, self.D

        for i in range(64):
            if i < 16:
                f = (b & c) | ((~b) & d)
                g = i
            elif i < 32:
                f = (d & b) | ((~d) & c)
                g = (5 * i + 1) % 16
            elif i < 48:
                f = b ^ c ^ d
                g = (3 * i + 5) % 16
            else:
                f = c ^ (b | (~d))
                g = (7 * i) % 16
```

Before

LET'S OBFUSCATE CONTROL FLOW

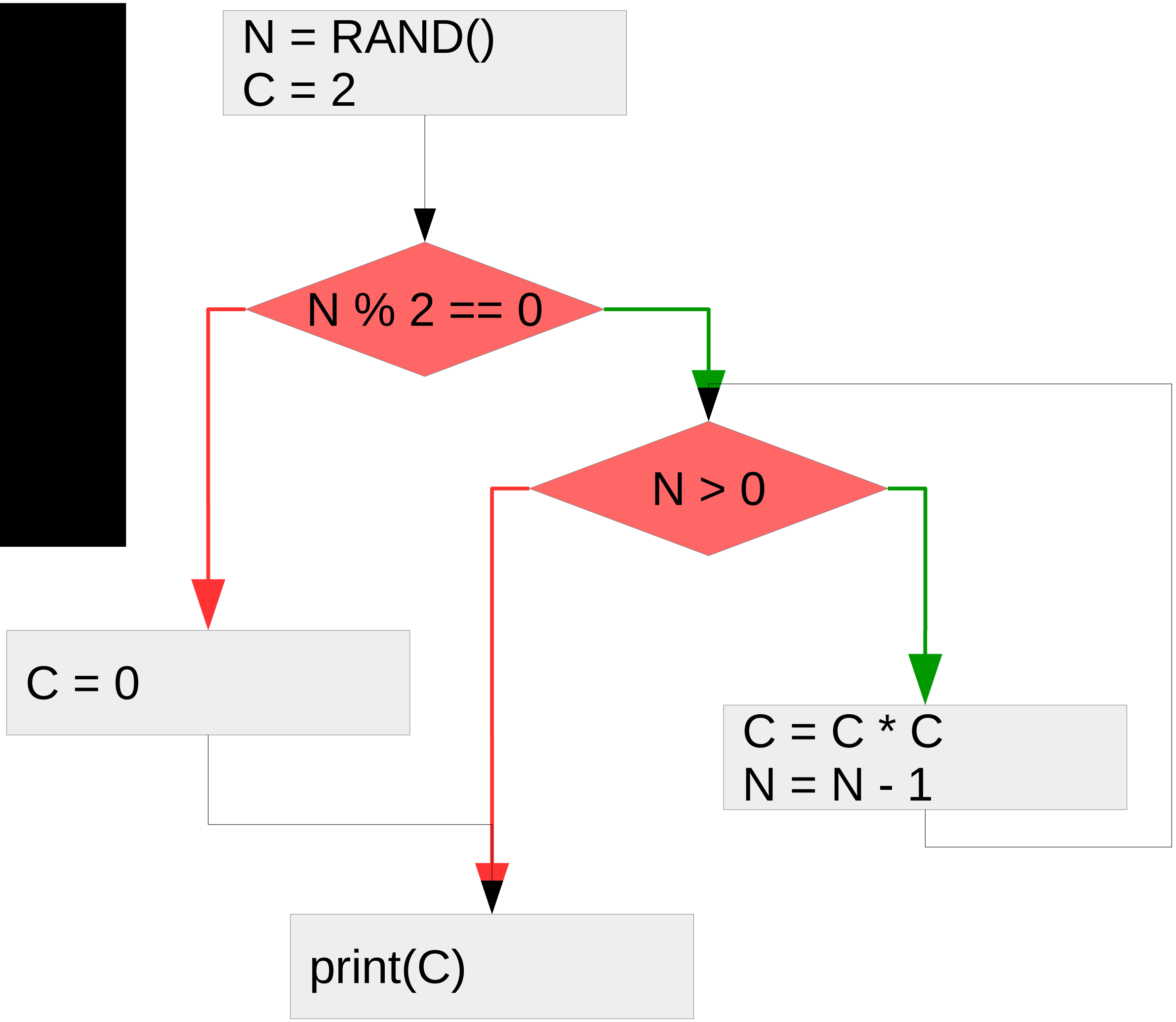


Control Flow Graph

- All programs make use of control instructions
  - if, while, for, switch, etc
- They define a “Control Flow Graph”
  - Composed of test and instructions blocks
  - Define which instruction is executed when
  - Wise attacker can deduce information of the CFG
- We want to obfuscate that

Control Flow Graph - Example

