Cryptography is not just Encryption - Obfuscation

Autumn School of Algebra 2015

Martin Mach

21. 11. 2015

Structure

- Some examples
- Obfuscation under VBB security
- Weaker definitions
- Functional encryption

Why to Obfuscate Programs?

- Do not let anyone to copy our program/algorithm
- Hide a password written in the program

Why to Obfuscate Programs?

- Do not let anyone to copy our program/algorithm
- Hide a password written in the program

```
class Program
{
    static void SuperSecretPasswordProtectedStuff(string passwd)
    {
        if (passwd == "SuperSecretPassword")
        {
            Console.WriteLine("Congratulations. Here's some super secret private information: ....\n");
        }
        else
        {
            Console.WriteLine("Wrong password, fool.\n");
        }
    }
    static void Main(string[] args)
    {
        string passwd = Console.In.ReadLine();
        SuperSecretPasswordProtectedStuff(passwd);
    }
}
```

Why to Obfuscate Programs?

- Compiled program:

```
4D5A 9000 0300 0000 0400 0000 FFFF 0000  MZ .........ÿÿ..
B800 0000 0000 0000 4000 0000 0000 0000  ¸.......@.......
0000 0000 0000 0000 0000 0000 0000 0000  ................
0000 0000 0000 0000 0000 0000 8000 0000  ............€...
0E1F BA0E 00B4 09CD 21B8 014C CD21 5468  ..º..´.Í!¸.LÍ!Th
6973 2070 726F 6772 616D 2063 616E 6E6F  is program canno
7420 6265 2072 756E 2069 6E20 444F 5320  t be run in DOS
6D6F 6465 2E0D 0D0A 2400 0000 0000 0000  mode....$.......
5045 0000 4C01 0300 7B9C 4D56 0000 0000  PE..L...{œMV....
0000 0000 E000 2200 0B01 3000 000A 0000  ....à."...0.....
0008 0000 0000 0000 2629 0000 0020 0000  ........&)... ..
0040 0000 0000 4000 0020 0000 0002 0000  .@....@.. ......
0400 0000 0000 0000 0600 0000 0000 0000  ................
0080 0000 0002 0000 0000 0000 0300 6085  .€............`…
0000 1000 0010 0000 0000 1000 0010 0000  ................
0000 0000 1000 0000 0000 0000 0000 0000  ................
D328 0000 4F00 0000 0040 0000 BC05 0000  Ó(..O....@..¼...
0000 0000 0000 0000 0000 0000 0000 0000  ................
0060 0000 0C00 0000 2428 0000 1C00 0000  .`......$(......
```

Why to Obfuscate Programs?

- In the hexa code of the compiled program:

```
7369 6F6E 696E 6700 5374 7269 6E67 0050   sioning.String.P
726F 6772 616D 0053 7973 7465 6D00 6765   rogram.System.ge
745F 496E 004D 6169 6E00 4F62 6675 7363   t_In.Main.Obfusc
6174 696F 6E00 5379 7374 656D 2E52 6566   ation.System.Ref
6C65 6374 696F 6E00 5465 7874 5265 6164   lection.TextRead
6572 002E 6374 6F72 0053 7973 7465 6D2E   er..ctor.System.
4469 6167 6E6F 7374 6963 7300 5379 7374   Diagnostics.Syst
656D 2E52 756E 7469 6D65 2E49 6E74 6572   em.Runtime.Inter
6F70 5365 7276 6963 6573 0053 7973 7465   opServices.Syste
6D2E 5275 6E74 696D 652E 436F 6D70 696C   m.Runtime.Compil
6572 5365 7276 6963 6573 0044 6562 7567   erServices.Debug
6769 6E67 4D6F 6465 7300 6172 6773 004F   gingModes.args.O
626A 6563 7400 6F70 5F45 7175 616C 6974   bject.op_Equalit
7900 0000 0027 5300 7500 7000 6500 7200   y....'S.u.p.e.r.
5300 6500 6300 7200 6500 7400 5000 6100   S.e.c.r.e.t.P.a.
7300 7300 7700 6F00 7200 6400 0080 A943   s.s.w.o.r.d.€©C
```

Why to Obfuscate Programs?

- This can be fix by usage of a hash of the password

Why to Obfuscate Programs?

