What to Obfuscate When You're Obfuscating

Lecture 24

What is Obfuscation?

- **Program obfuscation** is a way to make a program unintelligible while *preserving functionality*.

What is Obfuscation?

- **Program obfuscation** is a way to make a program unintelligible while *preserving functionality*.

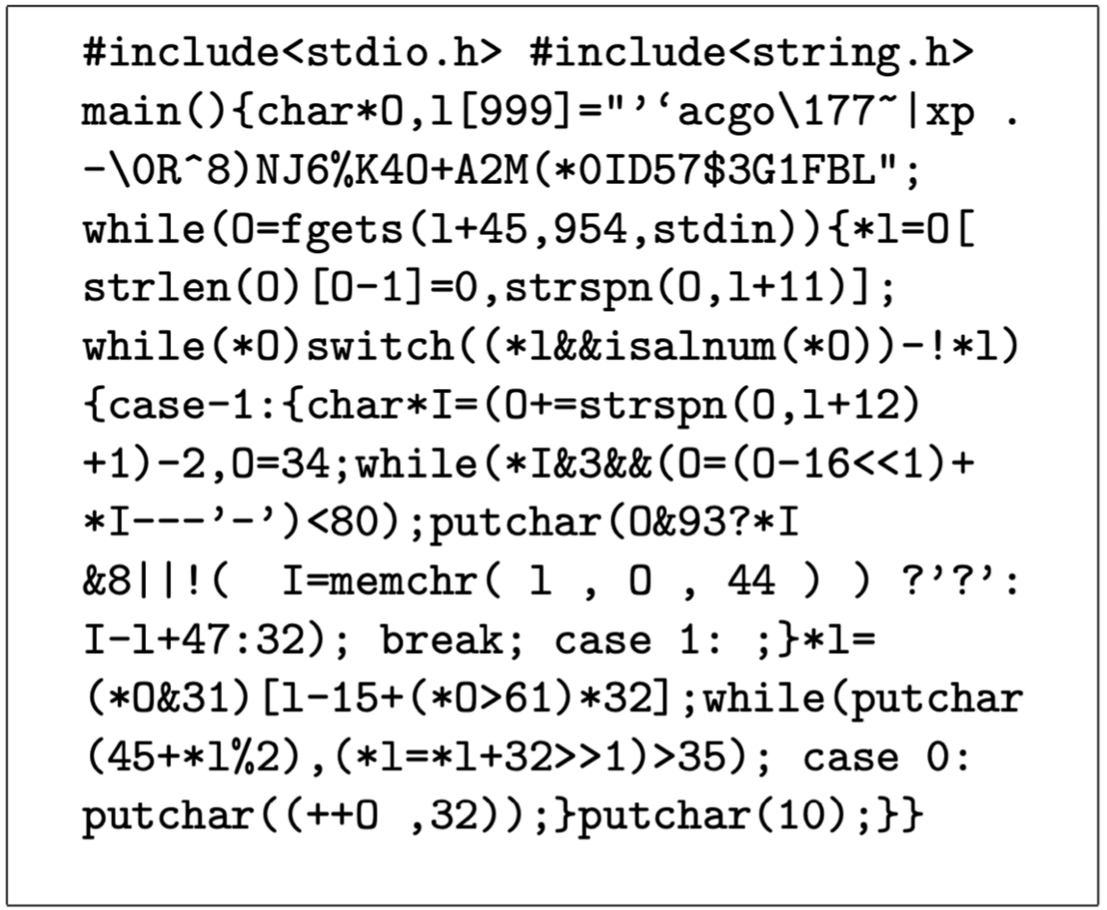
```
pi pi pi pi pi pi pi pi pi pi pika pipi pi pi pi pi pi pi pi pipi pi pi
pi pi pi pi pi pi pi pi pipi pi pi pi pipi pi pichu pichu pichu pichu ka
chu pipi pi pi pikachu pipi pi pikachu pi pi pi pi pi pi pi pikachu
pikachu pi pi pi pikachu pipi pi pi pikachu pichu pichu pi pi pi pi pi
pi pi pi pi pi pi pi pi pi pi pikachu pipi pikachu pi pi pi pikachu ka
ka ka ka ka ka pikachu ka ka ka ka ka ka ka ka pikachu pipi pi pikachu
pipi pikachu
```

What is Obfuscation?

- **Program obfuscation** is a way to make a program unintelligible while *preserving functionality*.

Hello, world! program

```
pi pi pi pi pi pi pi pi pi pi pika pipi pi pi pi pi pi pi pi pipi pi pi
pi pi pi pi pi pi pi pi pipi pi pi pi pipi pi pichu pichu pichu pichu ka
chu pipi pi pi pikachu pipi pi pikachu pi pi pi pi pi pi pi pikachu
pikachu pi pi pi pikachu pipi pi pi pikachu pichu pichu pi pi pi pi pi
pi pi pi pi pi pi pi pi pi pi pikachu pipi pikachu pi pi pi pikachu ka
ka ka ka ka ka pikachu ka ka ka ka ka ka ka ka pikachu pipi pi pikachu
pipi pikachu
```

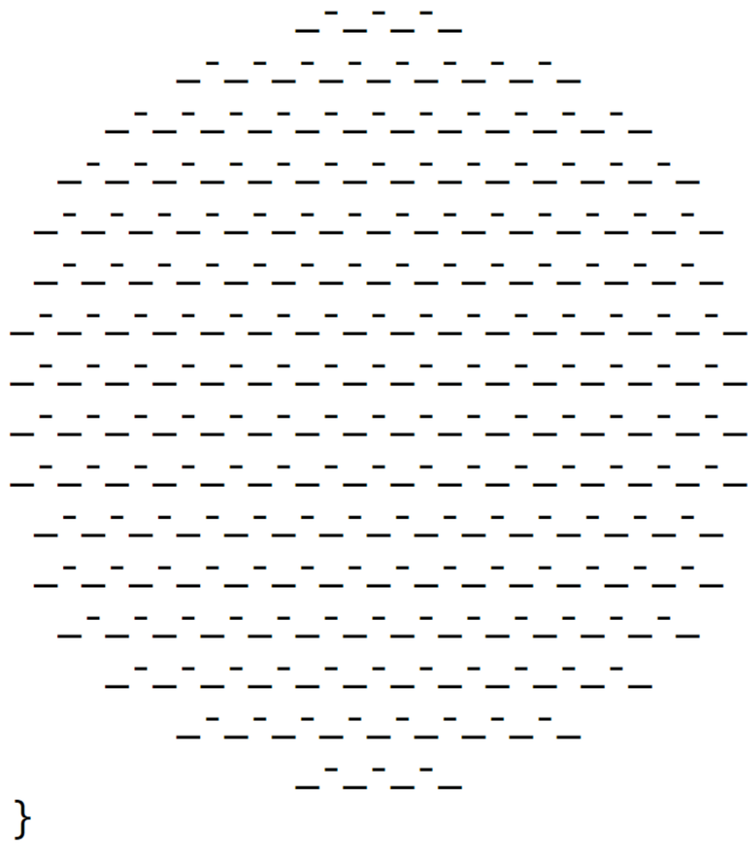


```
#include<stdio.h> #include<string.h>
main(){char*O,l[999]="’‘acgo\177~|xp .
-\OR^8)NJ6%K4O+A2M(*OID57$3G1FBL";
while(O=fgets(l+45,954,stdin)){*l=O[
strlen(O)[O-1]=0,strspn(O,l+11)];
while(*O)switch((*l&&isalnum(*O))-!*l)
{case-1:{char*I=(O+=strspn(O,l+12)
+1)-2,O=34;while(*I&3&&(O=(O-16<<1)+
*I---’-’)<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);}}
```

