Obfuscator-LLVM

-

Software Obfuscation for the Masses

area41 // Zürich // 02-06-2014

Hes·so
University of Applied Sciences
Western Switzerland

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

AREA41
area41 Security Conference

Who?

Who?

- *Pascal Junod (@cryptopathe)*, Julien Rinaldini (@pyknite), Johan Wehrli (@jowehrli) // HEIG-VD
- Julie Michielin // Kudelski Security
- Several bachelor & master students

Why?

Black-Box Adversaries

- Play the security game according to the rules
- Interact with components according to the defined APIs
- Adversaries considered in most « provably-secure » schemes by cryptographers

Grey-Box Adversaries

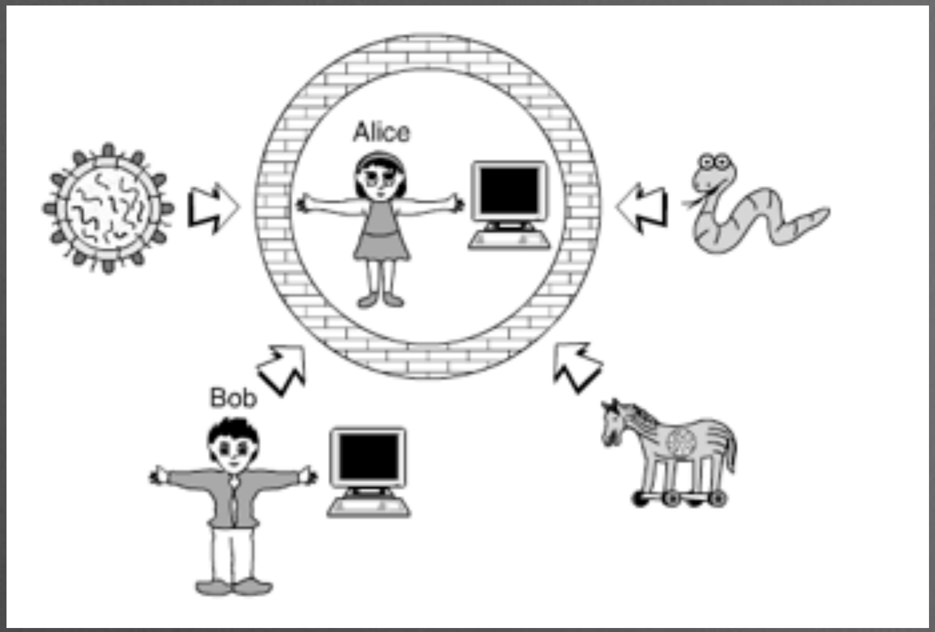
- Adversaries looking to exploit additional « side-channel » information
  - Time
  - Power/EM leakage
  - Faults

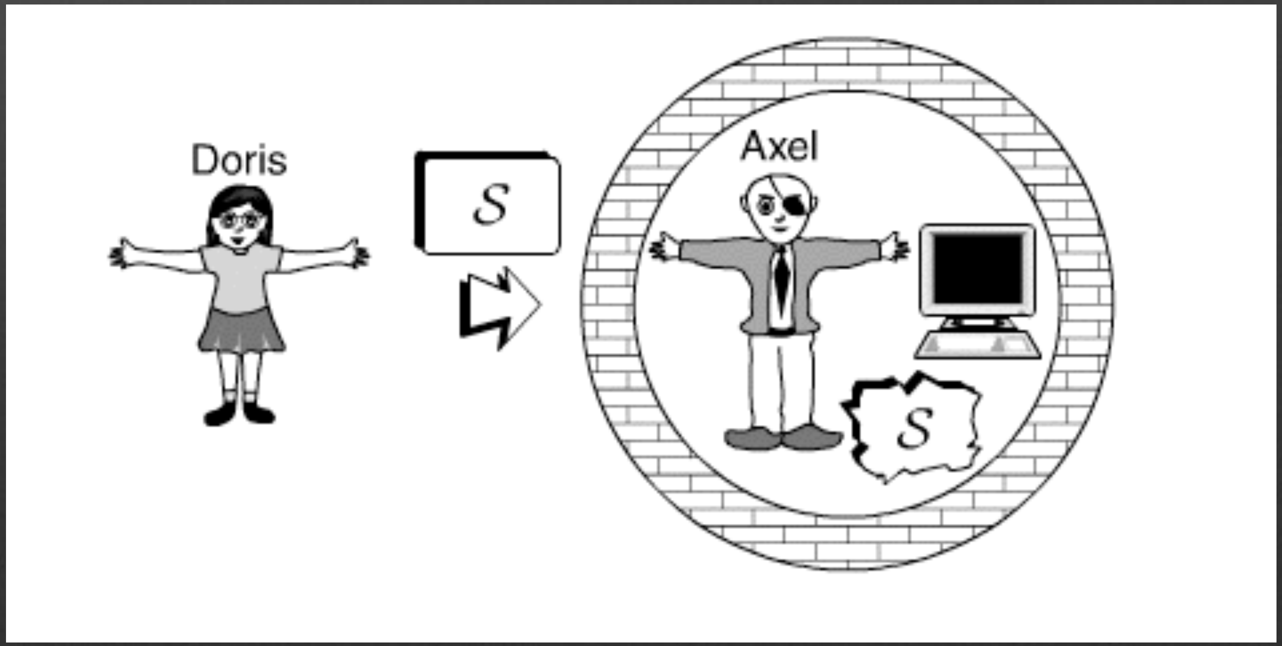
White-Box Adversaries

- (Almost) completely master SW/HW
- Can read every memory
- Can disturb every computation at will

White-Box Adversaries

- Examples in real life:
  - DRM circumventing
  - License management scheme cracking
  - Rogue SW reverse engineering, IP stealing
  - Malware analysis
  - Interoperability work
  - Security audits


Alice
Bob


Doris
Axel
S
S

Attack Scenarios

- Reverse engineering
  - IP stealing, cryptographic secret extractions
- Code modification
- Code distribution

Main Motivations

- Bad guys use software protection all the time. Why not good guys?
- Performing research on automatic desobfuscation not always easy (costly commercial tools)
- No open-source C/C++ obfuscation tool available that is capable of performing code transformation

Problems

- Dual-use tools
- Example of the iOS 7 jailbreak Christmas drama
  - Obfuscated code
    - 0-day jailbreaking exploit ?
    - Bundled pirate app repository ?
  - See http://crypto.junod.info/2013/12/24/about-obfuscator-llvm-dual-use-tools-and-academic-ethics/
- How to audit a SW protection tool with respect to backdoor insertion?

What?

Software Protection

- Goals of software protection:
  - Code more difficult to understand
  - Code more difficult to modify
  - Code more diverse

Obfuscation

Tamperproofing

Watermarking

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah
xinU / lreP rehtona tsuJ";sub p{
@p{"r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;
($q*=2)+=$f=!fork;map{$P=$P[$f^ord
($p{$_})&6];$p{$_}=/ ^$P/ix?$P:close$_}keys
%p}p;p;p;p;p;map{$p{$_}=~/^[P.]/&&
close$_}%p;wait until$?;map{/^r/&&<$_>}%p;
$_=$d[$q];sleep rand(2)if/\S/;print
```