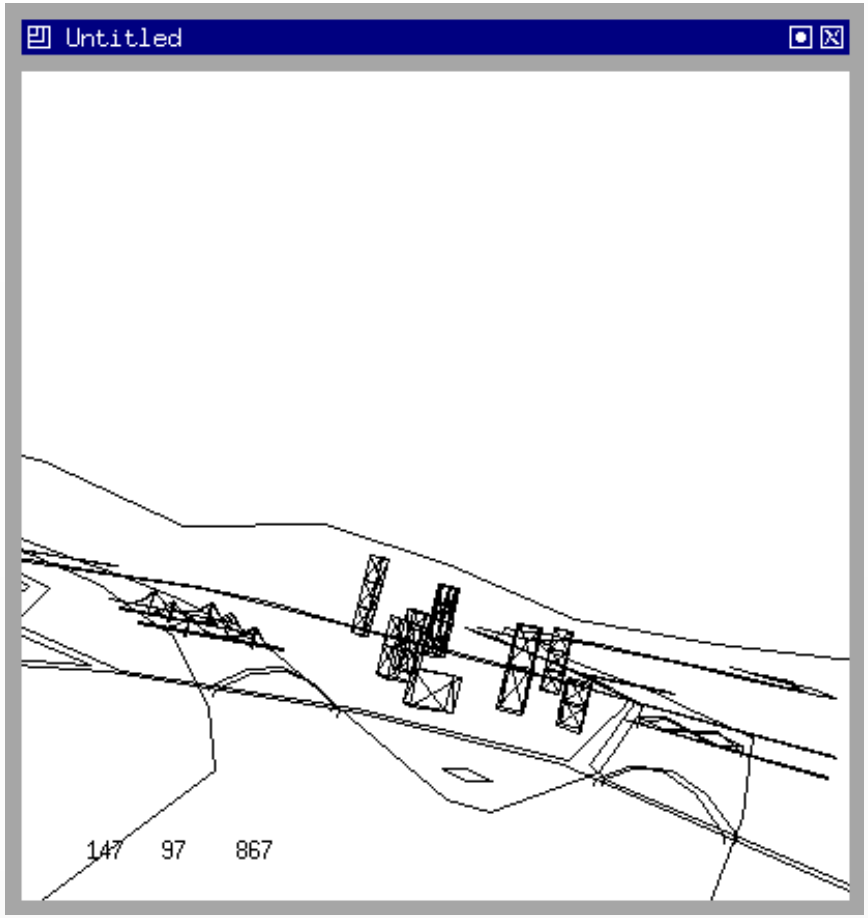
- This can be fix by usage of a hash of the password
- But there is still some work with obfuscation of this program:

```
7900 0000 0027 5300 7500 7000 6500 7200  y....'S.u.p.e.r.
5300 6500 6300 7200 6500 7400 5000 6100  S.e.c.r.e.t.P.a.
7300 7300 7700 6F00 7200 6400 0080 A943  s.s.w.o.r.d..€©C
006F 006E 0067 0072 0061 0074 0075 006C  .o.n.g.r.a.t.u.l
0061 0074 0069 006F 006E 0073 002E 0020  .a.t.i.o.n.s...
0048 0065 0072 0065 0027 0073 0020 0073  .H.e.r.e.'.s. .s
006F 006D 0065 0020 0073 0075 0070 0065  .o.m.e. .s.u.p.e
0072 0020 0073 0065 0063 0072 0065 0074  .r. .s.e.c.r.e.t
0020 0070 0072 0069 0076 0061 0074 0065  . .p.r.i.v.a.t.e
0020 0069 006E 0066 006F 0072 006D 0061  . .i.n.f.o.r.m.a
0074 0069 006F 006E 003A 0020 0042 004C  .t.i.o.n.:. .B.L
0041 0048 002C 0020 0042 004C 0041 0048  .A.H.,. .B.L.A.H
002C 0020 0042 004C 0041 0048 0020 002E  .,. .B.L.A.H. ..
002E 002E 000A 0001 2D57 0072 006F 006E  ........-W.r.o.n
0067 0020 0070 0061 0073 0073 0077 006F  .g. .p.a.s.s.w.o
0072 0064 002C 0020 0066 006F 006F 006C  .r.d.,. .f.o.o.l
002E 000A 0000 0000 97EA E105 35EE A241  ........—êá.5î¢A
971B 9068 E684 9340 0004 2001 0108 0320  —. hæ„"@.. ....
```

IOCCC

- The International Obfuscated C Code Contest
- Goal is to write the most Obscure/Obfuscated C program within the rules.
- Some exmaples:

- Fly simulator
- Less then 2 kilobytes of code
- Complete with relatively accurate 6-degree-of-freedom dynamics, loadable wireframe scenery, and a small instrument panel.



```
#include                                    <math.h>
#include                                   <sys/time.h>
#include                                   <X11/Xlib.h>
#include                                  <X11/keysym.h>
                                          double L ,o ,P
                                         ,_=dt,T,Z,D=1,d,
                                         s[999],E,h= 8,I,
                                         J,K,w[999],M,m,O
                                        ,n[999],j=33e-3,i=
                                        1E3,r,t, u,v ,W,S=
                                        74.5,l=221,X=7.26,
                                        a,B,A=32.2,c, F,H;
                                        int N,q, C, y,p,U;
                                       Window z; char f[52]
                                    ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                   XEvent z; XNextEvent(e ,&z);
                                       ++*((N=XLookupKeysym
                                         (&z.xkey,0))-IT?
                                         N-LT? UP-N?& E:&
                                         J:& u: &h); --*(
                                         DN -N? N-DT ?N==
                                         RT?&u: & W:&h:&J
                                          ); } m=15*F/l;
                                          c+=(I=M/ l,l*H
                                          +I*M+a*X)*_; H
                                          =A*r+v*X-F*l+(
                                          E=.1+X*4.9/l,t
                                          =T*m/32-I*T/24
                                           )/S; K=F*M+(
                                           h* 1e4/l-(T+
                                           E*5*T*E)/3e2
                                           )/S-X*d-B*A;
                                           a=2.63 /l*d;
                                           X+=( d*l-T/S
                                            *(.19*E +a
                                            *.64+J/1e3
                                            )-M* v +A*
                                            Z)*_; l +=
                                            K *_; W=d;
                                            sprintf(f,
                                            "%5d  %3d"
                                            "%7d",p =l
                                           /1.7,(C=9E3+
                                  O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                                X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                                *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
```