Figure 1: The winning entry of the 1998 *International Obfuscated C Code Contest*, an ASCII/Morse code translator by Frans van Dorsselaer [vD] (adapted for this paper).

```
#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*
                               179*v)/2312; select(p=0,0,0,0,&G); v-=(
                                W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                                 )/107e2)*_; D=cos(o); E=sin(o); } }
```

```
#define _ F-->00 || F-OO--;
long F=00,OO=00;
main(){F_OO();printf("%1.3f\n", 4.*-F/OO/OO);}F_OO()
{
```

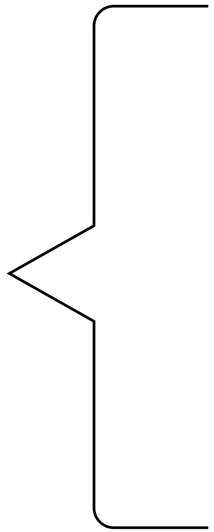


```
}
```

Answer: Run me!

Program Obfuscation

Programs that contain secrets:



Program Obfuscation

Programs that contain secrets:

- Cryptographic keys

Program Obfuscation

Programs that contain secrets:

- Cryptographic keys
- Watermarks

Program Obfuscation

Programs that contain secrets:

- Cryptographic keys
- Watermarks
- Trapdoors

Program Obfuscation

Programs that contain secrets:

- Cryptographic keys
- Watermarks
- Trapdoors
- The Algorithm itself

Hiding secrets

Hiding secrets

Hiding secrets

Hmm… I am going out of town but would like to delegate all 6.875 emails to my TAs.

Hiding secrets

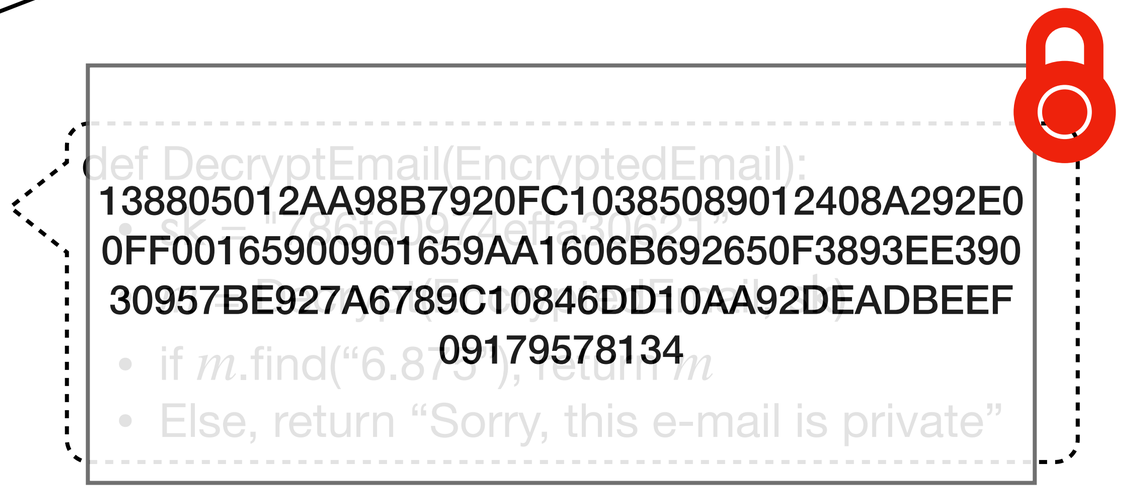
Hmm… I am going out of town but would like to delegate all 6.875 emails to my TAs.

def DecryptEmail(EncryptedEmail):

- sk = "786fe0974effa30621”
- m = Decrypt(EncryptedEmail, sk)
- if m.find(“6.875”), return m
- Else, return “Sorry, this e-mail is private”

Hiding secrets

Hmm… I am going out of town but would like to delegate all 6.875 emails to my TAs.



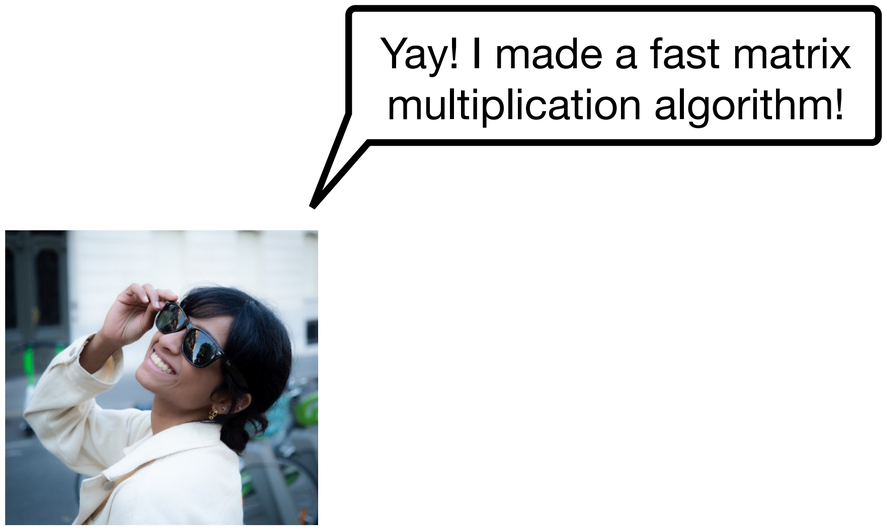
Hiding secrets

Hmm… I am going out of town but would like to delegate all 6.875 emails to my TAs.