```
N = RAND()
C = 2
if (N % 2 == 0) {
    while (N > 0) {
        C = C * C
        N = N - 1
    }
} else {
    C = 0
}
print(C)
```



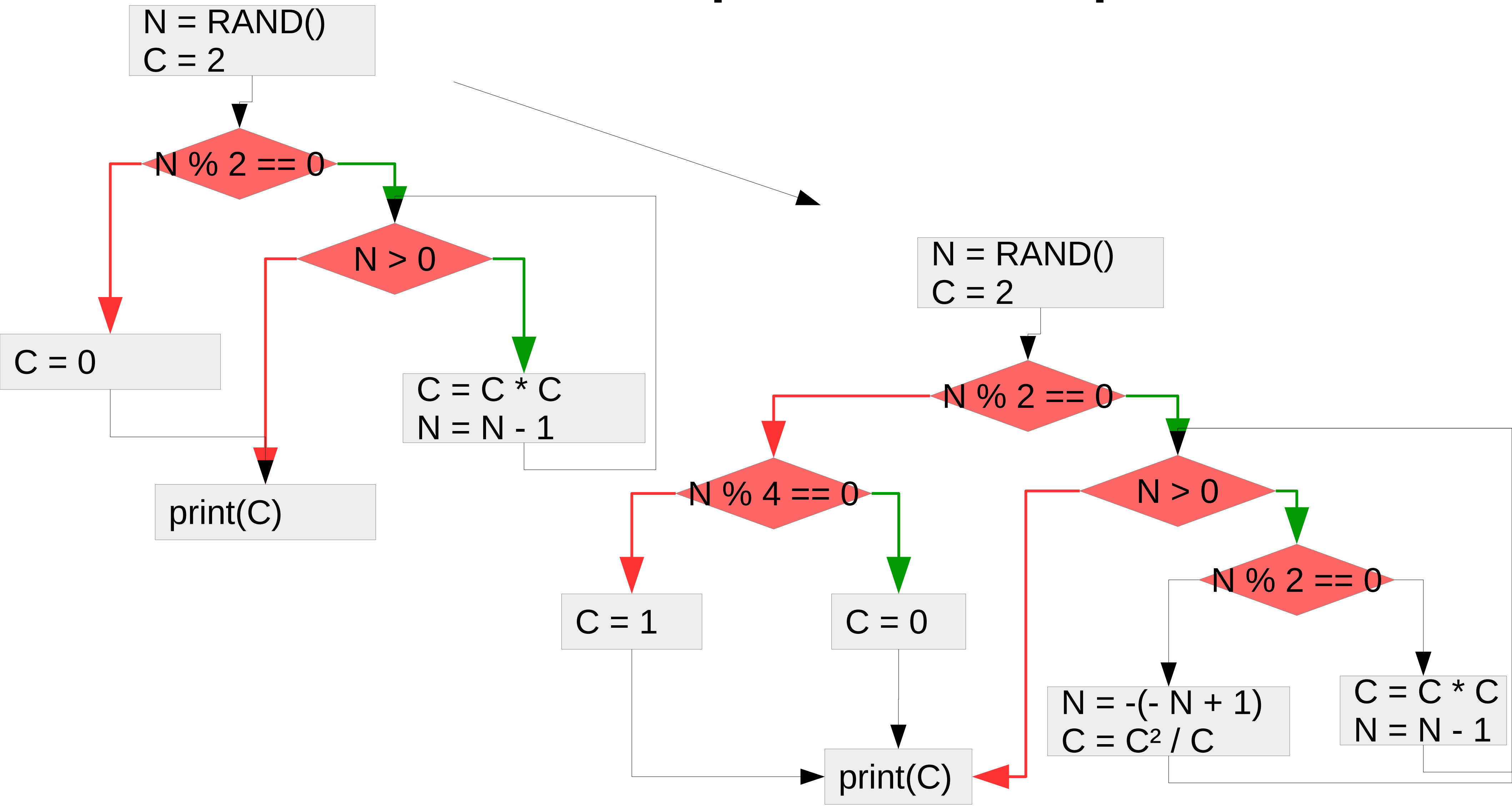
Control Flow Graph – Naive (?)

- Ideas:
  - Add dead branches