- Coder form ASCII to Morse alphabet and vice versa

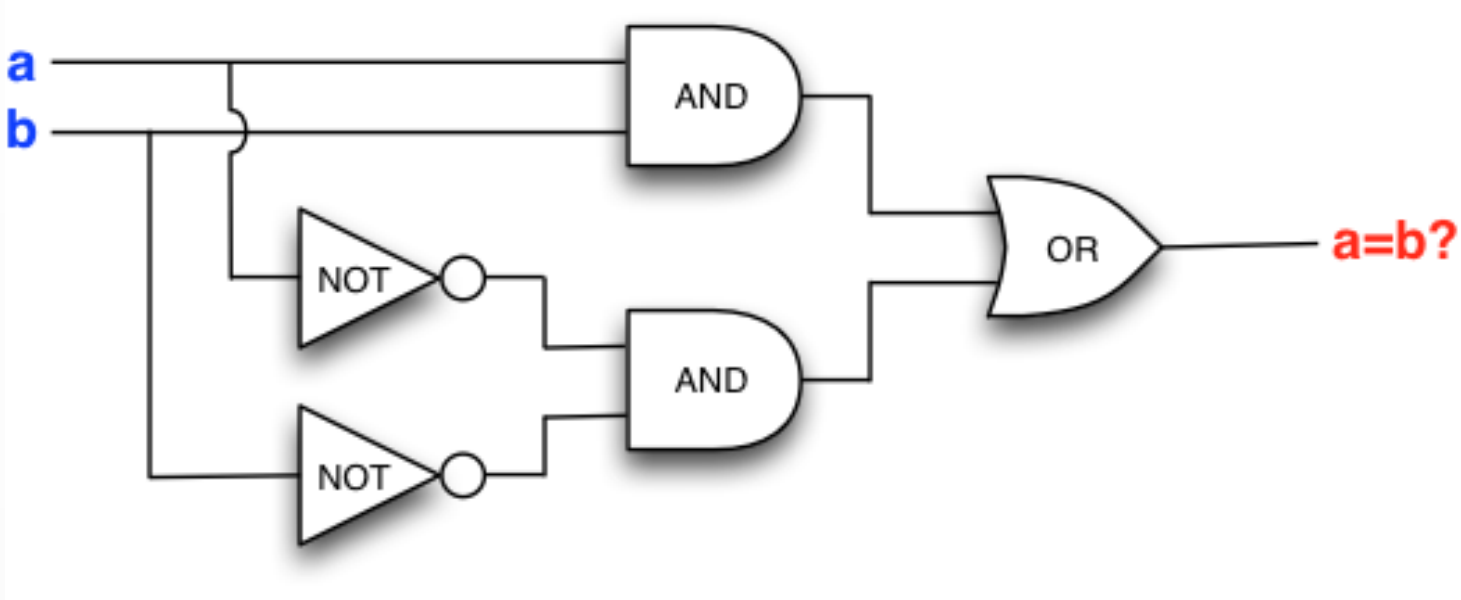
```
#include<stdio.h> #include<string.h> main(){char*o,l[999]=
"''acgo\177~|xp .-\OR^8)NJ6%K4O+A2M(*OID57$3G1FBL";while(o=
fgets(l+45,954,stdin)){*l=o[strlen(o)[o-1]=0,strspn(o,l+11)];
while(*o)switch((*l&&isalnum(*o))-!*l){case-1:{char*I=(o+=
strspn(o,l+12)+1)-2,o=34;while(*I&3&&(o=(o-16<<1)+*I---'-')<80);
putchar(o&93?*I&8||!( I=memchr( l , o , 44 ) ) ?'?':I-l+47:32);
break;case 1: ;}*l=(*o&31)[l-15+(*o>61)*32];while(putchar(45+*l%2),
(*l=*l+32>>1)>35);case 0:putchar((++o,32));}putchar(10);}}
```

Section 2

Obfuscation under VBB Security

Boolean Circuit

- Model of boolean function $f : \{0,1\}^n \to \{0,1\}^m$
- Composed of AND, OR and NOT gates



Virtual Black Box

- Program/object which is viewed only in terms of its inputs and outputs

Obfuscation under VBB Security

Definition (Obfuscator of circuits under Virtual Black Box security)

$\mathcal{O}$ is an *obfuscator of circuits* if

1. *Correctness:* $\forall c$ circuit , $\mathcal{O}(c) \equiv c$.
2. *Efficiency:* $\forall c$ circuit, $|\mathcal{O}(c)| \leq \text{poly}(|c|)$.
3. *VBB:* $\forall A$, A is bounded, $\exists S$ PPT simulator s.t. $\forall c$ circuit:

$$\left|\Pr\left[A(\mathcal{O}(c)) = 1\right] - \Pr\left[S^c\left(1^{|c|}\right) = 1\right]\right| \leq \mu\left(|c|\right),$$

where μ is a negligible function.