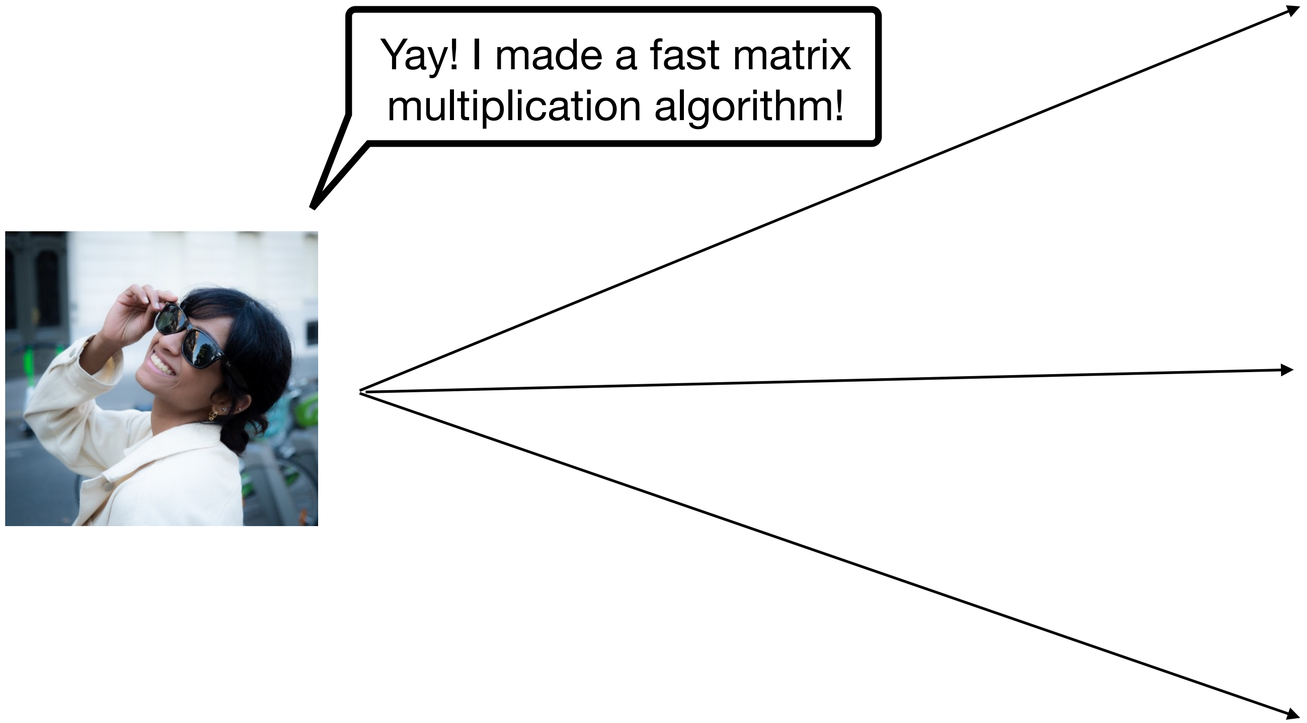
```
def DecryptEmail(EncryptedEmail):
    sk = "786fe0974effa3b621"
    m = Decrypt(sk, EncryptedEmail)
    if m.find("6.875"), return m
    Else, return "Sorry, this e-mail is private"
```

138805012AA98B7920FC10385089012408A292E0
0FF00165900901659AA1606B692650F3893EE390
30957BE927A6789C10846DD10AA92DEADBEEF
09179578134

Watermarking



Watermarking

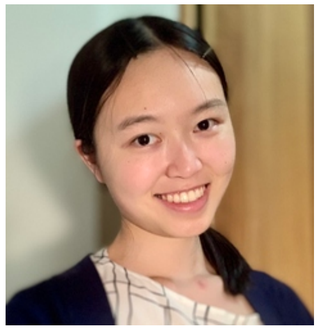


Watermarking

Yay! I made a fast matrix multiplication algorithm!

FMM
Author: Surya
Customer: Vinod

FMM
Author: Surya
Customer: Tina



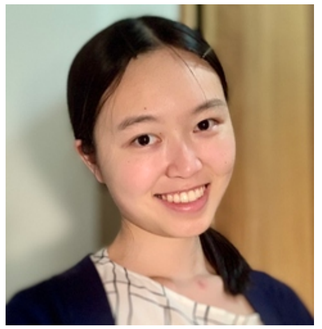
FMM
Author: Surya
Customer: Matt

Watermarking

Yay! I made a fast matrix multiplication algorithm!

3bcc4baa285258a4
242c4bd5092108fa
8ac7460be9a97706

6bf31c3e5aa0e434
46f98ceb880d6750
0a7be5d9807d11cf



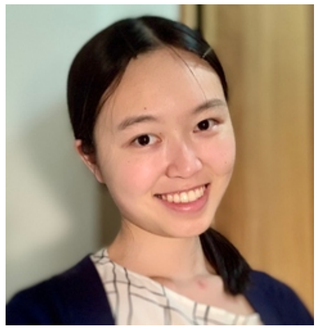
527031f92ffae4286
87521850702de15
da843c27062ee79c

Watermarking

Yay! I made a fast matrix multiplication algorithm!

3bcc4baa285258a4
242c4bd5092108fa
8ac7460be9a97706

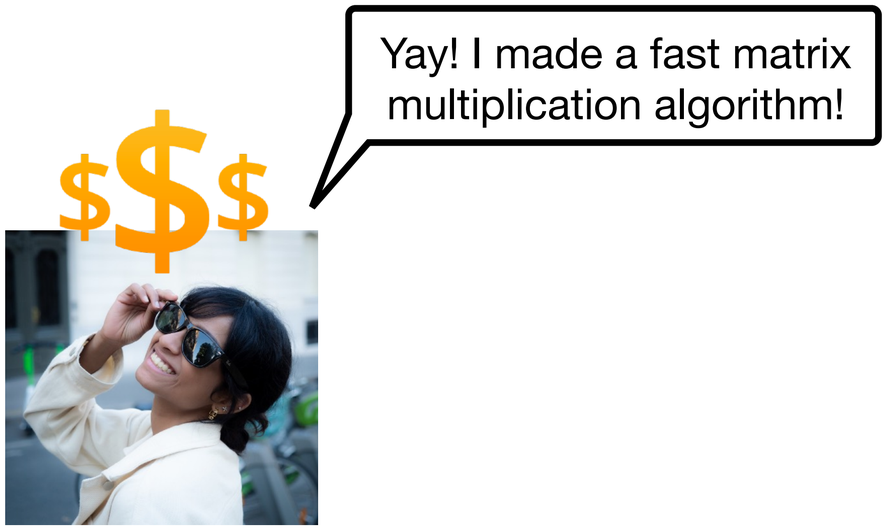
6bf31c3e5aa0e434
46f98ceb880d6750
0a7be5d9807d11cf



527031f92ffae4286
87521850702de15
da843c27062ee79c

The watermarks are
now difficult to remove!

The algorithm itself!



The algorithm itself!

Yay! I made a fast matrix multiplication algorithm!

$1 million

FMM
Runtime: $O(n^2)$

The algorithm itself!

Yay! I made a fast matrix multiplication algorithm!

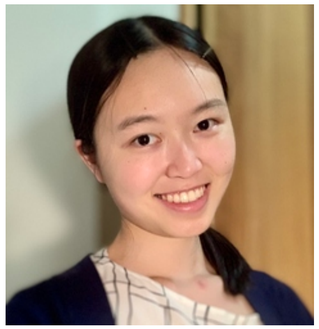
$1 million

FMM
Runtime: $O(n^2)$

Algorithm with $O(n^{2.5})$ dummy steps

$10k

FMM
Runtime: $O(n^{2.5})$



The algorithm itself!

Yay! I made a fast matrix multiplication algorithm!

Algorithm with $O(n^{2.5})$ dummy steps

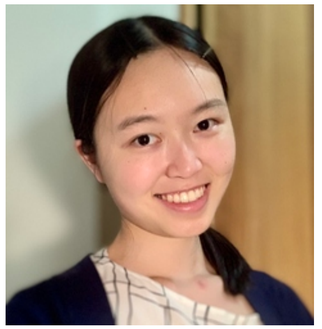
Algorithm with $O(n^3)$ dummy steps

$1 million

FMM
Runtime: $O(n^2)$

$10k

FMM
Runtime: $O(n^{2.5})$



$1

FMM
Runtime: $O(n^3)$

The algorithm itself!

Yay! I made a fast matrix multiplication algorithm!