  - Duplicate branches

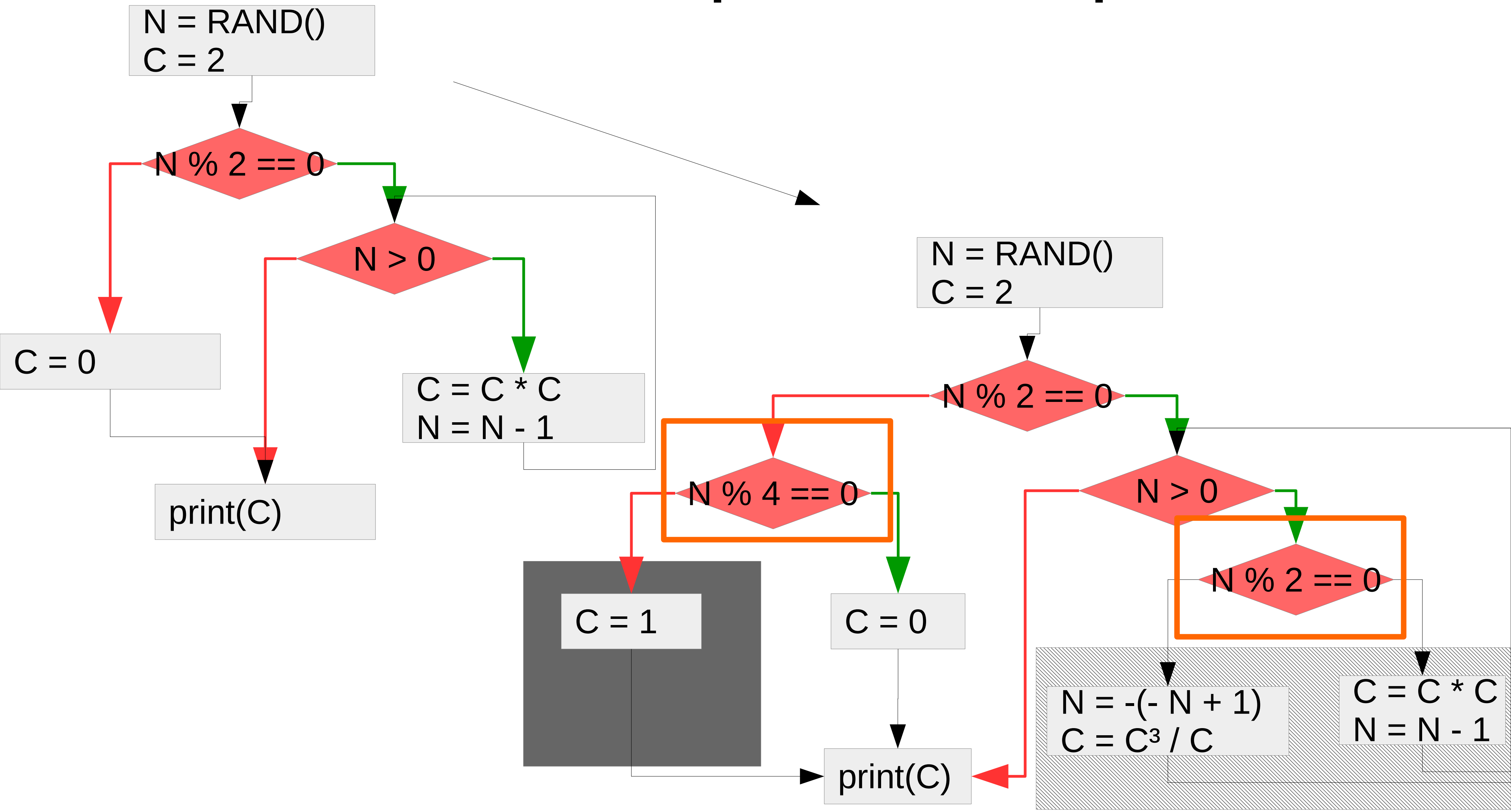
- Increases the amount of code to analyze

→ Use opaque predicates !

Control Flow Graph - Example



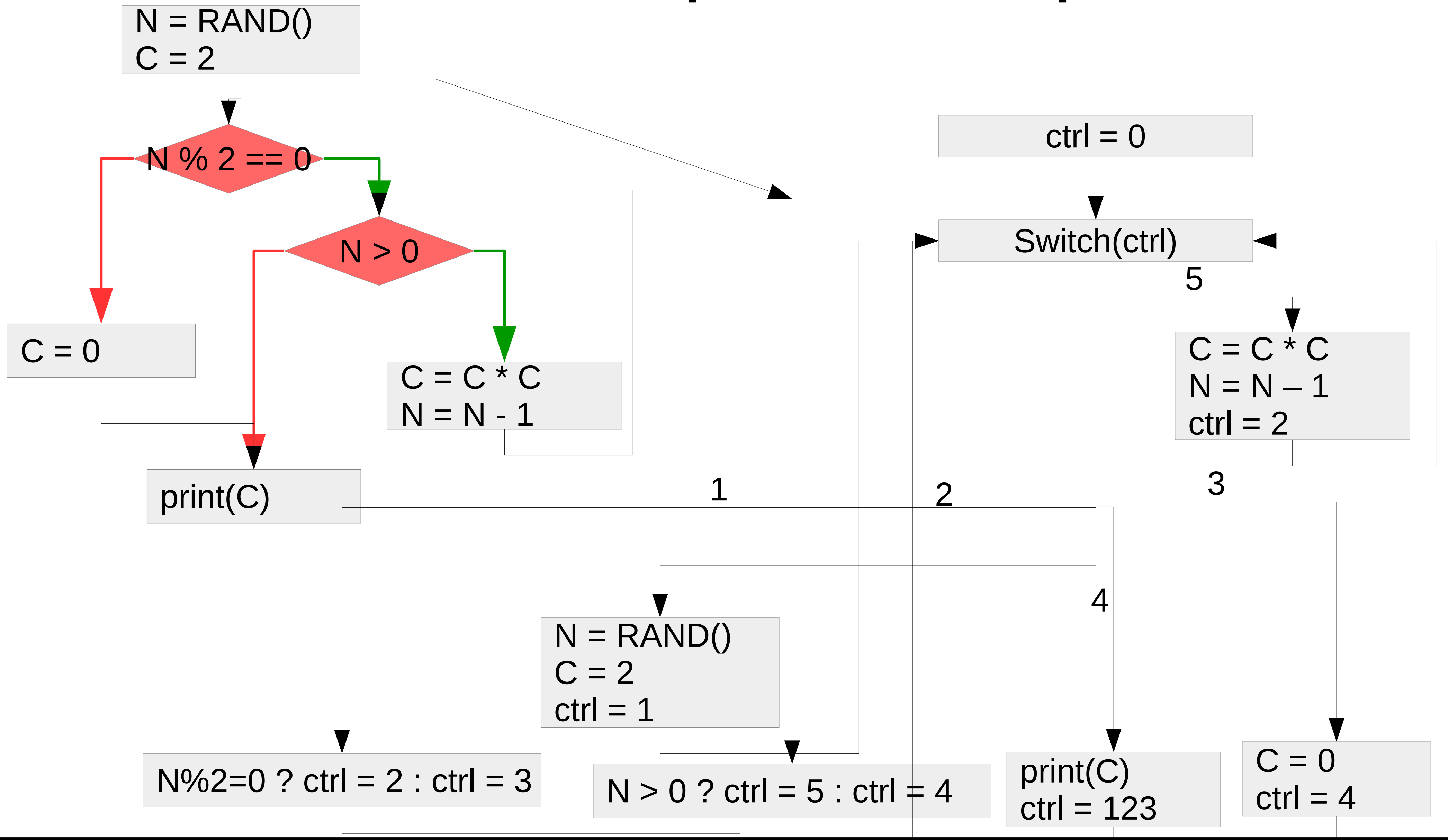
Control Flow Graph - Example