Obfuscation under VBB Security

Definition (Obfuscator of circuits under Virtual Black Box security)

$\mathcal{O}$ is an *obfuscator of circuits* if

1. *Correctness:* $\forall c$ circuit , $\mathcal{O}(c) \equiv c$.
2. *Efficiency:* $\forall c$ circuit, $|\mathcal{O}(c)| \leq \text{poly}(|c|)$.
3. *VBB:* $\forall A$, A is bounded, $\exists S$ PPT simulator s.t. $\forall c$ circuit:

$$\left|\Pr\left[A(\mathcal{O}(c)) = 1\right] - \Pr\left[S^c\left(1^{|c|}\right) = 1\right]\right| \leq \mu\left(|c|\right),$$

where μ is a negligible function.

- Having access to the obfuscaded program is the same as having access to the program only as black box.

Impossibility of Obfuscation under VBB Security

Theorem (Impossibility of obfuscation under VBB securit)

Obfuscators of circuits under VBB security do not exists.

Source: On the (Im)possibility of Obfuscating Programs, B. Barack and collective, 2001

Section 3

Weaker Definitions

Definition (Indistinguishability Obfuscator for circuits)

We call $i\mathcal{O}$ an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if

1. *Correctness:* $\forall \lambda \in \mathbb{N}$ security parametr, $\forall C \in \mathcal{C}_\lambda$, $\forall x$ input, we have that
$$\Pr\left[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)\right] = 1.$$
2. *Polynomial slowdown:* $\exists p$ polynom s.t. $\forall C \in \mathcal{C}_\lambda$, we have $|C'| \leq p(|C|)$, where $C' = i\mathcal{O}(\lambda, C)$.
3. *Computaitonal indistinguishability:* Suppose that $\forall x : C_0(x) = C_1(x)$ then for any PPT adversaries $Samp, D$, $\exists \mu$ a negligible function s.t.:
$$\Big|\Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right] -$$
$$-\Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right]\Big| \leq \mu(\lambda).$$

Indistinguishability Obfuscation

- Given two programs computing the same output for all inputs
- It is impossible distinguish between obfuscations of these programs

Differing-inputs Obfuscation

Definition (Differing-inputs Obfuscator for circuits)

We call $di\mathcal{O}$ a *Different-inputs obfuscator* for a different-inputs circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}$ if

1. *Correctness:* $\forall \lambda \in \mathbb{N}$ security parametr, $\forall C \in \mathcal{C}$, $\forall x$ input, we have that
$$\Pr\left[C'(x) = C(x) : C' \leftarrow di\mathcal{O}(\lambda, C)\right] = 1.$$
2. *Polynomial slowdown:* $\exists p$ polynom s.t. $\forall C \in \mathcal{C}$ circuit, we have $|C'| \leq p(|C|)$, where $C' = \mathrm{diO}(\lambda, C)$.
3. *Different-inputs:* For any PPT distinguisher D, for $(C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)$, $\exists \mu$ a negligible function s.t.: $\forall \lambda \in \mathbb{N}$ security parametr and $\forall x$ holds:
$\Pr\left[C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right] > 1 - \mu(\lambda)$. Then we have:
$$\left|\Pr\left[D(di\mathcal{O}(\lambda, C_0), \sigma) = 1\right] - \Pr\left[D(di\mathcal{O}(\lambda, C_1), \sigma) = 1\right]\right| \leq \mu(\lambda).$$

Differing-inputs Obfuscation

- Given two programs computing the same output for nearly each input
- It is hard to find the input where the programs differ
- Then it is also hard to distinguish between obfuscations of these programs

Differing-inputs Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $di\mathcal{O}$)

Every $di\mathcal{O}$ obfuscator is also a $i\mathcal{O}$ obfuscator.

Differing-inputs Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $di\mathcal{O}$)

Every $di\mathcal{O}$ obfuscator is also a $i\mathcal{O}$ obfuscator.

Proof.

- Definitions of $i\mathcal{O}$ and $di\mathcal{O}$ differs in assumption, which pairs of circuits C_0, C_1 are considered:

Differing-inputs Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $di\mathcal{O}$)

Every $di\mathcal{O}$ obfuscator is also a $i\mathcal{O}$ obfuscator.

Proof.

- Definitions of $i\mathcal{O}$ and $di\mathcal{O}$ differs in assumption, which pairs of circuits C_0, C_1 are considered:
  - $i\mathcal{O}$: $C_0(x) = C_1(x)$,

Differing-inputs Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $di\mathcal{O}$)

Every $di\mathcal{O}$ obfuscator is also a $i\mathcal{O}$ obfuscator.

Proof.

- Definitions of $i\mathcal{O}$ and $di\mathcal{O}$ differs in assumption, which pairs of circuits C_0, C_1 are considered:
  - $i\mathcal{O}$: $C_0(x) = C_1(x)$,
  - $di\mathcal{O}$: $\Pr[C_0(x) = C_1(x)] > 1 - \mu(\lambda)$ for λ security parametr, μ a negligible function.

Differing-inputs Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $di\mathcal{O}$)

Every $di\mathcal{O}$ obfuscator is also a $i\mathcal{O}$ obfuscator.

Proof.