Algorithm with $O(n^{2.5})$ dummy steps

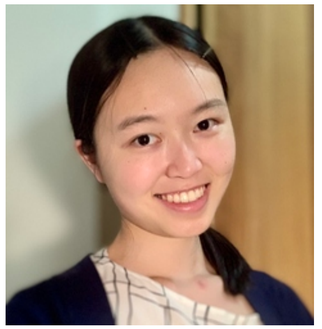
Algorithm with $O(n^3)$ dummy steps

$1 million

cdb443b0a41f518d
9ae55ae65a3c9c8d
ac37e34f63ad52f7a

$10k

088c2493182b421f
1b376c46909148cb
6a732c75ac75b989



$1

4ebaf73b13cd4d8d
c89bc0141b21d5c2
e9081ec6ff8a68f0d

The algorithm itself!

Yay! I made a fast matrix multiplication algorithm!

Algorithm with $O(n^{2.5})$ dummy steps

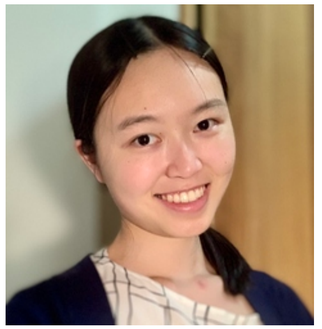
Algorithm with $O(n^3)$ dummy steps

$1 million

cdb443b0a41f518d
9ae55ae65a3c9c8d
ac37e34f63ad52f7a

$10k

088c2493182b421f
1b376c46909148cb
6a732c75ac75b989



$1

4ebaf73b13cd4d8d
c89bc0141b21d5c2
e9081ec6ff8a68f0d

This makes the fast algorithm hard to extract!

So… How do we define obfuscation?

Virtual Black-Box

Virtual Black-Box

A ppt algorithm $\mathcal{O}$ is an obfuscation for a collection $\mathscr{C}$ of circuits if:

Virtual Black-Box

A ppt algorithm $\mathcal{O}$ is an obfuscation for a collection $\mathcal{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[\mathcal{O}(C; r) = C] = 1.$

Virtual Black-Box

A ppt algorithm $\mathcal{O}$ is an obfuscation for a collection $\mathcal{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[\mathcal{O}(C; r) = C] = 1.$
- **(Polynomial slowdown)** The size of $\mathcal{O}(C)$ of is poly($|C|$).

Virtual Black-Box

A ppt algorithm $\mathcal{O}$ is an obfuscation for a collection $\mathcal{C}$ of circuits if:

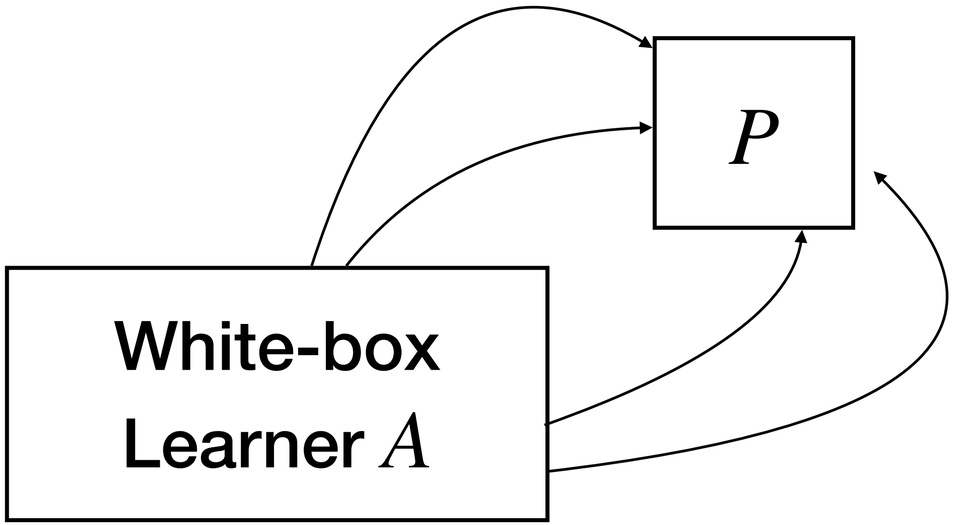
- **(Perfect functionality)** $\Pr_r[\mathcal{O}(C; r) = C] = 1.$
- **(Polynomial slowdown)** The size of $\mathcal{O}(C)$ of is poly($|C|$).
- **(VBB property)** $\mathcal{O}(C)$ reveals no more information than black-box access to C!

Virtual Black-Box

- $\mathcal{O}(C)$ reveals no more information than black-box access to C!

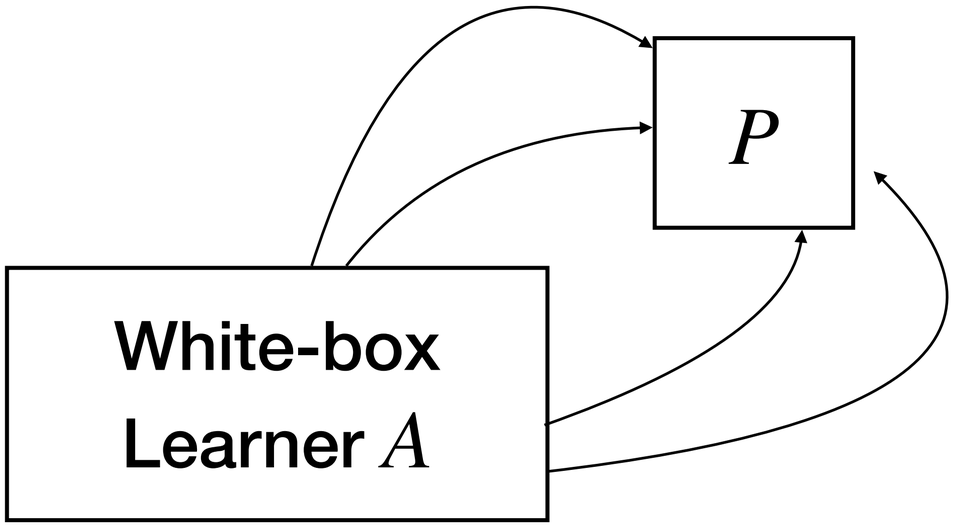
Virtual Black-Box

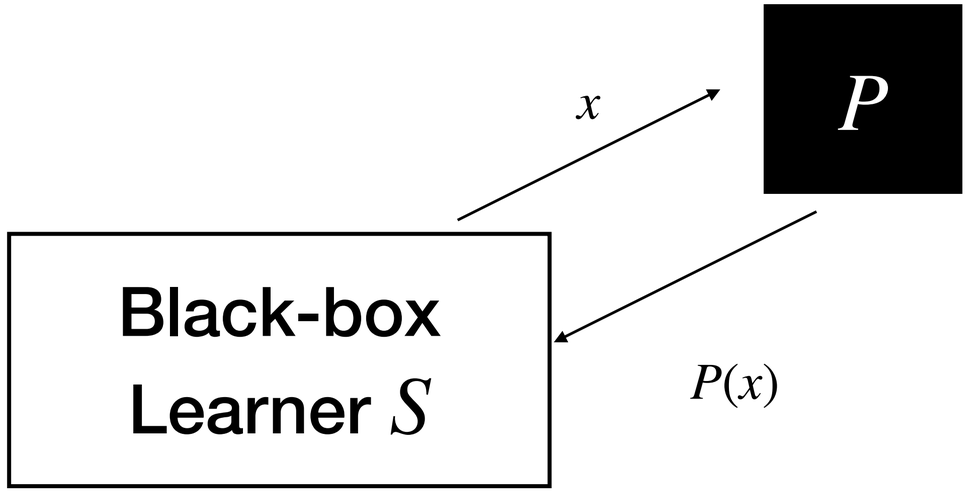
- $\mathcal{O}(C)$ reveals no more information than black-box access to C!