```
                                    for(v A((u A((e A((r-2?0:(V A(1[U])),"C")
),system("stty raw -echo min 0"),fread(l,78114,1,e),B(e),"B")),"A")); 118-(x
=*c++); (y=x/8%8,z=(x&199)-4 S 1 S 1 S 186 S 2 S 2 S 3 S 0,r=(y>5)*2+y,z=(x&
207)-1 S 2 S 6 S 2 S 182 S 4)?D(0)D(1)D(2)D(3)D(4)D(5)D(6)D(7)(z=x-2 C C C C
C C C C+129 S 6 S 4 S 6 S 8 S 8 S 6 S 2 S 2 S 12)?x/64-1?((0 O a(y)=a(x) O 9
[o]=a(5),8[o]=a(4) O 237==*c++?((int (*)())(2-*c++?fwrite:fread))(l+*k+1[k]*
256,128,1,(fseek(y=5[k]-1?u:v,((3[k]|4[k]<<8)<<7|2[k])<<7,Q=0),y)):0 O y=a(5
),z=a(4),a(5)=a(3),a(4)=a(2),a(3)=y,a(2)=z O c=l+d(5) O y=l[x=d(9)],z=l[++x]
,x[l]=a(4),l[--x]=a(5),a(5)=y,a(4)=z O 2-*c?Z||read(0,&Z,1),1&*c++?Q=Z,Z=0:(
Q=!!Z):(c++,Q=r=V?fgetc(V):-1,s=s&~1|r<0) O++c,write(1,&7[o],1) O z=c+2-l,w,
c=l+q O p,c=l+z O c=l+q O s^=1 O Q=q[l] O s|=1 O q[l]=Q O Q=~Q O a(5)=l[x=q]
,a(4)=l[++x] O s|=s&16|9<Q%16?Q+=6,16:0,z=s|=1&s|Q>159?Q+=96,1:0,y=Q,h(s<<8)
O l[x=q]=a(5),l[++x]=a(4) O x=Q%2,Q=Q/2+s%2*128,s=s&~1|x O Q=l[d(3)]O x=Q  /
128,Q=Q*2+s%2,s=s&~1|x O l[d(3)]=Q O s=s&~1|1&Q,Q=Q/2|Q<<7 O Q=l[d(1)]O s=~1
&s|Q>>7,Q=Q*2|Q>>7 O l[d(1)]=Q O m y n(0,-,7)y) O m z=0,y=Q|=x,h(y) O m z=0,
y=Q^=x,h(y) O m z=Q*2|2*x,y=Q&=x,h(y) O m Q n(s%2,-,7)y) O m Q n(0,-,7)y)  O
m Q n(s%2,+,7)y) O m Q n(0,+,7)y) O z=r-8?d(r+1):s|Q<<8,w O p,r-8?o[r+1]=z,r
[o]=z>>8:(s=~40&z|2,Q=z>>8) O r[o]--||--o[r-1]O a(5)=z=a(5)+r[o],a(4)=z=a(4)
+o[r-1]+z/256,s=~1&s|z>>8 O ++o[r+1]||r[o]++O o[r+1]=*c++,r[o]=*c++O z=c-l,w
,c=y*8+l O x=q,b z=c-l,w,c=l+x) O x=q,b c=l+x) O b p,c=l+z) O a(y)=*c++O r=y
            ,x=0,a(r)n(1,-,y)s<<8) O r=y,x=0,a(r)n(1,+,y)s<<8))));
            system("stty cooked echo"); B((B((V?B(V):0,u)),v)); }
```

On the (Im)possibility of Obfuscating Programs*

Boaz Barak† Oded Goldreich‡ Russell Impagliazzo§ Steven Rudich¶
Amit Sahai‖ Salil Vadhan** Ke Yang††

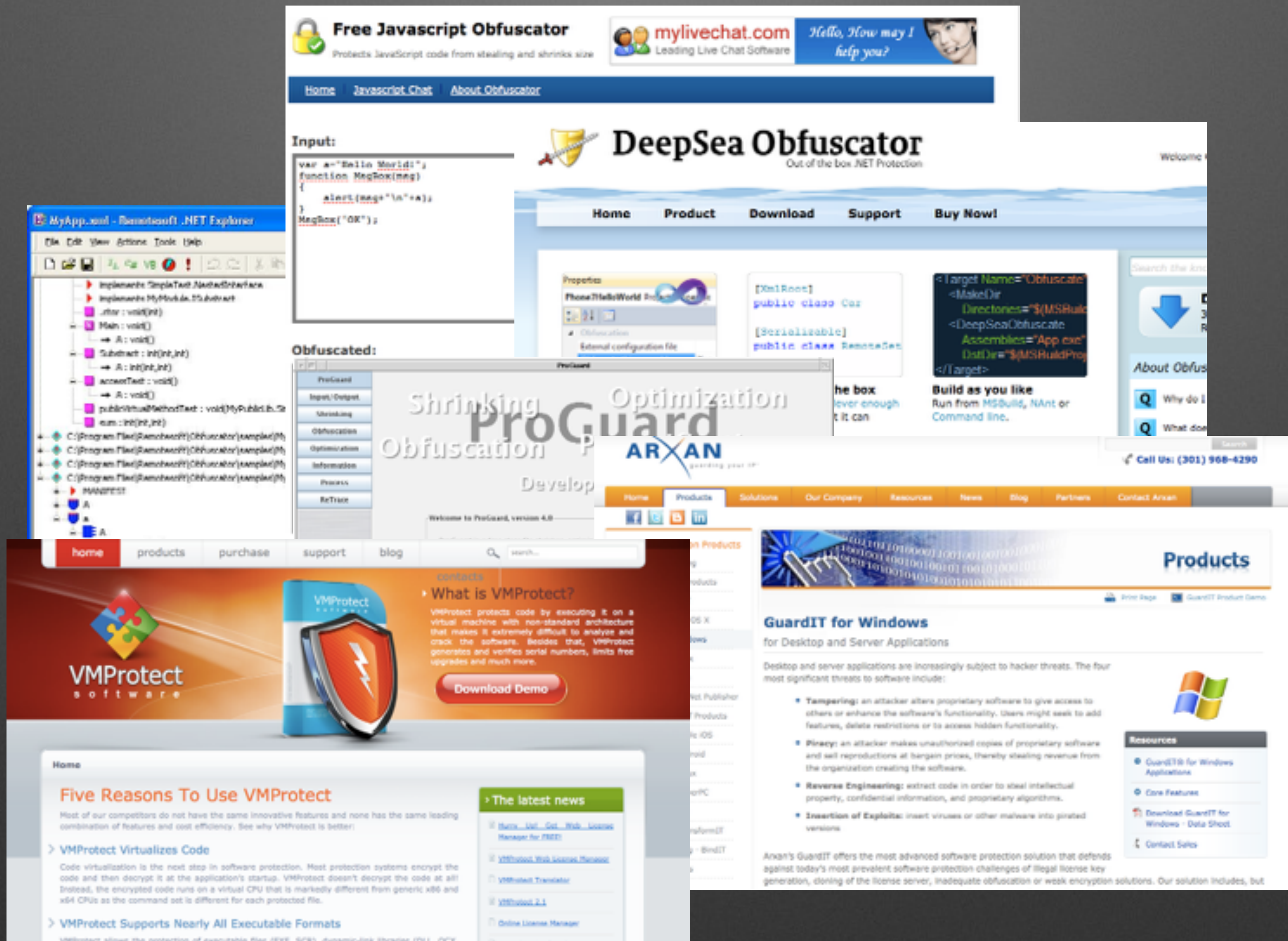
July 29, 2010

Abstract

Informally, an *obfuscator* $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program (or circuit) P and produces a new program $\mathcal{O}(P)$ that has the same functionality as P yet is "unintelligible" in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem. Most of these applications are based on an interpretation of the "unintelligibility" condition in obfuscation as meaning that $\mathcal{O}(P)$ is a "virtual black box," in the sense that anything one can efficiently compute given $\mathcal{O}(P)$, one could also efficiently compute given oracle access to P.

In this work, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations of the above intuition, obfuscation is impossible. We prove this by constructing a family of efficient programs $\mathcal{P}$ that are *unobfuscatable* in the sense that (a) given *any* efficient program P' that computes the same function as a program $P \in \mathcal{P}$, the "source code" P can be efficiently reconstructed, yet (b) given *oracle access* to a (randomly selected) program $P \in \mathcal{P}$, no efficient algorithm can reconstruct P (or even distinguish a certain bit in the code from random) except with negligible probability.