- Definitions of $i\mathcal{O}$ and $di\mathcal{O}$ differs in assumption, which pairs of circuits C_0, C_1 are considered:
  - $i\mathcal{O}$: $C_0(x) = C_1(x)$,
  - $di\mathcal{O}$: $\Pr[C_0(x) = C_1(x)] > 1 - \mu(\lambda)$ for λ security parametr, μ a negligible function.
- If $C_0(x) = C_1(x)$ then the condition for $di\mathcal{O}$ is fulfilled.

□

Best-possible Obfuscation

Definition (Best-possible Obfuscator for circuits)

We call $b\mathcal{O}$ a *Best-possible obfuscator* for a circuit class $\mathcal{C}$ if

1. *Correctness:* $\forall \lambda \in \mathbb{N}$ security parametr, $\forall C \in \mathcal{C}$, $\forall x$ input, we have that

$$\Pr\left[C'(x) = C(x) : C' \leftarrow b\mathcal{O}(\lambda, C)\right] = 1.$$

2. *Polynomial slowdown:* $\exists p$ polynom s.t. $\forall C \in \mathcal{C}$ circuit, we have $|C'| \leq p(|C|)$, where $C' = b\mathcal{O}(\lambda, C)$.
3. *Best-possible:* For any PPT learner L, $\exists S$ simulator s.t.: for $(C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)$, $|C_0| \equiv |C_1|$, for every input x is $C_0(x) = C_1(x)$. Then:

$$L(b\mathcal{O}(C_0)) \stackrel{c}{\equiv} S(C_1),$$

where $A \stackrel{c}{\equiv} B$ means that two distributions A and B are computationally indistinguishable.

Best-possible Obfuscation

- Given program C_0
- Let C_1 be any program of similar size computing the same function as C_0
- Everything what can be extracted from obfuscated $b\mathcal{O}(C_0)$ can be also extracted from C_1

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - Consider an empty learner L who outputs whatewer obfuscation is given

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - Consider an empty learner L who outputs whatewer obfuscation is given
  - S is a PPT simulator

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - Consider an empty learner L who outputs whatewer obfuscation is given
  - S is a PPT simulator
  - Because $\mathcal{O}$ je Best-possible obfuscator, then $L(\mathcal{O}(C_0)) \stackrel{c}{\equiv} S(C_1)$, so also $\mathcal{O}(C_0) \stackrel{c}{\equiv} S(C_1)$

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - Consider an empty learner L who outputs whatewer obfuscation is given
  - S is a PPT simulator
  - Because $\mathcal{O}$ je Best-possible obfuscator, then $L(\mathcal{O}(C_0)) \stackrel{c}{\equiv} S(C_1)$, so also $\mathcal{O}(C_0) \stackrel{c}{\equiv} S(C_1)$
  - Also $L(\mathcal{O}(C_1)) \stackrel{c}{\equiv} S(C_1)$ and $\mathcal{O}(C_1) \stackrel{c}{\equiv} S(C_1)$

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - Consider an empty learner L who outputs whatewer obfuscation is given
  - S is a PPT simulator
  - Because $\mathcal{O}$ je Best-possible obfuscator, then $L(\mathcal{O}(C_0)) \stackrel{c}{\equiv} S(C_1)$, so also $\mathcal{O}(C_0) \stackrel{c}{\equiv} S(C_1)$
  - Also $L(\mathcal{O}(C_1)) \stackrel{c}{\equiv} S(C_1)$ and $\mathcal{O}(C_1) \stackrel{c}{\equiv} S(C_1)$
  - Computational indistinguishability is an equivalence and we have $\mathcal{O}(C_0) \stackrel{c}{\equiv} \mathcal{O}(C_1)$

Best-possible Obfuscation

Theorem (Relation between $i\mathcal{O}$ and $b\mathcal{O}$)

The obfuscator $\mathcal{O}$ is a $b\mathcal{O}$ obfuscator if and only if it is an $i\mathcal{O}$ obfuscator.

Proof.

- $b\mathcal{O} \Rightarrow i\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - Consider an empty learner L who outputs whatewer obfuscation is given
  - S is a PPT simulator
  - Because $\mathcal{O}$ je Best-possible obfuscator, then $L(\mathcal{O}(C_0)) \stackrel{c}{\equiv} S(C_1)$, so also $\mathcal{O}(C_0) \stackrel{c}{\equiv} S(C_1)$
  - Also $L(\mathcal{O}(C_1)) \stackrel{c}{\equiv} S(C_1)$ and $\mathcal{O}(C_1) \stackrel{c}{\equiv} S(C_1)$
  - Computational indistinguishability is an equivalence and we have $\mathcal{O}(C_0) \stackrel{c}{\equiv} \mathcal{O}(C_1)$
  - Thanks to it there is not any PPT adversary D who can distinguish between obfuscations of C_0 and C_1

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - $i\mathcal{O}$ is indistinguishability obfuscator

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - $i\mathcal{O}$ is indistinguishability obfuscator
  - For every PPT learner L let S be PPT simulator that gets C_1, runs $i\mathcal{O}(C_1)$ and activates L on $i\mathcal{O}(C_1)$