Virtual Black-Box

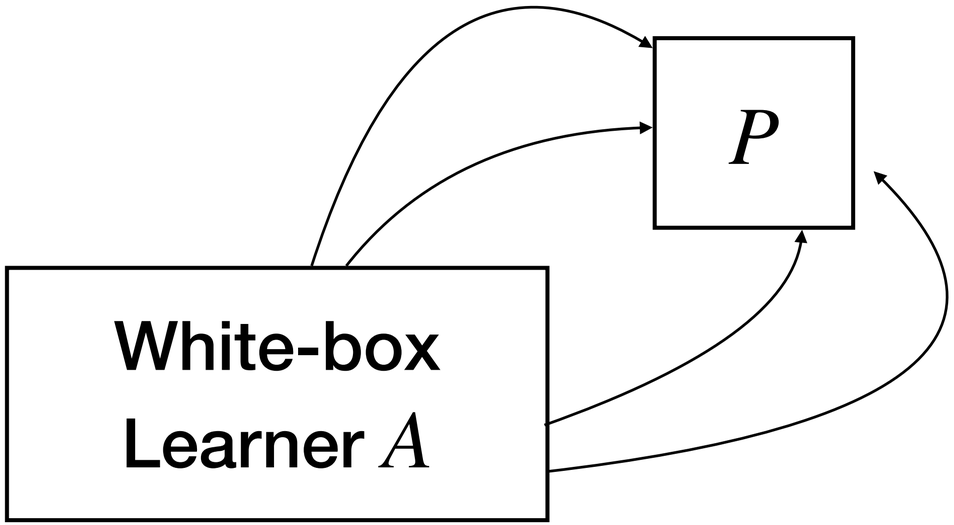
- $\mathcal{O}(C)$ reveals no more information than black-box access to C!

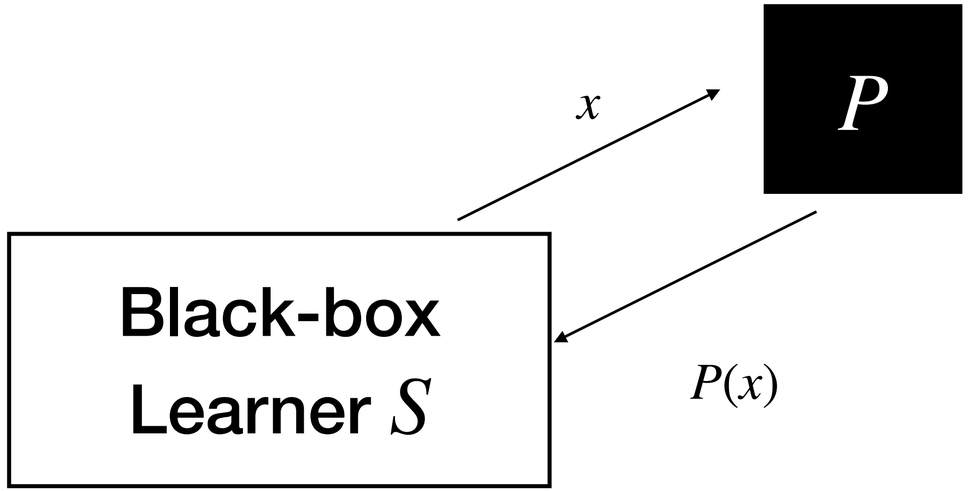




Virtual Black-Box

- $\mathcal{O}(C)$ reveals no more information than black-box access to C!



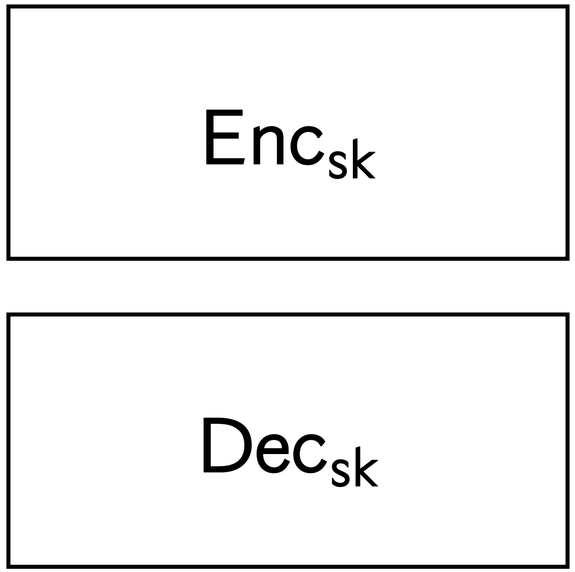


For all white-box learners A, there exists a simulator S such that

$$|\Pr[A(\mathcal{O}(C)) = 1] - \Pr[S^C(1^{|C|}) = 1]| \leq \mathsf{negl}(|C|)$$

PKE from SKE!

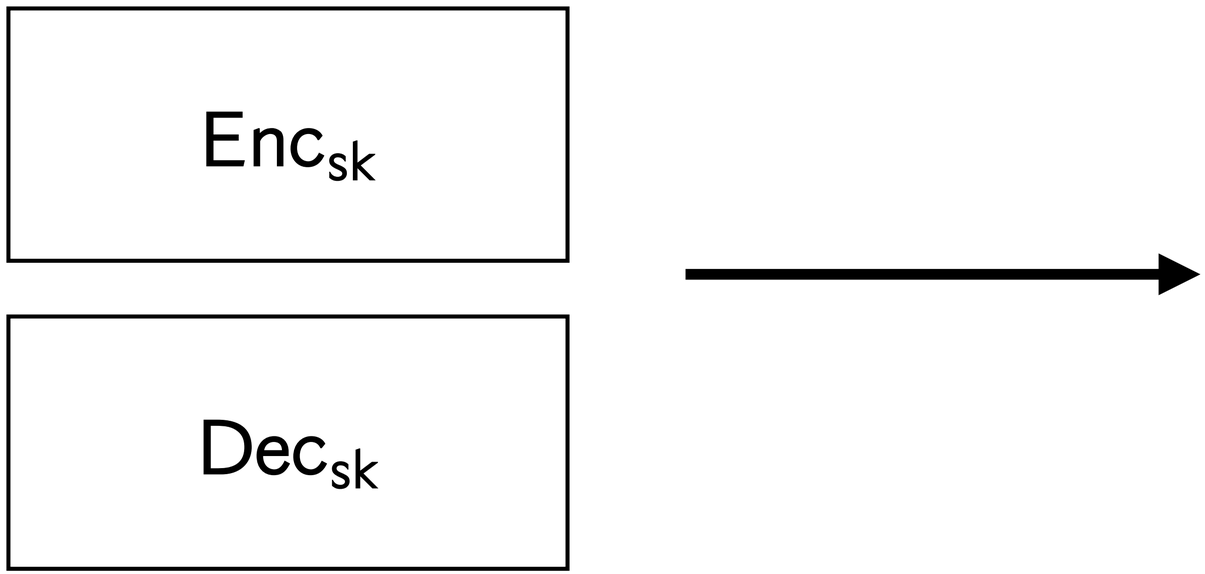
Secret-Key Encryption



Public-Key Encryption!