We extend our impossibility result in a number of ways, including even obfuscators that (a) are not necessarily computable in polynomial time, (b) only approximately preserve the functionality, and (c) only need to work for very restricted models of computation ($\mathbf{TC}^0$). We also rule out several potential applications of obfuscators, by constructing "unobfuscatable" signature schemes, encryption schemes, and pseudorandom function families.


Free Javascript Obfuscator
mylivechat.com
Leading Live Chat Software
Hello, How may I help you?
Home
Javascript Chat
About Obfuscator
Input:
Obfuscated:
DeepSea Obfuscator
Out of the box .NET Protection
Home
Product
Download
Support
Buy Now!
Build as you like
ProGuard
Shrinking
Optimization
Obfuscation
ARXAN
Call Us: (301) 968-4290
Products
GuardIT for Windows
for Desktop and Server Applications
Resources
home
products
purchase
support
blog
VMProtect
software
What is VMProtect?
Download Demo
Five Reasons To Use VMProtect
VMProtect Virtualizes Code
VMProtect Supports Nearly All Executable Formats
The latest news

Caveat Emptor

- Source vs. binary obfuscation
- Supported languages/platforms
- Associated cost
- Resistance

Well-Known Techniques

- Packing
- Anti-debugging tricks insertion
- Code interleaving
- Code transformation
- Code virtualization
- ...

How?

Home

cryptopathe edited this page on 20 mai · 23 revisions

Obfuscator-LLVM is a project initiated in June 2010 by the information security group of the University of Applied Sciences and Arts Western Switzerland of Yverdon-les-Bains (HEIG-VD).

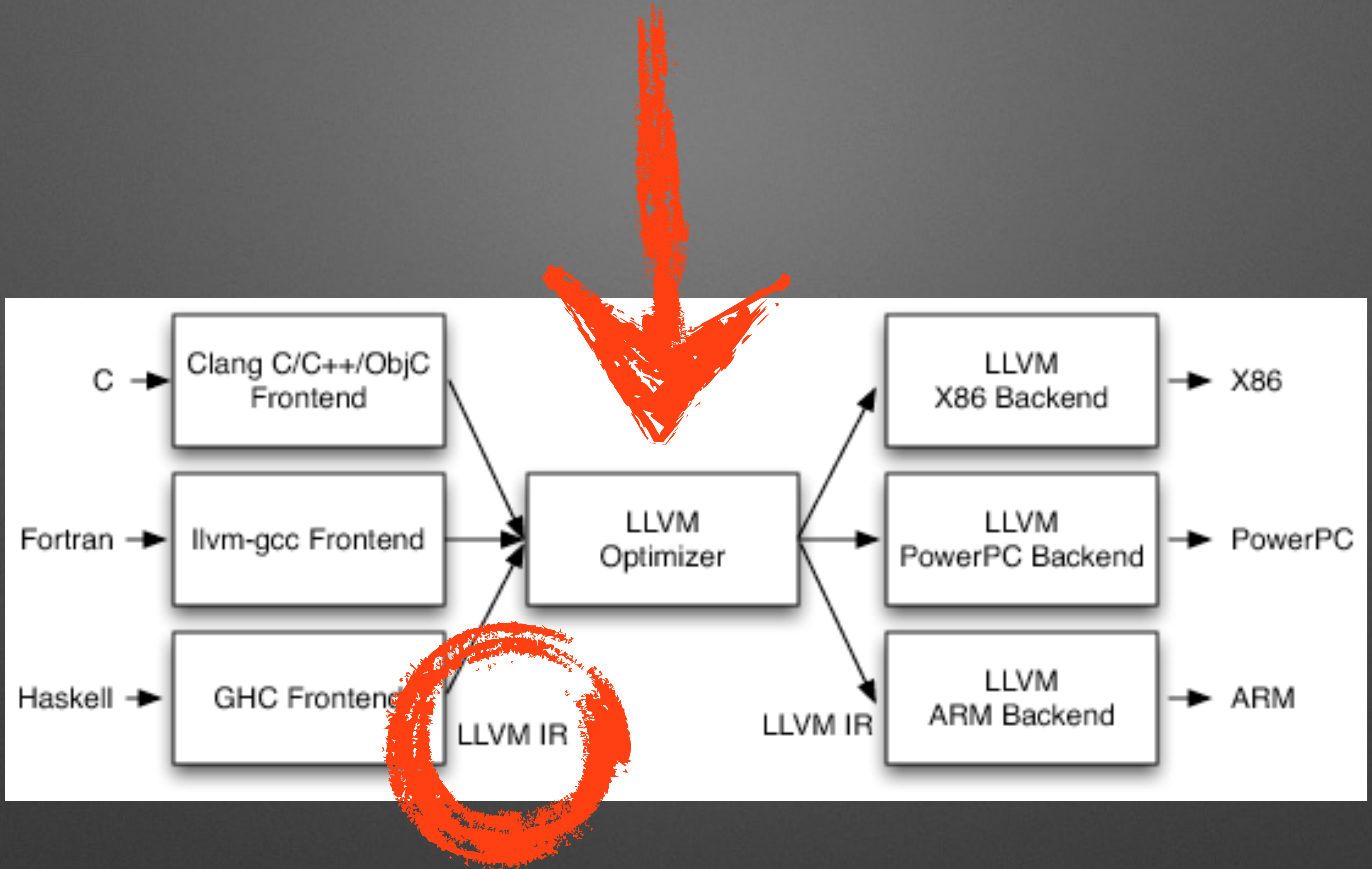
The aim of this project is to provide an open-source fork of the LLVM compilation suite able to provide increased software security through code obfuscation and tamper-proofing. As we currently mostly work at the Intermediate Representation (IR) level, our tool is compatible with all programming languages (C, C++, Objective-C, Ada and Fortran) and target platforms (x86, x86-64, PowerPC, PowerPC-64, ARM, Thumb, SPARC, Alpha, CellSPU, MIPS, MSP430, SystemZ, and XCore) currently supported by LLVM.

- Features
- Benchmarks
- Installation
- News
- FAQ
- How to Contribute
- Contributors
- License

Clone this wiki locally

https://github.com/obfusca

Clone in Desktop


C
Fortran
Haskell
Clang C/C++/ObjC Frontend
llvm-gcc Frontend
GHC Frontend
LLVM IR
LLVM Optimizer
LLVM IR
LLVM X86 Backend
LLVM PowerPC Backend
LLVM ARM Backend
X86
PowerPC
ARM

LLVM

- Complete compilation framework, competitor of GCC
- Project supported by Apple since 2005
- Very dynamic community, state-of-the-art SW architecture
- Front-ends available for C, C++, Objective-C, Fortran, Ada, Haskell, Python, Ruby, …
- Back-ends available for x86, x86-64, PowerPC, PowerPC-64, ARM, Thumb, Sparc, Alpha, MIPS, MSP430, SystemZ, XCore

Instructions Substitution

Instructions Substitution

- Replace an arithmetic or Boolean expression by an equivalent one
  - `A ^ B = (A & ~B) | (~A & B)`
  - `A + B = A - (-B)`
  - `A+B = (A+R) + (B+R) - 2*R`
  - …

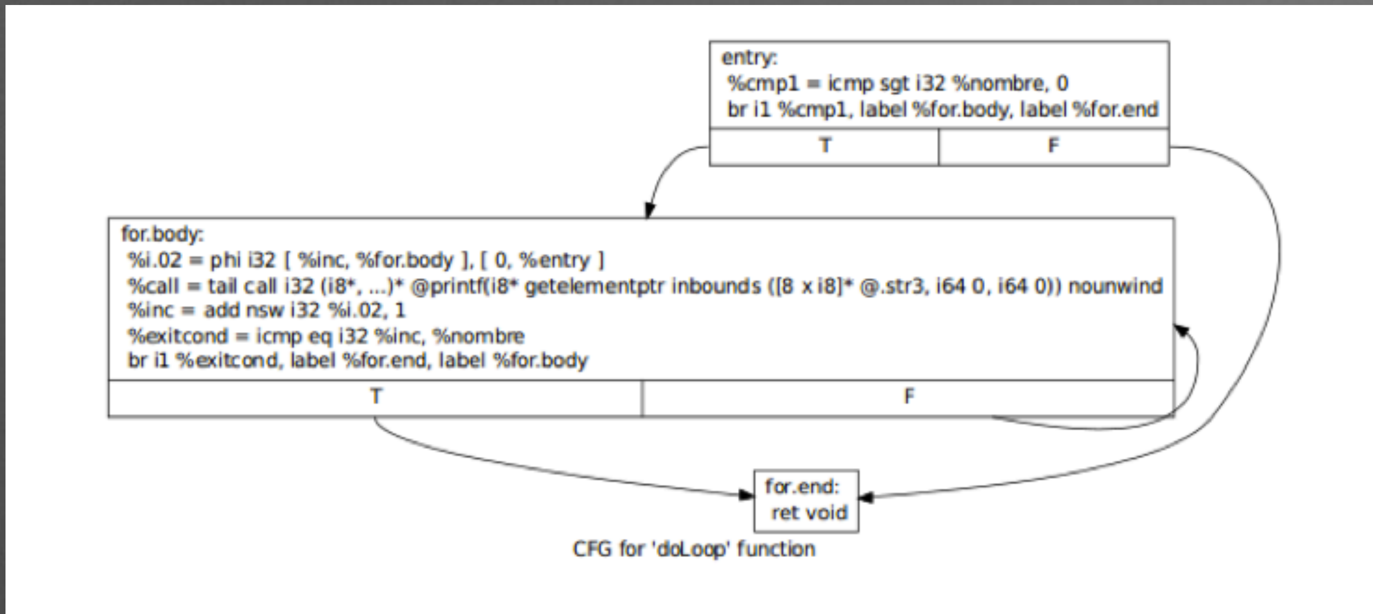
Example: AES Implementation