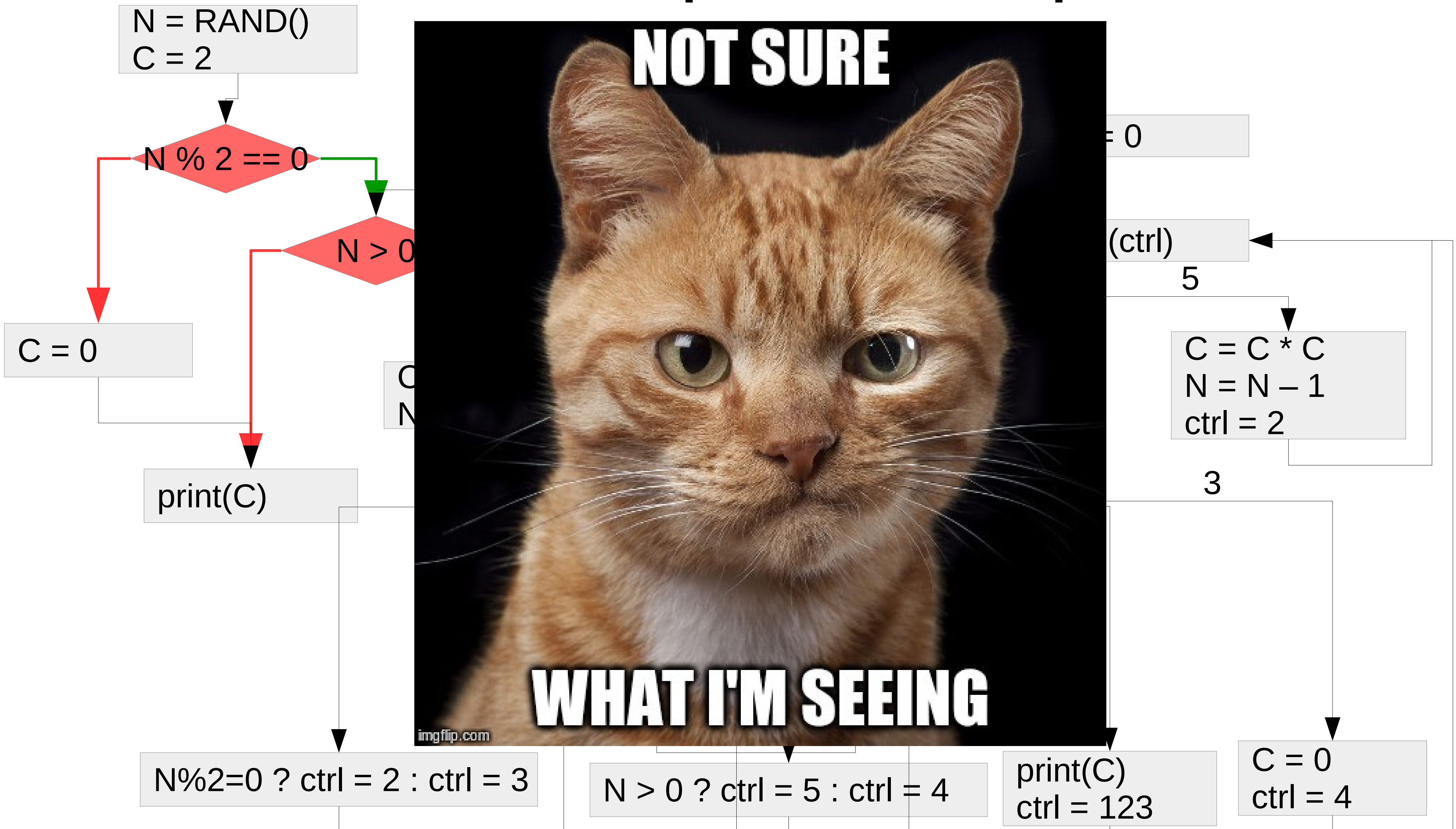
Control Flow Graph – Flattening

- Can we do better (i.e. destroy the graph) ?
- Yes! We can flatten the graph
  - Technique called *Chenxification* after Chenxi Wang
  - Improved by Lazlo & Kiss
- The idea:
  - Replace the whole program by a big switch / case
  - Put all instruction blocks in it
  - Jump on blocks depending on a control value

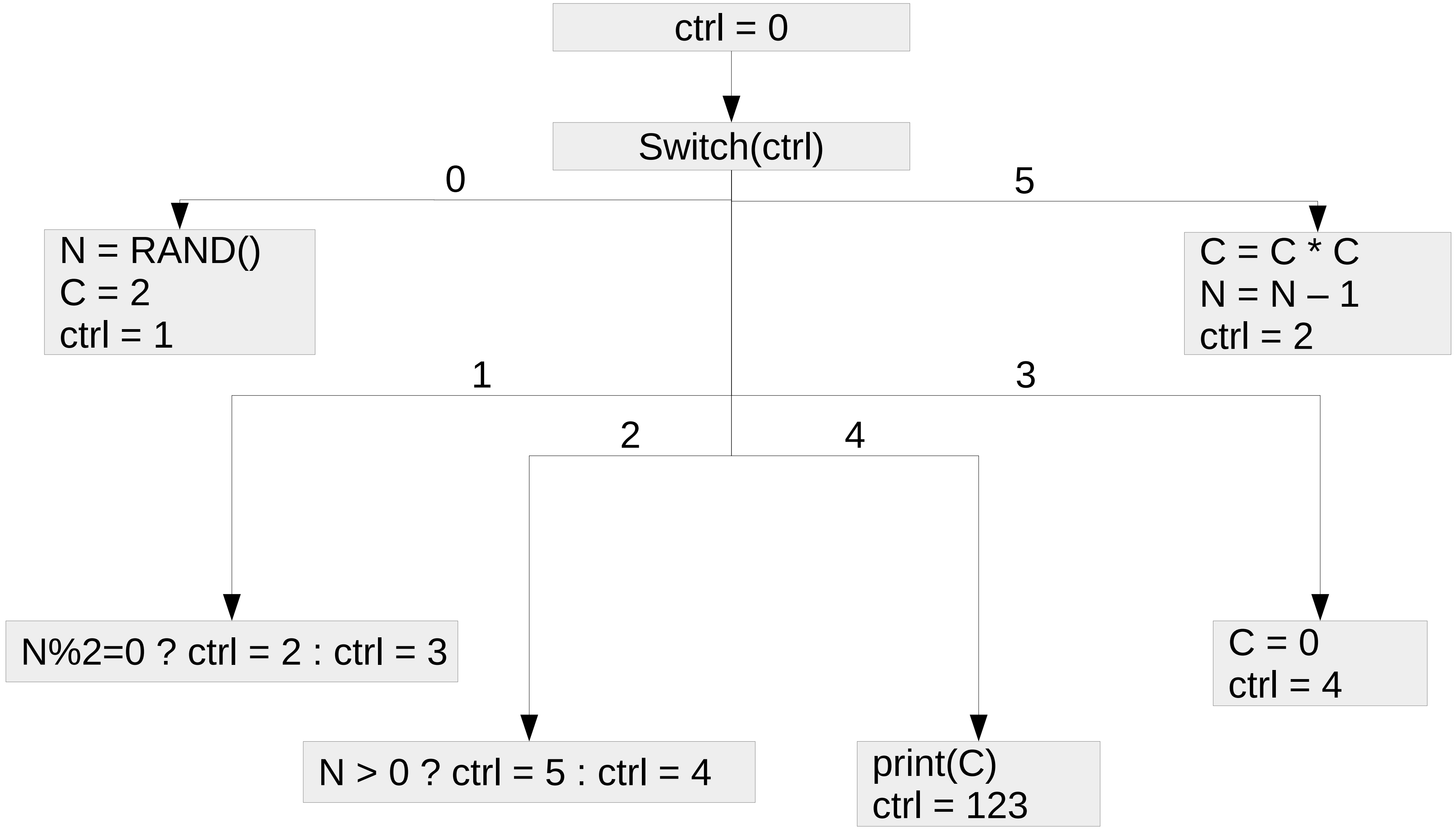
Control Flow Graph - Example



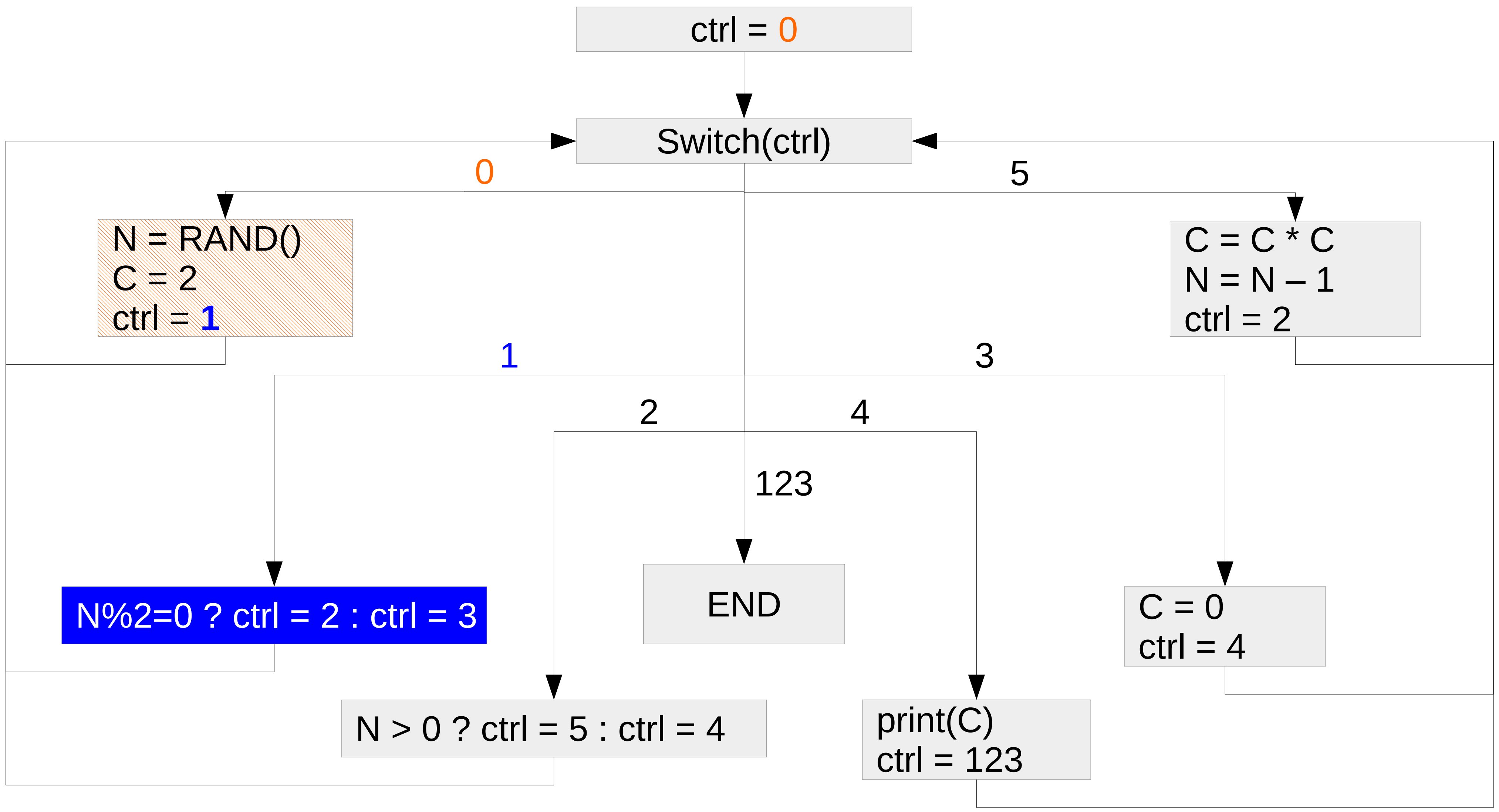