PKE from SKE!

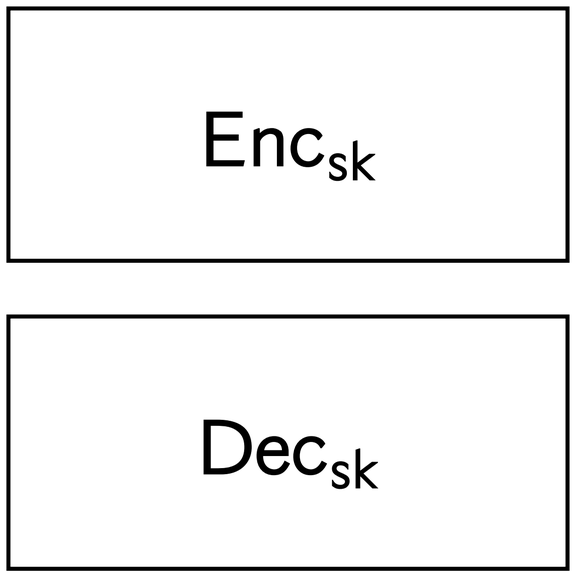
Secret-Key Encryption



Public-Key Encryption!

PKE from SKE!

Secret-Key Encryption

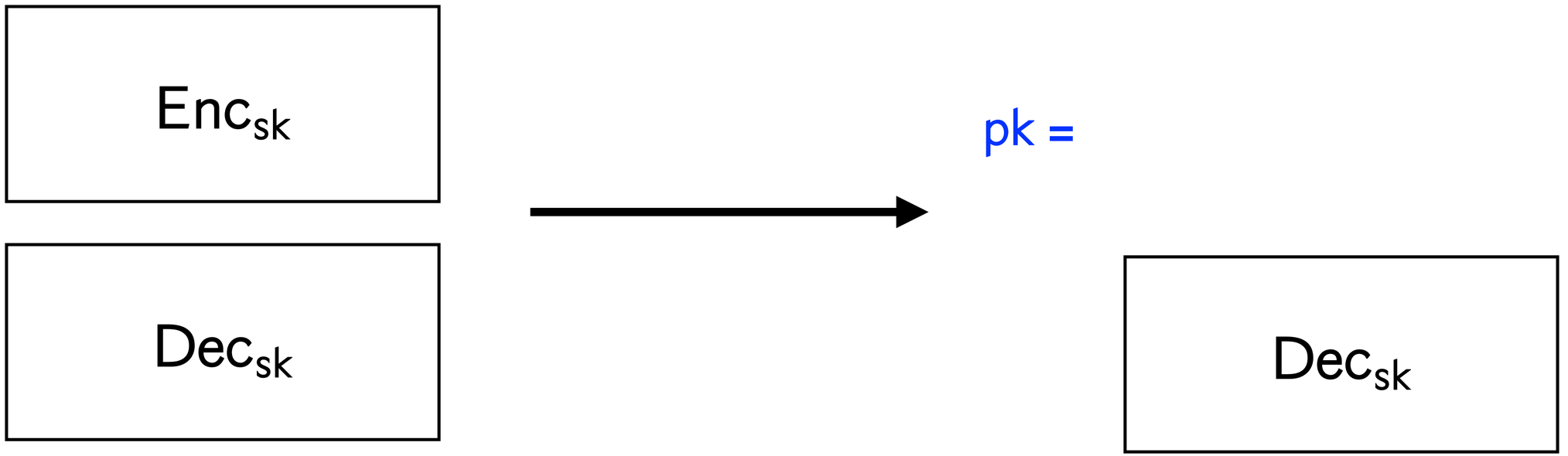


Public-Key Encryption!

Dec_{sk}

PKE from SKE!

Secret-Key Encryption



Public-Key Encryption!

PKE from SKE!

Secret-Key Encryption

Enc_{sk}

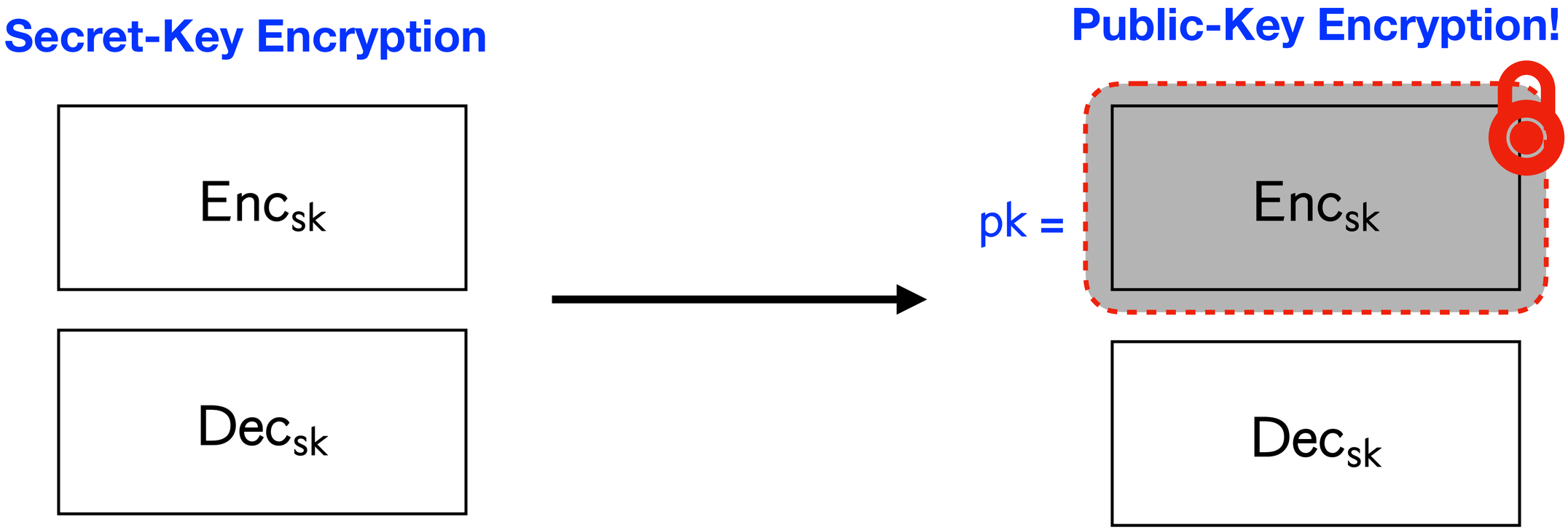
Dec_{sk}

Public-Key Encryption!

$pk =$ Enc_{sk}

Dec_{sk}

PKE from SKE!



OWF + VBB gives public key encryption!!

Note that we previously used LWE, DDH, RSA, QR etc. to obtain PKE from SKE.

Diffie-Hellman (1976)

Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-way because it must be feasible to do the compilation, but infeasible to reverse the process. Since efficiency in size of program and run time are not crucial in this application, such compilers may be possible if the structure of the machine language can be optimized to assist in the confusion.

Fully-Homomorphic Encryption

OWF + VBB also gives you FHE!!