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - $i\mathcal{O}$ is indistinguishability obfuscator
  - For every PPT learner L let S be PPT simulator that gets C_1, runs $i\mathcal{O}(C_1)$ and activates L on $i\mathcal{O}(C_1)$
  - From an indistinguishability assumption holds $L(i\mathcal{O}(C_0)) \stackrel{c}{\equiv} L(i\mathcal{O}(C_1)) = S(C_1)$

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - $i\mathcal{O}$ is indistinguishability obfuscator
  - For every PPT learner L let S be PPT simulator that gets C_1, runs $i\mathcal{O}(C_1)$ and activates L on $i\mathcal{O}(C_1)$
  - From an indistinguishability assumption holds $L(i\mathcal{O}(C_0)) \stackrel{c}{\equiv} L(i\mathcal{O}(C_1)) = S(C_1)$
  - Thanks to it is $i\mathcal{O}$ a Best-possible obfuscator

Best-possible Obfuscation

Proof - next part.

- $i\mathcal{O} \Rightarrow b\mathcal{O}$
  - C_0, C_1 two programs of similar size computing the same functionality
  - $i\mathcal{O}$ is indistinguishability obfuscator
  - For every PPT learner L let S be PPT simulator that gets C_1, runs $i\mathcal{O}(C_1)$ and activates L on $i\mathcal{O}(C_1)$
  - From an indistinguishability assumption holds $L(i\mathcal{O}(C_0)) \stackrel{c}{\equiv} L(i\mathcal{O}(C_1)) = S(C_1)$
  - Thanks to it is $i\mathcal{O}$ a Best-possible obfuscator

□

Virtual Grey Box Obfuscation

Definition (Virtual Grey Box Ofuscator for circuits)

We call $\mathcal{O}_{vgb}$ a *Virtual Gray Box Obfuscator* for a circuit class $\mathcal{C}$ if

1. *Corretness:* $\forall \lambda \in \mathbb{N}$ security parametr, $\forall C \in \mathcal{C}$, $\forall x$ input, we have that
$$\Pr\left[C'(x) = C(x) : C' \leftarrow \mathcal{O}_{vgb}(\lambda, C)\right] = 1.$$
2. *Polynomial slowdown:* $\exists p$ polynom s.t. $\forall C \in \mathcal{C}$ circuit, we have $|C'| \leq p(|C|)$, where $C' = \mathcal{O}_{vgb}(\lambda, C)$.
3. *Virtual Grey Box:* For every PPT adversary A, every predicate $\pi : \mathcal{C} \to \{0,1\}$, $\lambda = |C|$ security parameter, there $\exists S$ an unbounded simulator, $q(\cdot)$ polynom and a negligible function μ s.t.:
$$\left|\Pr\left[A(\mathcal{O}_{vgb}(C)) = \pi(C)\right] - \Pr\left[S^{C_{[q(\lambda)]}}(1^\lambda) = \pi(C)\right]\right| \leq \mu(\lambda),$$
where $C_{[q(\lambda)]}$ is an oracle that allows at most $q(n)$ queries.

Virtual Grey Box Obfuscation

- Relaxes the definition of VBB Obfuscator
- Simulator S is unbounded
- We still have a bounded number of oracle queries to C.

Virtual Grey Box Obfuscation

Theorem (Relation between $di\mathcal{O}$ and $\mathcal{O}_{vgb}$)

A program $\mathcal{O}$ is a $di\mathcal{O}$ if and only if it's a $\mathcal{O}_{vgb}$ obfuscator.

Source: On Virtual Grey Box Obfuscation for General Circuits, N. Bitansky, R.Canetti, Y. T. Kalai and O. Paneth, 2014

Section 4

Usage of Obfuscation

Functionality

Definition (Functionality)

A *functionality* F defined over (K, X) is a function $F\colon K \times X \to \{0,1\}^*$. The set K is called the *key space*, the set X is called the *plaintext space*. The space key K contains a special key called the *empty key* denoted ϵ.

Functionality

Definition (Functionality)

A *functionality* F defined over (K, X) is a function $F\colon K \times X \to \{0,1\}^*$. The set K is called the *key space*, the set X is called the *plaintext space*. The space key K contains a special key called the *empty key* denoted ϵ.

- K is a set of functions on X

Functionality

Definition (Functionality)

A *functionality* F defined over (K, X) is a function $F\colon K \times X \to \{0,1\}^*$. The set K is called the *key space*, the set X is called the *plaintext space*. The space key K contains a special key called the *empty key* denoted ϵ.

- K is a set of functions on X
- Functionality F describes the functions of a plaintext that can be learned from ciphertext

Functional Encryption

Definition (Functional encryption scheme)

A *functional encryption scheme* FE for a functionality F defined over (K, X) is a tuple of four PPT algorithms (Setup, Key, Encrypt, Decrypt) satysfying:

1. *Correctness:* $\forall k \in K, \forall x \in X$:
   - *generate a public and master key pair:* $(pk, mk) \leftarrow \text{Setup}(1^\lambda)$,
   - *generate secret key for* k*:* $sk_k \leftarrow \text{Key}(mk, k)$,
   - *encrypt message* x*:* $c \leftarrow \text{Encrypt}(pk, x)$,
   - *compute functionality from* c*:* $y \leftarrow \text{Decrypt}(sk_k, c)$,

   Then we require: $\Pr[y = F(k, x)] = 1$.