```
-mllvm -sub
-mllvm -sub-loop=3
```

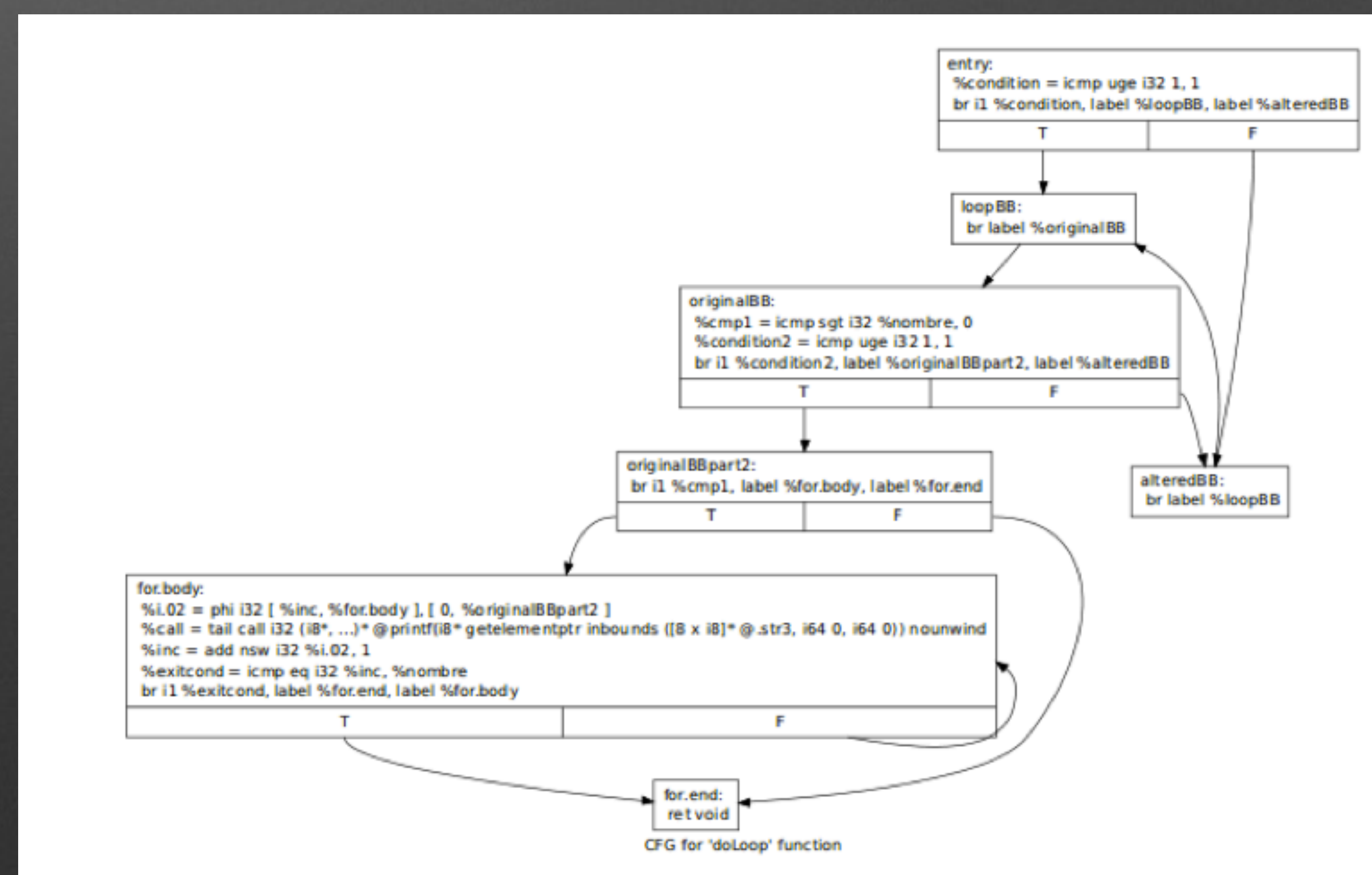
```
0000000100000c61    xor    ecx, dword [ds:r10+r11*4]
0000000100000c65    xor    ecx, dword [ds:rdx+r15*4]
0000000100000c69    mov    rdx, qword [ss:rbp-0x28+var_m16]
0000000100000c6d    xor    eax, dword [ds:rdi+rdx*4]
0000000100000c70    mov    rdx, qword [ss:rbp-0x28+var_m56]
0000000100000c74    xor    eax, dword [ds:rdx+r13+0xfffffffffffffff4]
0000000100000c79    mov    rsi, qword [ss:rbp-0x28+var_m32]
0000000100000c7d    xor    ebx, dword [ds:rdi+rsi*4]
0000000100000c80    xor    ebx, dword [ds:rdx+r13+0xfffffffffffffff8]
0000000100000c85    mov    rsi, qword [ss:rbp-0x28+var_m40]
0000000100000c89    xor    ecx, dword [ds:rdi+rsi*4]
0000000100000c8c    xor    ecx, dword [ds:rdx+r13+0xfffffffffffffffc]
0000000100000c91    xor    r8d, dword [ds:rdx+r13]
```