Fully-Homomorphic Encryption

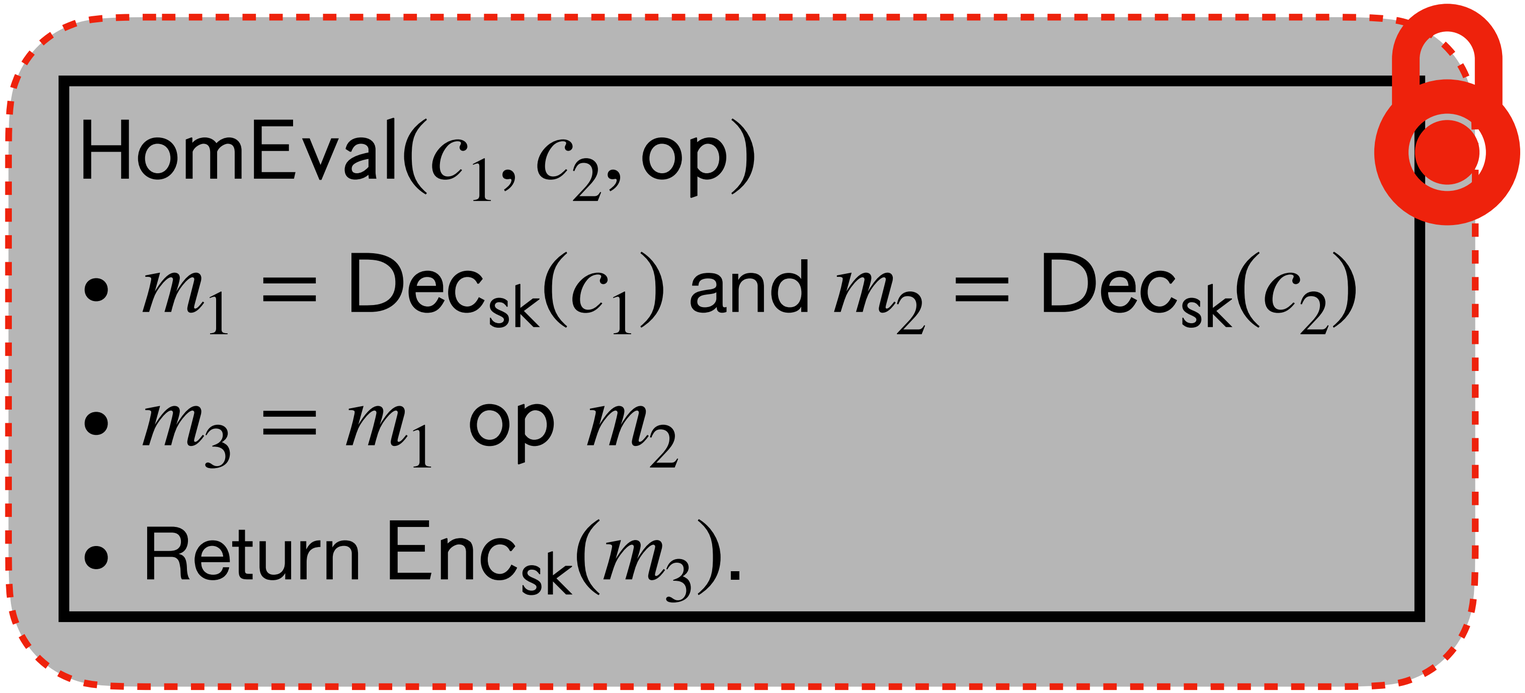
OWF + VBB also gives you FHE!!

HomEval(c_1, c_2, op)

- $m_1 = \mathsf{Dec}_{\mathsf{sk}}(c_1)$ and $m_2 = \mathsf{Dec}_{\mathsf{sk}}(c_2)$
- $m_3 = m_1$ op m_2
- Return $\mathsf{Enc}_{\mathsf{sk}}(m_3)$.

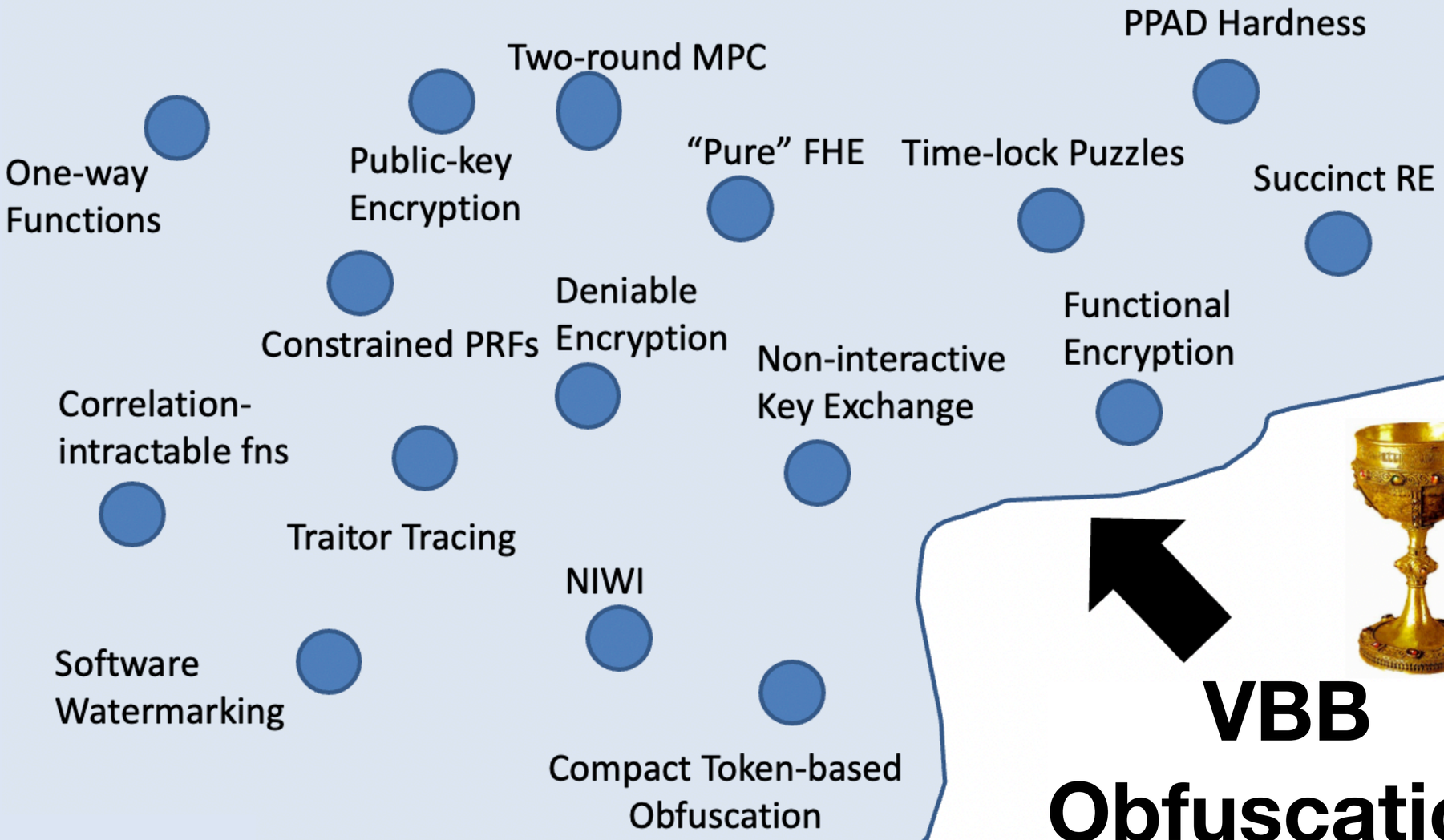
Fully-Homomorphic Encryption

OWF + VBB also gives you FHE!!