Functional Encryption

Definition (Functional encryption scheme)

A *functional encryption scheme* FE for a functionality F defined over (K, X) is a tuple of four PPT algorithms (Setup, Key, Encrypt, Decrypt) satysfying:

1. *Correctness:* $\forall k \in K, \forall x \in X$:
   - *generate a public and master key pair:* $(pk, mk) \leftarrow \text{Setup}(1^\lambda)$,
   - *generate secret key for* k*:* $sk_k \leftarrow \text{Key}(mk, k)$,
   - *encrypt message* x*:* $c \leftarrow \text{Encrypt}(pk, x)$,
   - *compute functionality from* c*:* $y \leftarrow \text{Decrypt}(sk_k, c)$,

   Then we require: $\Pr[y = F(k, x)] = 1$.

- A functional encryption enables to evaluate $F(k, x)$ given the encryption of x and secret key sk_k for k.

Identity Based Encryption

- Allows every party to generate a public key from known identity value string
- Trusted authority choose it's master private and master public key
- Trusted third party generates private keys for individual users public keys

Identity Based Encryption

- We can imagine IBE as evaluation of function on message $y = E(Id, x)$
- Function returns x only if identity Id is allowed to decrypt, otherwise halts

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:
  - We generate (pk, mk)

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:
  - We generate (pk, mk)
  - Attacker A declares two inputs x_1, x_2

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:
  - We generate (pk, mk)
  - Attacker A declares two inputs x_1, x_2
  - Attacker is given a secret key sk_k for k such that $F(k, x_1) = F(k, x_2)$ for all required k

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:
  - We generate (pk, mk)
  - Attacker A declares two inputs x_1, x_2
  - Attacker is given a secret key sk_k for k such that $F(k, x_1) = F(k, x_2)$ for all required k
  - A is given cyphertext $c_b = \text{Encrypt}(pk, m_b)$, $b \in \{0, 1\}$

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:
  - We generate (pk, mk)
  - Attacker A declares two inputs x_1, x_2
  - Attacker is given a secret key sk_k for k such that $F(k, x_1) = F(k, x_2)$ for all required k
  - A is given cyphertext $c_b = \text{Encrypt}(pk, m_b)$, $b \in \{0, 1\}$
  - A is outputs $b' \in \{0, 1\}$

Security Definition of Functional Eencryption

- We say that Functional encryption scheme is indistinguishability secure, if the following experiment holds true:
  - We generate (pk, mk)
  - Attacker A declares two inputs x_1, x_2
  - Attacker is given a secret key sk_k for k such that $F(k, x_1) = F(k, x_2)$ for all required k
  - A is given cyphertext $c_b = \text{Encrypt}(pk, m_b)$, $b \in \{0, 1\}$
  - A is outputs $b' \in \{0, 1\}$
  - There is μ a negligible fuction s.t. $\Pr[b = b'] \leq \frac{1}{2} + \mu(\lambda)$

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator
- $(\text{Setup}_P, \text{Encrypt}_P, \text{Eval}_P, \text{Decrypt}_P)$ is a public-key encryption scheme

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator
- $(\text{Setup}_P, \text{Encrypt}_P, \text{Eval}_P, \text{Decrypt}_P)$ is a public-key encryption scheme
- Encryption of a value x will be an encryption of x using the public key pk_P from the public-key encryption scheme

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator
- $(\text{Setup}_P, \text{Encrypt}_P, \text{Eval}_P, \text{Decrypt}_P)$ is a public-key encryption scheme
- Encryption of a value x will be an encryption of x using the public key pk_P from the public-key encryption scheme
- C is a circuit which is obfuscation of a program that uses sk_P to decrypt x

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator
- $(\text{Setup}_P, \text{Encrypt}_P, \text{Eval}_P, \text{Decrypt}_P)$ is a public-key encryption scheme
- Encryption of a value x will be an encryption of x using the public key pk_P from the public-key encryption scheme
- C is a circuit which is obfuscation of a program that uses sk_P to decrypt x
- A secret key for functional encryption is sk_k which outputs $C(x)$

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator
- $(\text{Setup}_P, \text{Encrypt}_P, \text{Eval}_P, \text{Decrypt}_P)$ is a public-key encryption scheme
- Encryption of a value x will be an encryption of x using the public key pk_P from the public-key encryption scheme
- C is a circuit which is obfuscation of a program that uses sk_P to decrypt x
- A secret key for functional encryption is sk_k which outputs $C(x)$
- This works for a Black-Box obfuscator

Functional Encryption from Indistinguishability Obfuscation

- Let's construct scheme for functional encryption
- $i\mathcal{O}$ is an indistinguishability obfuscator
- $(\text{Setup}_P, \text{Encrypt}_P, \text{Eval}_P, \text{Decrypt}_P)$ is a public-key encryption scheme
- Encryption of a value x will be an encryption of x using the public key pk_P from the public-key encryption scheme
- C is a circuit which is obfuscation of a program that uses sk_P to decrypt x
- A secret key for functional encryption is sk_k which outputs $C(x)$
- This works for a Black-Box obfuscator
- An Indistinguishability obfuscator can leak sk_P