```
0000000100005d5a    xor    eax, 0x10844000
0000000100005d5f    mov    ebx, r12d
0000000100005d62    or     ebx, 0x81421101
0000000100005d68    mov    edi, ecx
0000000100005d6a    or     edi, 0x30200040
0000000100005d70    and    ebx, 0xb1621141
0000000100005d76    and    ecx, 0x42946c0a
0000000100005d7c    and    edi, 0xb1621141
0000000100005d82    and    r12d, 0xc0982b4
0000000100005d89    or     ecx, ebx
0000000100005d8b    or     r12d, edi
0000000100005d8e    xor    r12d, ecx
0000000100005d91    or     eax, 0x9011194
0000000100005d96    or     esi, 0x12846842
0000000100005d9c    and    eax, 0x1b8579d6
0000000100005da1    and    r11d, 0x60300408
0000000100005da8    and    esi, 0x1b8579d6
0000000100005dae    and    r9d, 0x844a8221
```

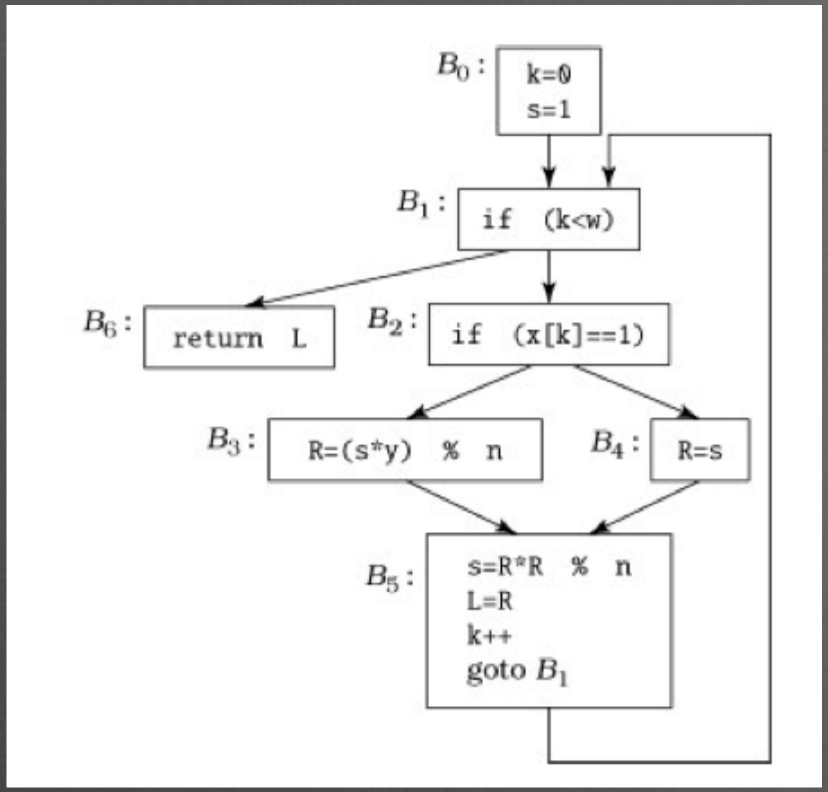
Bogus Control Flow Insertion


entry:
%cmp1 = icmp sgt i32 %nombre, 0
br i1 %cmp1, label %for.body, label %for.end
T
F
for.body:
%i.02 = phi i32 [%inc, %for.body], [0, %entry]
%call = tail call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([8 x i8]* @.str3, i64 0, i64 0)) nounwind
%inc = add nsw i32 %i.02, 1
%exitcond = icmp eq i32 %inc, %nombre
br i1 %exitcond, label %for.end, label %for.body
T
F
for.end:
ret void
CFG for 'doLoop' function

-mllvm -bcf

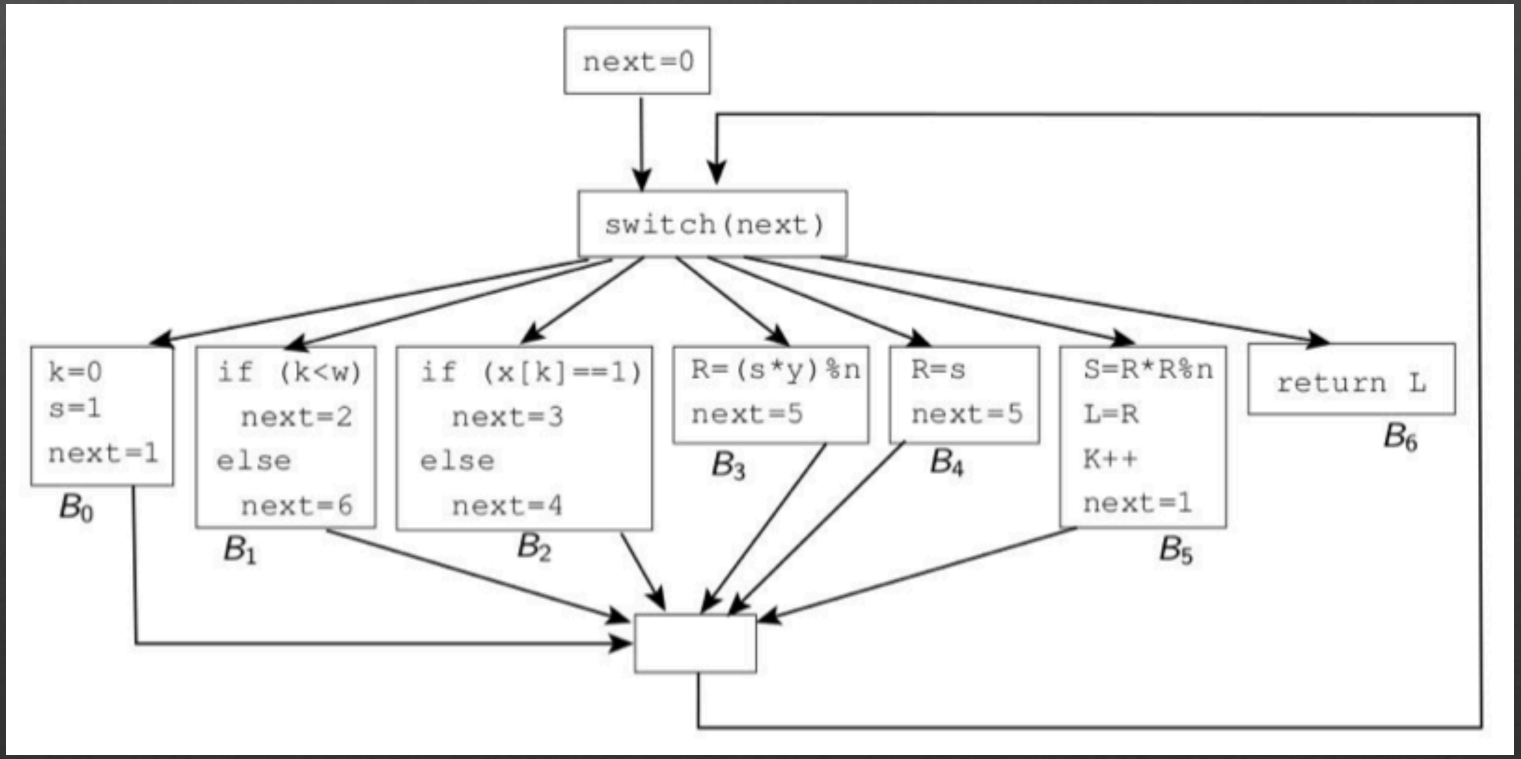

entry:
%condition = icmp uge i32 1, 1
br i1 %condition, label %loopBB, label %alteredBB
T
F
loopBB:
br label %originalBB
originalBB:
%cmp1 = icmp sgt i32 %nombre, 0
%condition2 = icmp uge i32 1, 1
br i1 %condition2, label %originalBBpart2, label %alteredBB
T
F
originalBBpart2:
br i1 %cmp1, label %for.body, label %for.end
T
F
alteredBB:
br label %loopBB
for.body:
%i.02 = phi i32 [%inc, %for.body], [0, %originalBBpart2]
%call = tail call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([8 x i8]* @.str3, i64 0, i64 0)) nounwind
%inc = add nsw i32 %i.02, 1
%exitcond = icmp eq i32 %inc, %nombre
br i1 %exitcond, label %for.end, label %for.body
T
F
for.end:
ret void
CFG for 'doLoop' function

Control Flow Flattening

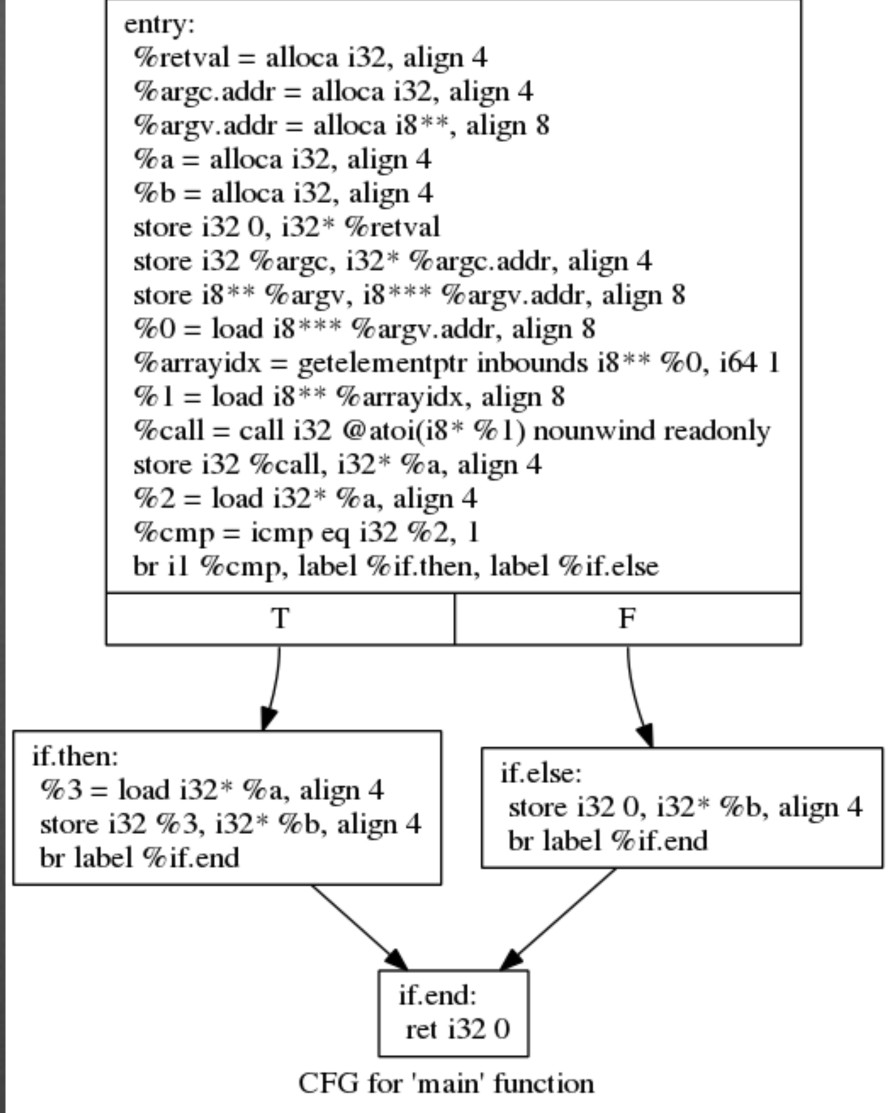


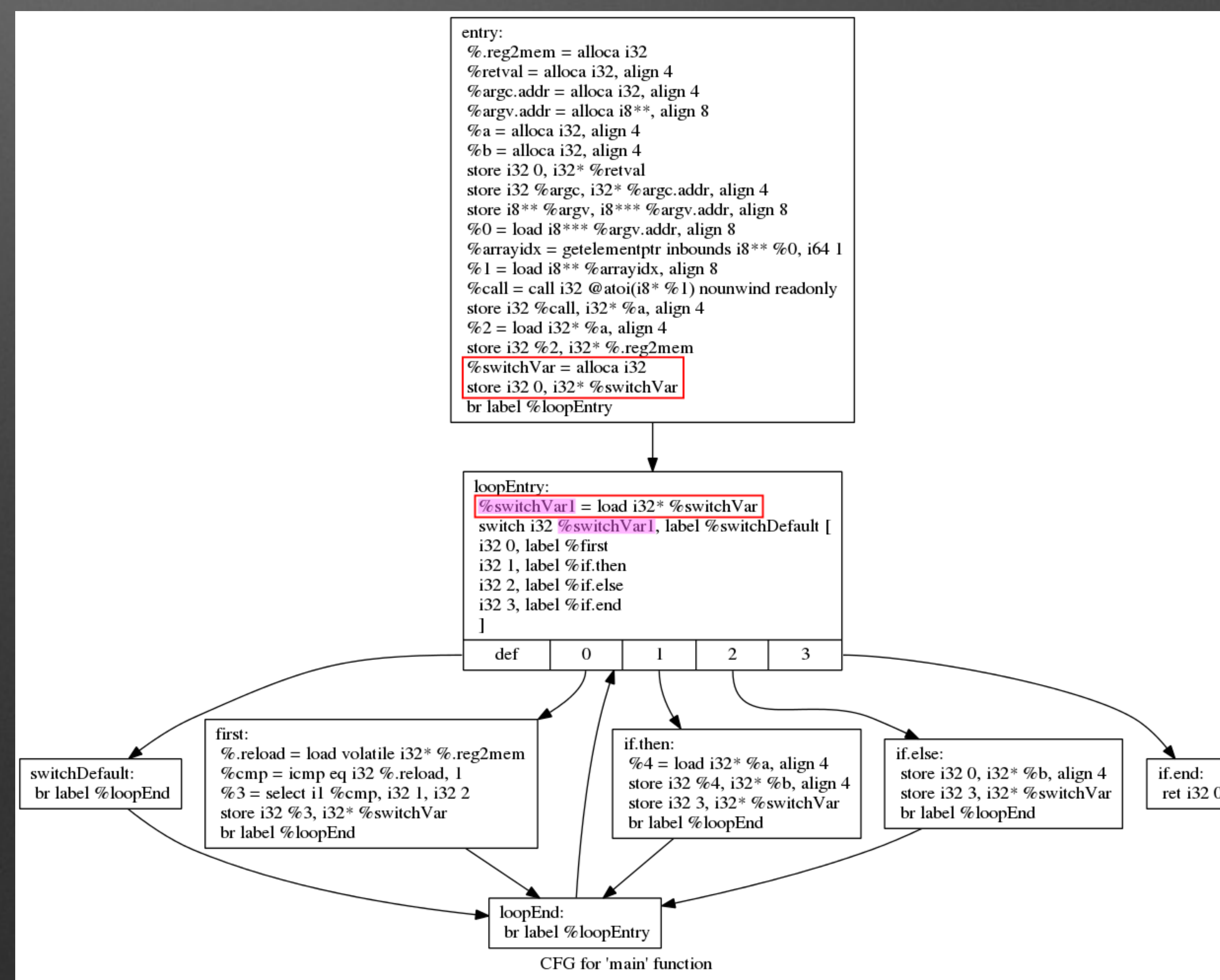
B_0 :
k=0
s=1
B_1 :
if (k<w)
B_6 :
return L
B_2 :
if (x[k]==1)
B_3 :
R=(s*y) % n
B_4 :
R=s
B_5 :
s=R*R % n
L=R
k++
goto B_1

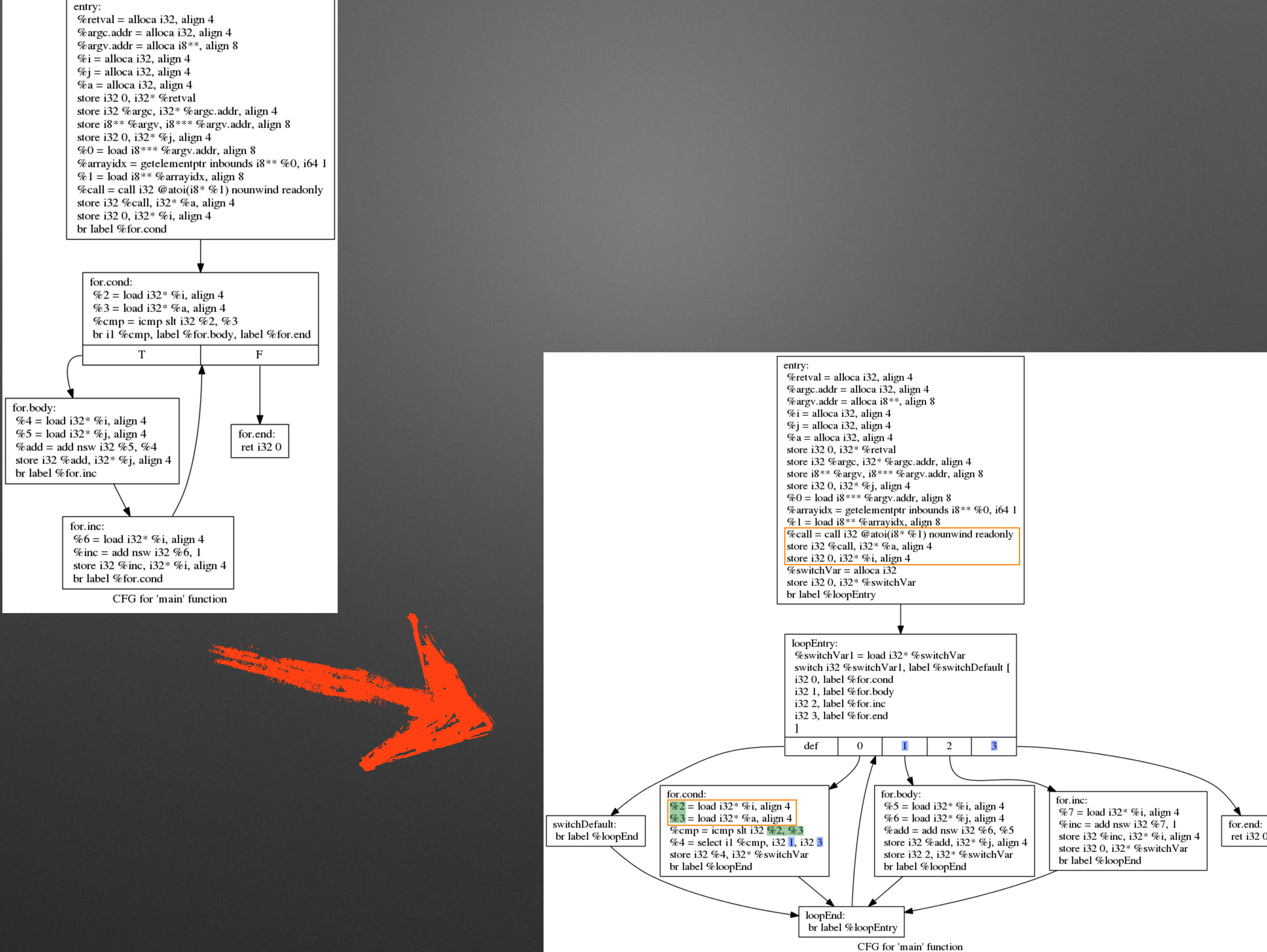
-mllvm -fla

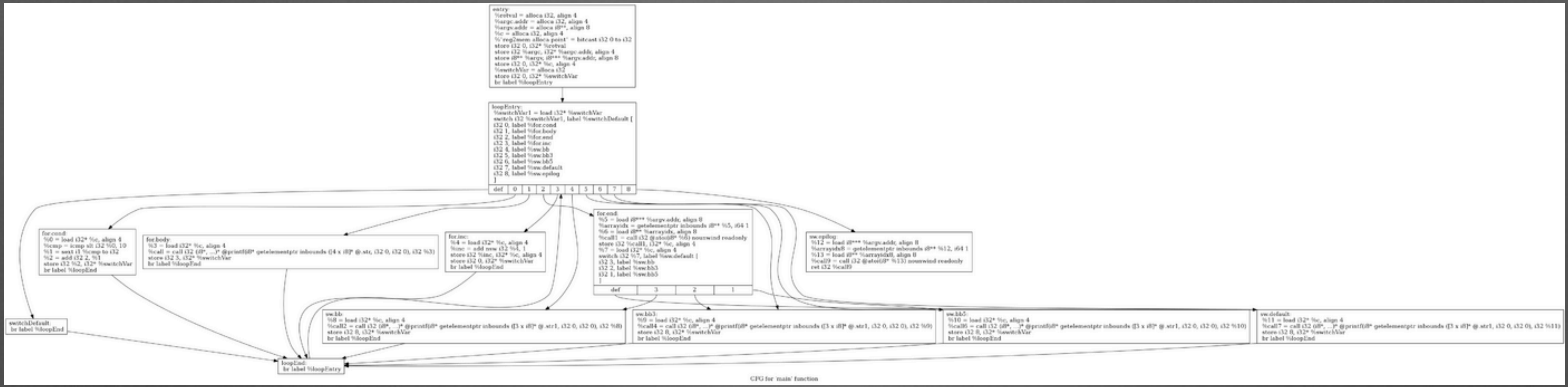


next=0
switch(next)
k=0
s=1
next=1
B_0
if (k<w)
next=2
else
next=6
B_1
if (x[k]==1)
next=3
else
next=4
B_2
R=(s*y)%n
next=5
B_3
R=s
next=5
B_4
S=R*R%n
L=R
K++
next=1
B_5
return L
B_6

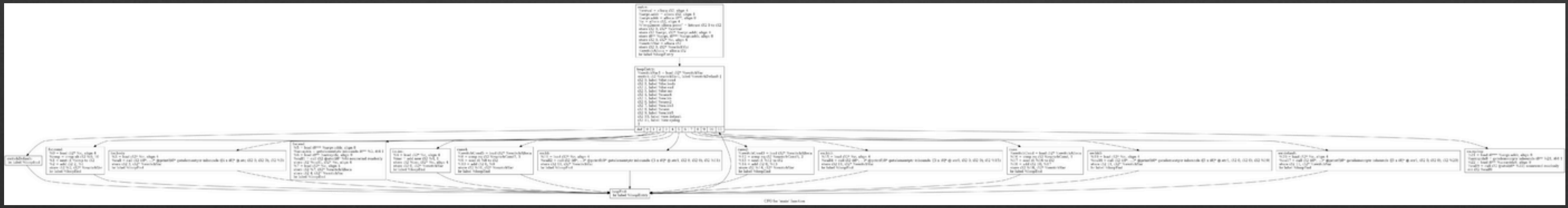

entry:
%retval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%a = alloca i32, align 4
%b = alloca i32, align 4
store i32 0, i32* %retval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %0, i64 1
%1 = load i8** %arrayidx, align 8
%call = call i32 @atoi(i8* %1) nounwind readonly
store i32 %call, i32* %a, align 4
%2 = load i32* %a, align 4
%cmp = icmp eq i32 %2, 1
br i1 %cmp, label %if.then, label %if.else
T
F
if.then:
%3 = load i32* %a, align 4
store i32 %3, i32* %b, align 4