Control Flow Graph - Example



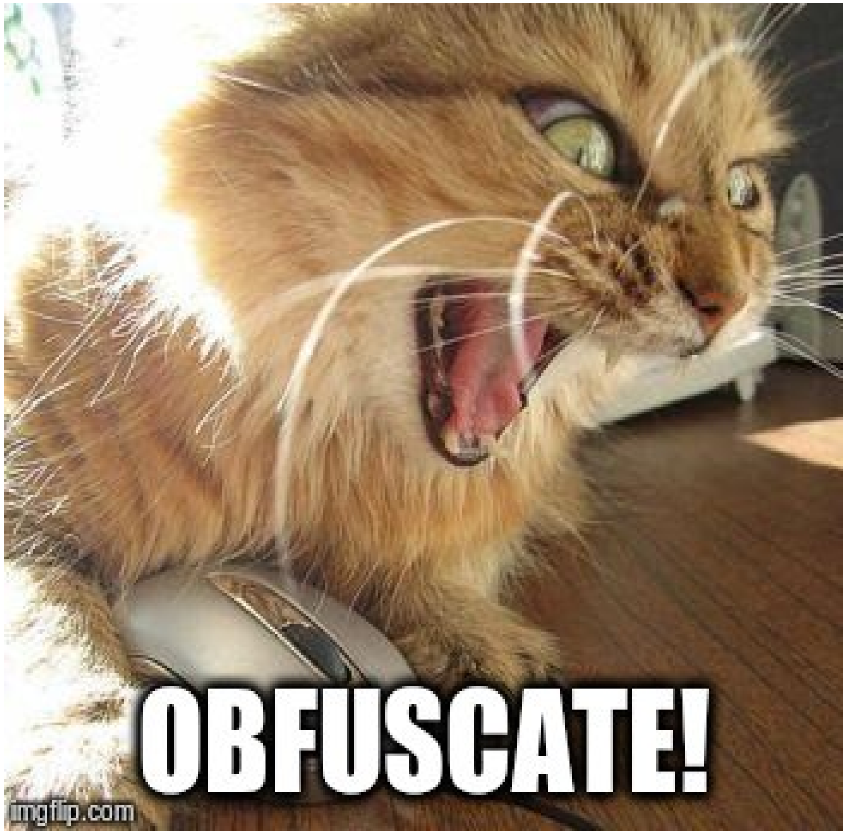
Control Flow Graph - Example



Control Flow Graph - Example



DEMO



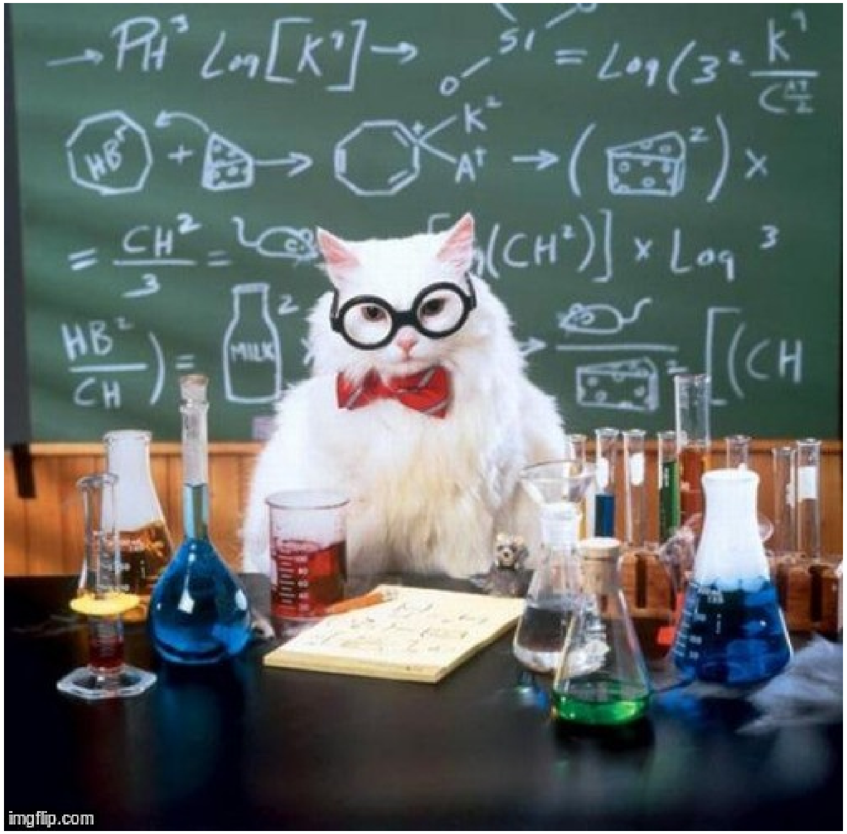
```python
import struct
import binascii
import math