Functional Encryption from Indistinguishability Obfuscation

- We fix the problem with generating two public keys pk_P^1, pk_P^2

Functional Encryption from Indistinguishability Obfuscation

- We fix the problem with generating two public keys pk_P^1, pk_P^2
- We require that the encryption of x consists of encryptions of x under both keys, let's name them e_1, e_2

Functional Encryption from Indistinguishability Obfuscation

- We fix the problem with generating two public keys pk_P^1, pk_P^2
- We require that the encryption of x consists of encryptions of x under both keys, let's name them e_1, e_2
- Receiver is not able to check that both cyphertexts encrypt the same message

Functional Encryption from Indistinguishability Obfuscation

- We fix the problem with generating two public keys pk_P^1, pk_P^2
- We require that the encryption of x consists of encryptions of x under both keys, let's name them e_1, e_2
- Receiver is not able to check that both cyphertexts encrypt the same message
- Encryptor generates non-interactive zero-knowledge proof that both cyphertexts encrypt the same message

Functional Encryption from Indistinguishability Obfuscation

- We fix the problem with generating two public keys pk_P^1, pk_P^2
- We require that the encryption of x consists of encryptions of x under both keys, let's name them e_1, e_2
- Receiver is not able to check that both cyphertexts encrypt the same message
- Encryptor generates non-interactive zero-knowledge proof that both cyphertexts encrypt the same message
- Our NIZK system consists of three algorithms ($\text{Setup}_{\text{NIZK}}$, $\text{Prove}_{\text{NIZK}}$, $\text{Verify}_{\text{NIZK}}$)

Functional Encryption from Indistinguishability Obfuscation

- We fix the problem with generating two public keys pk_P^1, pk_P^2
- We require that the encryption of x consists of encryptions of x under both keys, let's name them e_1, e_2
- Receiver is not able to check that both cyphertexts encrypt the same message
- Encryptor generates non-interactive zero-knowledge proof that both cyphertexts encrypt the same message
- Our NIZK system consists of three algorithms ($\text{Setup}_{\text{NIZK}}$, $\text{Prove}_{\text{NIZK}}$, $\text{Verify}_{\text{NIZK}}$)
- Obfuscated circuit C first checks the NIZK proof, then uses secret key sk_P^1 to decrypt e_1 to x

Functional Encryption from Indistinguishability Obfuscation

- Now we construct another program, which checks the NIZK proof then uses secret key sk_P^2 to decrypt e_2 to x

Functional Encryption from Indistinguishability Obfuscation

- Now we construct another program, which checks the NIZK proof then uses secret key sk_P^2 to decrypt e_2 to x
- Circuit C' is an obfuscation of this program

Functional Encryption from Indistinguishability Obfuscation

- Now we construct another program, which checks the NIZK proof then uses secret key sk_P^2 to decrypt e_2 to x
- Circuit C' is an obfuscation of this program
- Using an indistinguishability obfuscator, we have $C \overset{c}{\equiv} C'$

Functional Encryption from Indistinguishability Obfuscation

- Now we construct another program, which checks the NIZK proof then uses secret key sk_P^2 to decrypt e_2 to x
- Circuit C' is an obfuscation of this program
- Using an indistinguishability obfuscator, we have $C \overset{c}{\equiv} C'$
- C' computes the same as C but can't leak sk_P^1

Functional Encryption from Indistinguishability Obfuscation

Schema (Functional Encryption from $i\mathcal{O}$)

- Input : $k \in K, x \in X$
- Setup(1^λ) *algorithm:*
  - *Generate* $(pk_P^1, sk_P^1) \leftarrow \text{Setup}_P(1^\lambda)$
  - *Generate* $(pk_P^2, sk_P^2) \leftarrow \text{Setup}_P(1^\lambda)$
  - *Set* $mk = sk_P^1$
  - *Set* CRS $\leftarrow \text{Setup}_{\text{NIZK}}$ *are data needed for the* NIZK *proof*
  - *Set* PP $= \{pk_P^1, pk_P^2, \text{CRS}\}$ *as a public parametrs*
- Encrypt(PP, x) *algorithm:*
  - *Output* $c = (e_1, e_2, \pi)$, *where* $e_1 = \text{Encrypt}(pk_P^1, x, \text{CRS})$, $e_2 = \text{Encrypt}(pk_P^2, x, \text{CRS})$
  - π *is a* NIZK *proof that both* e_1, e_2 *encrypt the same message*

Functional Encryption from Indistinguishability Obfuscation

Schema (Functional Encryption from $i\mathcal{O}$ - Second part)

- Key(mk, k) *algorithm:*
  - *Output a secret key* sk_k*. It is a program which we get as an obfuscation of program* P_1 *which decrypts* e_1 *using* sk_P^1
  - *Secret key* $sk_k = i\mathcal{O}(P_1(e_1, e_2, \pi))$*, where* P_1 *outputs* $F(k, \text{Decrypt}_P(sk_P^1, e_1))$ *using* k, sk_P^1, CRS
- Decrypt$(sk_k, c = (e_1, e_2, \pi))$ *algorithm:*
  - *Outputs the output of obfuscated program* sk_k *on input* $c = (e_1, e_2, \pi)$
  - *Output is* $F(k, \text{Decrypt}_P(sk_P^1, e_1))$

Thank you for your attention!