br label %if.end
if.else:
store i32 0, i32* %b, align 4
br label %if.end
if.end:
ret i32 0
CFG for 'main' function

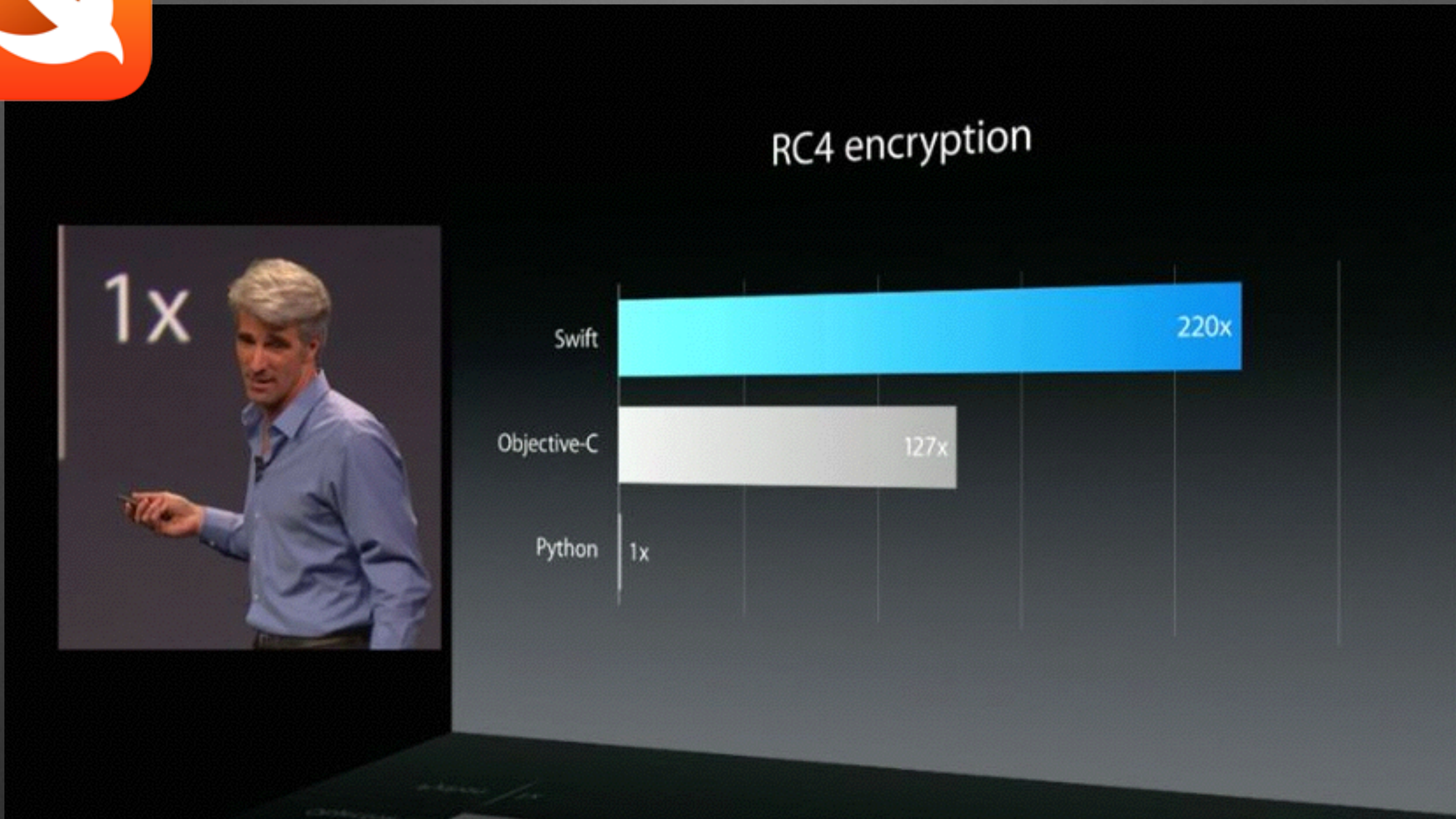

entry:
%.reg2mem = alloca i32
%retval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%a = alloca i32, align 4
%b = alloca i32, align 4
store i32 0, i32* %retval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %0, i64 1
%1 = load i8** %arrayidx, align 8
%call = call i32 @atoi(i8* %1) nounwind readonly
store i32 %call, i32* %a, align 4
%2 = load i32* %a, align 4
store i32 %2, i32* %.reg2mem
%switchVar = alloca i32
store i32 0, i32* %switchVar
br label %loopEntry
loopEntry:
%switchVar1 = load i32* %switchVar
switch i32 %switchVar1, label %switchDefault [
i32 0, label %first
i32 1, label %if.then
i32 2, label %if.else
i32 3, label %if.end
]
def
0
1
2
3
switchDefault:
br label %loopEnd
first:
%.reload = load volatile i32* %.reg2mem
%cmp = icmp eq i32 %.reload, 1
%3 = select i1 %cmp, i32 1, i32 2
store i32 %3, i32* %switchVar
br label %loopEnd
if.then:
%4 = load i32* %a, align 4
store i32 %4, i32* %b, align 4
store i32 3, i32* %switchVar
br label %loopEnd
if.else:
store i32 0, i32* %b, align 4
store i32 3, i32* %switchVar
br label %loopEnd
if.end:
ret i32 0
loopEnd:
br label %loopEntry
CFG for 'main' function


entry:
%retval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%i = alloca i32, align 4
%j = alloca i32, align 4
%a = alloca i32, align 4
store i32 0, i32* %retval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
store i32 0, i32* %j, align 4
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %0, i64 1
%1 = load i8** %arrayidx, align 8
%call = call i32 @atoi(i8* %1) nounwind readonly
store i32 %call, i32* %a, align 4
store i32 0, i32* %i, align 4
br label %for.cond
for.cond:
%2 = load i32* %i, align 4
%3 = load i32* %a, align 4
%cmp = icmp slt i32 %2, %3
br i1 %cmp, label %for.body, label %for.end
T
F
for.body:
%4 = load i32* %i, align 4
%5 = load i32* %j, align 4
%add = add nsw i32 %5, %4
store i32 %add, i32* %j, align 4
br label %for.inc
for.end:
ret i32 0
for.inc:
%6 = load i32* %i, align 4
%inc = add nsw i32 %6, 1
store i32 %inc, i32* %i, align 4
br label %for.cond
CFG for 'main' function
entry:
%retval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%i = alloca i32, align 4
%j = alloca i32, align 4
%a = alloca i32, align 4
store i32 0, i32* %retval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
store i32 0, i32* %j, align 4
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %0, i64 1
%1 = load i8** %arrayidx, align 8
%call = call i32 @atoi(i8* %1) nounwind readonly
store i32 %call, i32* %a, align 4
store i32 0, i32* %i, align 4
%switchVar = alloca i32
store i32 0, i32* %switchVar
br label %loopEntry
loopEntry:
%switchVar1 = load i32* %switchVar
switch i32 %switchVar1, label %switchDefault [
i32 0, label %for.cond
i32 1, label %for.body
i32 2, label %for.inc
i32 3, label %for.end
]
def
0
1
2
3
switchDefault:
br label %loopEnd
for.cond:
%2 = load i32* %i, align 4
%3 = load i32* %a, align 4
%cmp = icmp slt i32 %2, %3