lrot = lambda x, n: (x << n) | (x >> (32 - n))


class MD5():

    A, B, C, D = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)

    # r specifies the per-round shift amounts
    r = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
         5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
         4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
         6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

    # Use binary integer part of the sines of integers (Radians) as co
    k = [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in ra

    def __init__(self, message):
        length = struct.pack('<Q', len(message) * 8)
        while len(message) > 64:
            self._handle(message[:64])
            message = message[64:]
        message += '\x80'
        message += '\x00' * ((56 - len(message) % 64) % 64)
        message += length
        while len(message):
            self._handle(message[:64])
            message = message[64:]

    def _handle(self, chunk):
        w = list(struct.unpack('<' + 'I' * 16, chunk))

        a, b, c, d = self.A, self.B, self.C, self.D

        for i in range(64):
            if i < 16:
                f = (b & c) | ((~b) & d)
                g = i
            elif i < 32:
                f = (d & b) | ((~d) & c)
                g = (5 * i + 1) % 16
            elif i < 48:
                f = b ^ c ^ d
                g = (3 * i + 5) % 16
            else:
                f = c ^ (b | (~d))
                g = (7 * i) % 16
```

Before

After

```python
import struct
import binascii
import math
lrot = lambda x, n: x << n | x >> 32 - n


class MD5:
    A, B, C, D = 1732584193, 4023233417, 2562383102, 271733878
    r = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 5, 9,
         14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 4, 11, 16, 23, 4,
         11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 6, 10, 15, 21, 6, 10, 15,
         21, 6, 10, 15, 21, 6, 10, 15, 21]
    k = [int(math.floor(abs(math.sin(i + 1)) * 2 ** 32)) for i in range(64)]

    def __init__(self, message):
        _37fe1e = 81038L
        _020a4d = None
        while 65842L != _37fe1e:
            if _37fe1e == 170846L:
                if len(message):
                    _37fe1e = 95123
                else:
                    _37fe1e = 65842
            elif _37fe1e == 81038L:
                length = struct.pack('<Q', len(message) * 8)
                _37fe1e = 845416
            elif _37fe1e == 629076L:
                message += '\x80'
                message += '\x00' * ((56 - len(message) % 64) % 64)
                message += length
                _37fe1e = 170846
            elif _37fe1e == 95123L:
                self._handle(message[:64])
                message = message[64:]
                _37fe1e = 170846
            elif _37fe1e == 225646L:
                self._handle(message[:64])
                message = message[64:]
                _37fe1e = 845416
            elif _37fe1e == 845416L:
                if len(message) > 64:
                    _37fe1e = 225646
                else:
                    _37fe1e = 629076
```

PUTTING IT ALL TOGETHER



Putting it all together

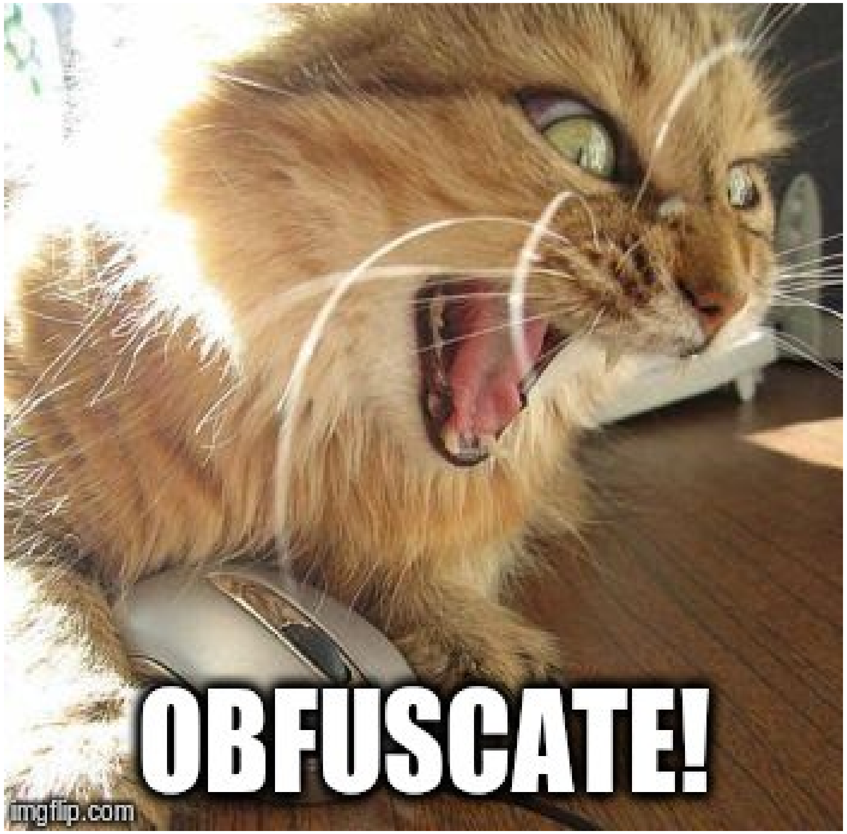
- We have three obfuscation transforms
- We should be able to combine them
  - Choose the correct order to maximize efficiency
  - Use data obfuscation to mask flattening control
  - Optionally iterate some transforms
- Keep in mind the performance impact
  - The execution time can increase significantly
  - The program size can explode
  - Maybe necessary to target sensitive functions

Putting it all together

- Keep in mind the performance loss

	SIZE	TIME
FLATTENING	+ 100 %	< +10 %
RENAMING	+0 %	+0 %
ARRAY ALIASING	x 10	+11 %

DEMO



After