%4 = select i1 %cmp, i32 1, i32 3
store i32 %4, i32* %switchVar
br label %loopEnd
for.body:
%5 = load i32* %i, align 4
%6 = load i32* %j, align 4
%add = add nsw i32 %6, %5
store i32 %add, i32* %j, align 4
store i32 2, i32* %switchVar
br label %loopEnd
for.inc:
%7 = load i32* %i, align 4
%inc = add nsw i32 %7, 1
store i32 %inc, i32* %i, align 4
store i32 0, i32* %switchVar
br label %loopEnd
for.end:
ret i32 0
loopEnd:
br label %loopEntry
CFG for 'main' function



entry:
%retval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%c = alloca i32, align 4
%"reg2mem alloca point" = bitcast i32 0 to i32
store i32 0, i32* %retval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
store i32 0, i32* %c, align 4
%switchVar = alloca i32
store i32 0, i32* %switchVar
br label %loopEntry
loopEntry:
%switchVar1 = load i32* %switchVar
switch i32 %switchVar1, label %switchDefault [
i32 0, label %for.cond
i32 1, label %for.body
i32 2, label %for.end
i32 3, label %for.inc
i32 4, label %sw.bb
i32 5, label %sw.bb3
i32 6, label %sw.bb5
i32 7, label %sw.default
i32 8, label %sw.epilog
]
def
0
1
2
3
4
5
6
7
8
for.cond:
%0 = load i32* %c, align 4
%cmp = icmp slt i32 %0, 10
%1 = sext i1 %cmp to i32
%2 = add i32 2, %1
store i32 %2, i32* %switchVar
br label %loopEnd
for.body:
%3 = load i32* %c, align 4
%call = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @.str, i32 0, i32 0), i32 %3)
store i32 3, i32* %switchVar
br label %loopEnd
for.inc:
%4 = load i32* %c, align 4
%inc = add nsw i32 %4, 1
store i32 %inc, i32* %c, align 4
store i32 0, i32* %switchVar
br label %loopEnd
for.end:
%5 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %5, i64 1
%6 = load i8** %arrayidx, align 8
%call1 = call i32 @atoi(i8* %6) nounwind readonly
store i32 %call1, i32* %c, align 4
%7 = load i32* %c, align 4
switch i32 %7, label %sw.default [
i32 3, label %sw.bb
i32 2, label %sw.bb3
i32 1, label %sw.bb5
]
def
3
2
1
sw.epilog:
%12 = load i8*** %argv.addr, align 8
%arrayidx8 = getelementptr inbounds i8** %12, i64 1
%13 = load i8** %arrayidx8, align 8
%call9 = call i32 @atoi(i8* %13) nounwind readonly
ret i32 %call9
switchDefault:
br label %loopEnd
sw.bb:
%8 = load i32* %c, align 4
%call2 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3 x i8]* @.str1, i32 0, i32 0), i32 %8)
store i32 8, i32* %switchVar
br label %loopEnd
sw.bb3:
%9 = load i32* %c, align 4
%call4 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3 x i8]* @.str1, i32 0, i32 0), i32 %9)
store i32 8, i32* %switchVar
br label %loopEnd
sw.bb5:
%10 = load i32* %c, align 4
%call6 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3 x i8]* @.str1, i32 0, i32 0), i32 %10)
store i32 8, i32* %switchVar
br label %loopEnd
sw.default:
%11 = load i32* %c, align 4
%call7 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3 x i8]* @.str1, i32 0, i32 0), i32 %11)
store i32 8, i32* %switchVar
br label %loopEnd
loopEnd:
br label %loopEntry
CFG for 'main' function



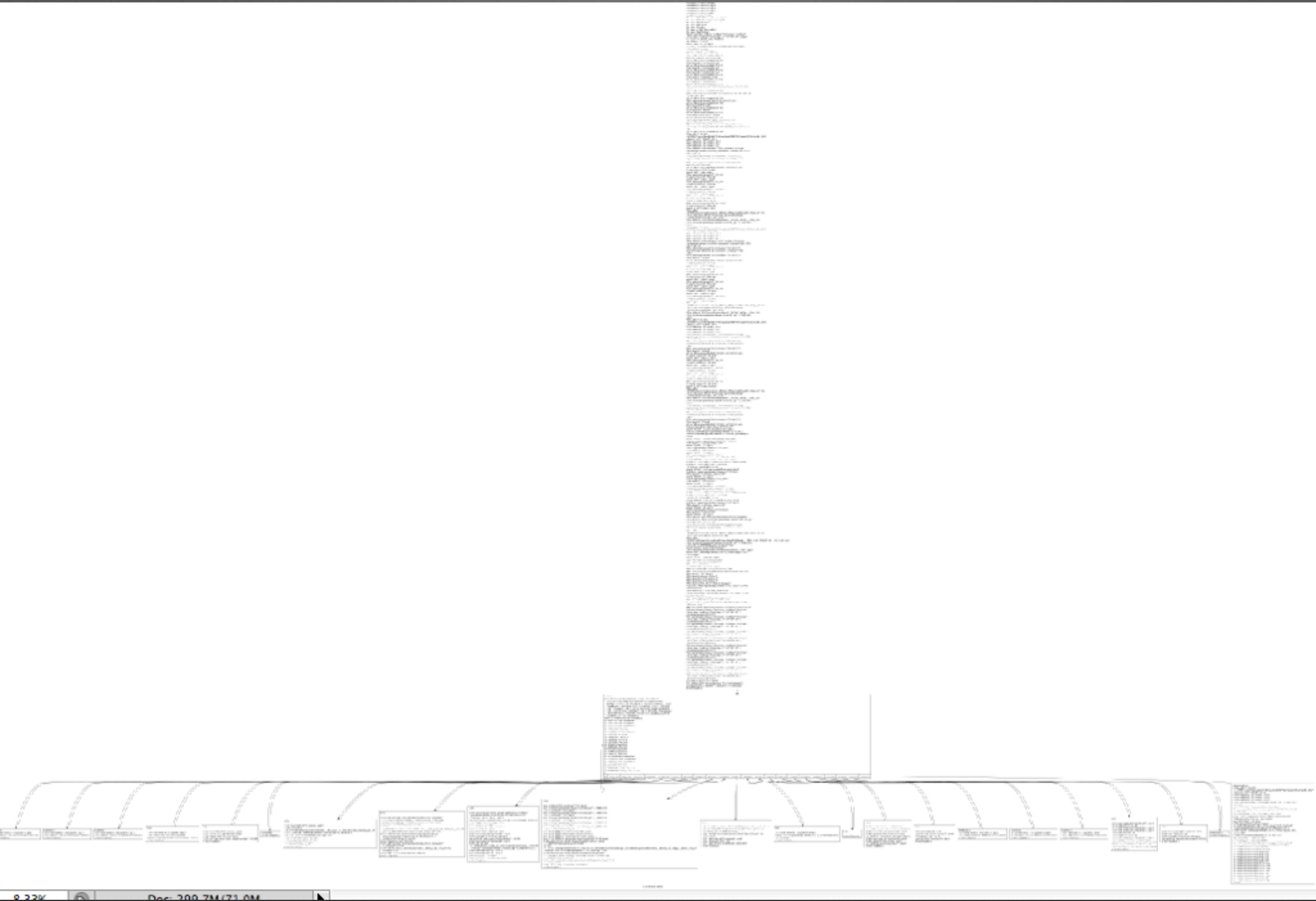

RC4 encryption
1x
Swift
220x
Objective-C
127x
Python
1x

```swift
//
//  main.swift
//  Created by Pascal Junod on 03.06.14.
//

import Foundation

let interestingNumbers = [
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0
for (kind, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}

println("Hello, World!")
println (largest)
```

```
@_Tv4main18interestingNumbersGVSs10DictionarySSGSaSi__ = global %VSs10Dictionar
 zeroinitializer, align 8
@_Tv4main7largestSi = global %Si zeroinitializer, align 8
@"_swift_FORCE_LOAD_$_swiftFoundation" = external global i1
@"_swift_FORCE_LOAD_$_swiftFoundation_$_main" = weak hidden constant i1* @"_swi
t_FORCE_LOAD_$_swiftFoundation"
@"_swift_FORCE_LOAD_$_swiftDarwin" = external global i1
@"_swift_FORCE_LOAD_$_swiftDarwin_$_main" = weak hidden constant i1* @"_swift_F
RCE_LOAD_$_swiftDarwin"
@"_swift_FORCE_LOAD_$_swiftObjectiveC" = external global i1
@"_swift_FORCE_LOAD_$_swiftObjectiveC_$_main" = weak hidden constant i1* @"_swi
t_FORCE_LOAD_$_swiftObjectiveC"
@"_swift_FORCE_LOAD_$_swiftDispatch" = external global i1
@"_swift_FORCE_LOAD_$_swiftDispatch_$_main" = weak hidden constant i1* @"_swift_
FORCE_LOAD_$_swiftDispatch"
@"_swift_FORCE_LOAD_$_swiftCoreGraphics" = external global i1
@"_swift_FORCE_LOAD_$_swiftCoreGraphics_$_main" = weak hidden constant i1* @"_s
ift_FORCE_LOAD_$_swiftCoreGraphics"
@0 = private unnamed_addr constant [6 x i16] [i16 80, i16 114, i16 105, i16 109
 i16 101, i16 0]
@metadata = internal constant %swift.full_heapmetadata { void (%swift.refcounte
*)* @arraydestroy, i8** null, %swift.type { i64 65 } }
@_TMdSi = external global %swift.full_type
```

Command Line Magics

- pjunod$ /Applications/Xcode6-Beta.app/Contents/./Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/swift -sdk /Applications/Xcode6-Beta.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk/ -emit-ir -o main.ir main.swift
- pjunod$ /Volumes/scratch/devel/build/bin/opt main.ir -o main_fla.ir -S -std-compile-opts -fla -O1



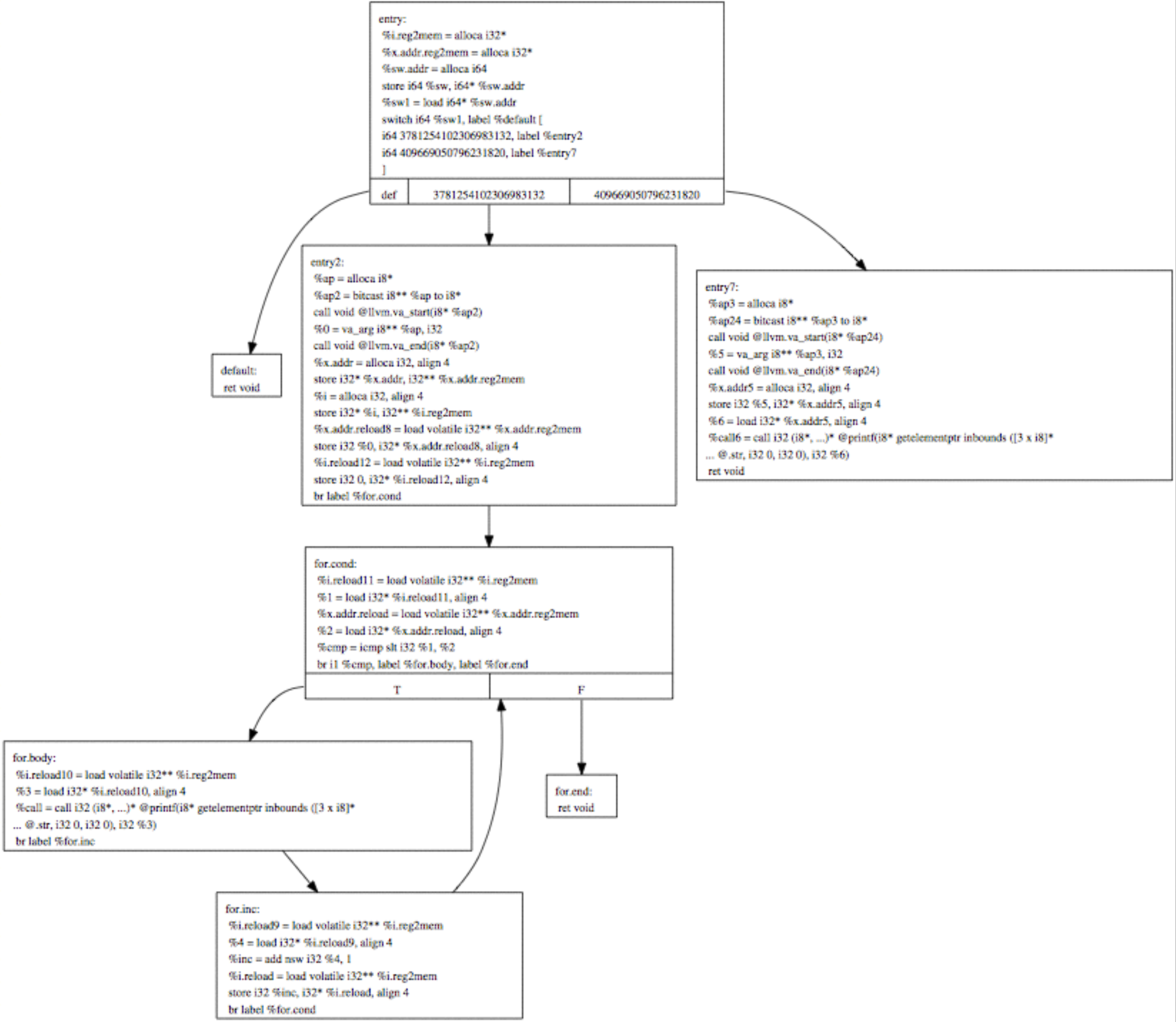
Code Flattening

- Routing variable generation
  - Weak: hard-coded
  - Better: dynamically generated using opaque constructs
  - Even better: depending on the inner state of the program (tamper-proofing)
- Coming soon: basic-block splitting before flattening

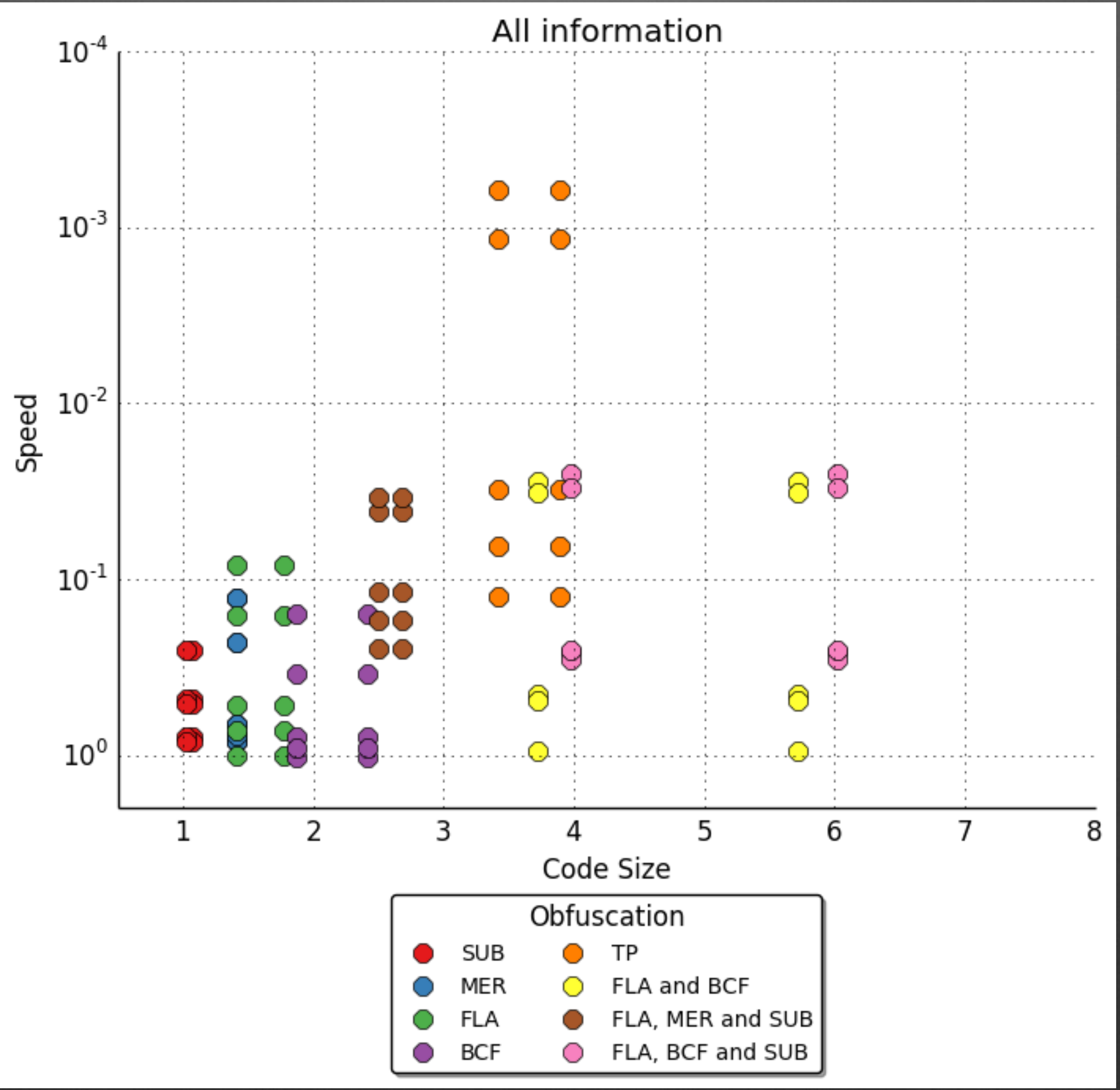
Procedures Merging

Procedure Merging

- In a compilation unit, put the code of all routines in a single one (that can be later flattened, etc.)
- Use the initial symbol as a wrapper to the huge routine
  - Responsible to handle parameters
- Reverse engineer has to figure out the signature of each function
- Not useful for exported APIs



CFG for 'merge1228995049' function


All information
Speed
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
1
2
3
4
5
6
7
8
Code Size
Obfuscation
SUB
MER
FLA
BCF
TP
FLA and BCF
FLA, MER and SUB
FLA, BCF and SUB

Current State of O-LLVM

- Published
  - Instruction substitutions
  - Bogus-control flow insertion
  - Basic code flattening

Current State of O-LLVM

- In testing phase
  - Procedure merging
  - Basic-block splitting
  - More resistant code flattening
  - Developer annotations

Current State of O-LLVM

- In implementation phase
  - Code tamper proofing (post-processor has to be re-written)

Current State of O-LLVM

- Foreseen / wished
  - Anti-debugging tricks insertion
  - Packing
  - Code virtualization
  - ???

Questions?

http://o-llvm.org
@ollvm