```python
    def __init__(self, _39d98514e2):
        _987be1bfc9 = _05d0ad3a32[4]
        _05d0ad3a32[4] = _05d0ad3a32[8]
        _05d0ad3a32[8] = _987be1bfc9
        _f68c4e = (_05d0ad3a32[9] % _05d0ad3a32[6]) ** 0 + 2 * (_05d0ad3a32
            [9] % _05d0ad3a32[6]) + (_05d0ad3a32[5] % _05d0ad3a32[6]
            ) ** 2 + 3 * (_05d0ad3a32[5] % _05d0ad3a32[6]) ** 3 + 3 * (
            _05d0ad3a32[1] % _05d0ad3a32[6]) ** 4 + 3 * (_05d0ad3a32[9] %
            _05d0ad3a32[6]) ** 5 + (_05d0ad3a32[13] % _05d0ad3a32[6]
            ) ** 6 + 3 * (_05d0ad3a32[13] % _05d0ad3a32[6]) ** 7 + 3 * (
            _05d0ad3a32[5] % _05d0ad3a32[6]) ** 9
        _20d741 = None
        while 3 * (_05d0ad3a32[13] % _05d0ad3a32[6]) ** 0 + 3 * (_05d0ad3a32
            [5] % _05d0ad3a32[6]) ** 2 + 3 * (_05d0ad3a32[1] % _05d0ad3a32[6]
            ) ** 4 + 3 * (_05d0ad3a32[13] % _05d0ad3a32[6]) ** 5 + 3 * (
            _05d0ad3a32[13] % _05d0ad3a32[6]) ** 6 + 2 * (_05d0ad3a32[13] %
            _05d0ad3a32[6]) ** 7 + 2 * (_05d0ad3a32[1] % _05d0ad3a32[6]
            ) ** 8 != _f68c4e:
            _a9a2038c7a = _05d0ad3a32[12]
            _05d0ad3a32[12] = _05d0ad3a32[4]
            _05d0ad3a32[4] = _a9a2038c7a
            if _f68c4e == 2 * (_05d0ad3a32[5] % _05d0ad3a32[6]) ** 0 + 2 * (
                _05d0ad3a32[13] % _05d0ad3a32[6]) + (_05d0ad3a32[5] %
                _05d0ad3a32[6]) ** 2 + (_05d0ad3a32[9] % _05d0ad3a32[6]
                ) ** 3 + 3 * (_05d0ad3a32[1] % _05d0ad3a32[6]) ** 5 + 2 * (
                _05d0ad3a32[5] % _05d0ad3a32[6]) ** 6 + 3 * (_05d0ad3a32[5] %
                _05d0ad3a32[6]) ** 7 + (_05d0ad3a32[1] % _05d0ad3a32[6]
                ) ** 8 + 2 * (_05d0ad3a32[5] % _05d0ad3a32[6]) ** 9:
                _32030861a0 = _05d0ad3a32[8]
                _05d0ad3a32[8] = _05d0ad3a32[12]
                _05d0ad3a32[12] = _32030861a0
                if len(_39d98514e2) > (_05d0ad3a32[4] % _05d0ad3a32[7]
                    ) ** 0 + 7 * (_05d0ad3a32[12] % _05d0ad3a32[7]):
                    _b7ccbce2db = _05d0ad3a32[9]
                    _05d0ad3a32[9] = _05d0ad3a32[13]
                    _05d0ad3a32[13] = _b7ccbce2db
                    _f68c4e = 8 * (_05d0ad3a32[8] % _05d0ad3a32[7]
                        ) ** 0 + 4 * (_05d0ad3a32[12] % _05d0ad3a32[7]) + 7 * (
                        _05d0ad3a32[8] % _05d0ad3a32[7]) ** 2 + (_05d0ad3a32
                        [4] % _05d0ad3a32[7]) ** 3 + 5 * (_05d0ad3a32[8] %
                        _05d0ad3a32[7]) ** 4 + 6 * (_05d0ad3a32[4] %
                        _05d0ad3a32[7]) ** 5 + (_05d0ad3a32[4] % _05d0ad3a32[7]
                        ) ** 6
                else:
                    _f68c4e = 6 * (_05d0ad3a32[4] % _05d0ad3a32[7]
                        ) ** 0 + 4 * (_05d0ad3a32[4] % _05d0ad3a32[7]) + 8 * (
                        _05d0ad3a32[12] % _05d0ad3a32[7]) ** 2 + 5 * (
                        _05d0ad3a32[8] % _05d0ad3a32[7]) ** 3 + 6 * (
                        _05d0ad3a32[4] % _05d0ad3a32[7]) ** 4 + 3 * (
                        _05d0ad3a32[8] % _05d0ad3a32[7]) ** 5
```

Before

```python
    def __init__(self, message):
        length = struct.pack('<Q', len(message) * 8)
        while len(message) > 64:
            self._handle(message[:64])
            message = message[64:]
        message += '\x80'
        message += '\x00' * ((56 - len(message) % 64) %
        message += length
        while len(message):
            self._handle(message[:64])
            message = message[64:]

    def _handle(self, chunk):
        w = list(struct.unpack('<' + 'I' * 16, chunk))

        a, b, c, d = self.A, self.B, self.C, self.D

        for i in range(64):
            if i < 16:
                f = (b & c) | ((~b) & d)
                g = i
            elif i < 32:
                f = (d & b) | ((~d) & c)
                g = (5 * i + 1) % 16
            elif i < 48:
                f = b ^ c ^ d
                g = (3 * i + 5) % 16
            else:
                f = c ^ (b | (~d))
                g = (7 * i) % 16

            x = b + lrot((a + f + self.k[i] + w[g]) & (
            a, b, c, d = d, x & 0xffffffff, b, c

        self.A = (self.A + a) & 0xffffffff
        self.B = (self.B + b) & 0xffffffff
        self.C = (self.C + c) & 0xffffffff
        self.D = (self.D + d) & 0xffffffff
```

Conclusion

- We achieve a nice looking obfuscation
  - Using somehow simple transforms
- But might not hold against advanced analysis
  - In particular dynamic analysis

    → Debugging, Symbolic execution, etc

- What about dynamic obfuscation?

    → Self modifying programs, white box crypto, etc

Conclusion

- We achieve a nice looking obfuscation
  - Using somehow simple transforms
- But might not hold against adv
  - In particular dynamic analysis

→ Debugging, Symbolic executio

- What about dynamic obfuscati

→ Self modifying programs, white

ANY QUESTIONS ?

Thank you for your attention