"CRYPTO-COMPLETE":

Nearly all crypto is an easy corollary of VBB!



Bad news…

Bad news…

Theorem [BGIRSVY ’01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

Bad news…

Theorem [BGIRSVY ’01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

Proof: “Programs that eat themselves”.

Bad news…

Theorem [BGIRSVY ’01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

Proof: “Programs that eat themselves”.

Bad news…

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

Proof: "Programs that eat themselves".

Define a family of programs $\mathscr{P} = \{P_{x,y}\}$ where x and y are n-bit strings as follows:

Bad news…

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

Proof: "Programs that eat themselves".

Define a family of programs $\mathscr{P} = \{P_{x,y}\}$ where x and y are n-bit strings as follows:

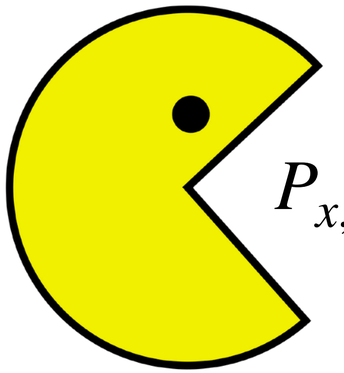
$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b = 0,\ \Pi = x \\ x, y & \text{if } b = 1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise.} \end{cases}$$

Bad news…

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

Proof: "Programs that eat themselves".

Define a family of programs $\mathscr{P} = \{P_{x,y}\}$ where x and y are n-bit strings as follows:



$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b = 0,\ \Pi = x \\ x, y & \text{if } b = 1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise.} \end{cases}$$

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b = 0,\ \Pi = x \\ x, y & \text{if } b = 1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise.} \end{cases}$$

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b = 0,\ \Pi = x \\ x, y & \text{if } b = 1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise.} \end{cases}$$

1. **Black-Box access to $P_{x,y}$:**

   Useless! For random x, y, an algorithm cannot distinguish $P_{x,y}$ from zero function.

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b = 0,\ \Pi = x \\ x, y & \text{if } b = 1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise.} \end{cases}$$

1. **Black-Box access to $P_{x,y}$:**

   Useless! For random x, y, an algorithm cannot distinguish $P_{x,y}$ from zero function.

2. **Can recover x, y given obfuscated code!**

   Given $P' = \mathcal{O}(P)$, run $P'(1, P')$.

Theorem [BGIRSVY '01]. $\forall \mathcal{O}, \exists P$ such that $\mathcal{O}$ fails to VBB obfuscate P.

$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b = 0,\ \Pi = x \\ x, y & \text{if } b = 1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise.} \end{cases}$$

1. **Black-Box access to $P_{x,y}$:**

   Useless! For random x, y, an algorithm cannot distinguish $P_{x,y}$ from zero function.

2. **Can recover x, y given obfuscated code!**

   Given $P' = \mathcal{O}(P)$, run $P'(1, P')$. □

Remarks

- **One interpretation:** Black-box security cannot be obtained if we give “physical code”
- Even inefficient VBB is impossible!
- Can be extended to construct unobfuscatable encryption/signature schemes.

Q: Uhm… What now?

Q: Uhm… What now?

A: Weaken the definition!!

Indistinguishability Obfuscation

Indistinguishability Obfuscation

A ppt algorithm $i\mathcal{O}$ is an obfuscation for a collection $\mathscr{C}$ of circuits if:

Indistinguishability Obfuscation

A ppt algorithm $i\mathcal{O}$ is an obfuscation for a collection $\mathscr{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[i\mathcal{O}(C;r) = C] = 1.$

Indistinguishability Obfuscation

A ppt algorithm $i\mathcal{O}$ is an obfuscation for a collection $\mathscr{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[i\mathcal{O}(C;r) = C] = 1.$
- **(Polynomial slowdown)** The size of $i\mathcal{O}(C)$ is poly($|C|$).

Indistinguishability Obfuscation

A ppt algorithm $i\mathcal{O}$ is an obfuscation for a collection $\mathcal{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[i\mathcal{O}(C;r) = C] = 1.$
- **(Polynomial slowdown)** The size of $i\mathcal{O}(C)$ is poly($|C|$).
- **(Indistinguishability)** For all pairs C_0 and C_1 of the same size that compute the **same function:**

Indistinguishability Obfuscation

A ppt algorithm $i\mathcal{O}$ is an obfuscation for a collection $\mathcal{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[i\mathcal{O}(C;r) = C] = 1.$
- **(Polynomial slowdown)** The size of $i\mathcal{O}(C)$ is poly($|C|$).
- **(Indistinguishability)** For all pairs C_0 and C_1 of the same size that compute the **same function:**

$$i\mathcal{O}(C_0;r) \approx_c i\mathcal{O}(C_1;r)$$

Indistinguishability Obfuscation

A ppt algorithm $i\mathcal{O}$ is an obfuscation for a collection $\mathscr{C}$ of circuits if:

- **(Perfect functionality)** $\Pr_r[i\mathcal{O}(C;r) = C] = 1.$
- **(Polynomial slowdown)** The size of $i\mathcal{O}(C)$ is poly($|C|$).
- **(Indistinguishability)** For all pairs C_0 and C_1 of the same size that compute the **same function:**

$$i\mathcal{O}(C_0;r) \approx_c i\mathcal{O}(C_1;r)$$

Compare iO vs VBB

Virtual Black-Box Obfuscation

- **(Perfect functionality)**
  $\Pr_r[\mathcal{O}(C; r) = C] = 1.$
- **(Polynomial slowdown)**
  The size of $\mathcal{O}(C)$ is poly($|C|$).
- **(VBB property)**
  $\mathcal{O}(C)$ reveals no more information than black-box access to C!

Indistinguishability Obfuscation

- **(Perfect functionality)**
  $\Pr_r[i\mathcal{O}(C; r) = C] = 1.$
- **(Polynomial slowdown)**
  The size of $i\mathcal{O}(C)$ is poly($|C|$).
- **(Indistinguishability)**
  For all pairs C_0 and C_1 computing the same function:
  $$i\mathcal{O}(C_0; r) \approx_c i\mathcal{O}(C_1; r)$$

If P = NP, iO exists!

If P = NP, iO exists!

Proof:

If P = NP, iO exists!

Proof:

For any circuit C, let $i\mathcal{O}$ of C be the **lexicographically first circuit** C' that computes the same function as C. Then, we have found a **canonical** representation of each C.

If P = NP, iO exists!

Proof:

For any circuit C, let $i\mathcal{O}$ of C be the **lexicographically first circuit** C' that computes the same function as C. Then, we have found a **canonical** representation of each C.

This can be done in P if P = NP.

If P = NP, iO exists!

Proof:

For any circuit C, let $i\mathcal{O}$ of C be the **lexicographically first circuit** C' that computes the same function as C. Then, we have found a **canonical** representation of each C.

This can be done in P if P = NP. □

If P = NP, iO exists!

Proof:

For any circuit C, let $i\mathcal{O}$ of C be the **lexicographically first circuit** C' that computes the same function as C. Then, we have found a **canonical** representation of each C.

This can be done in P if P = NP. □

If P = NP, iO exists!

Proof:

For any circuit C, let $i\mathcal{O}$ of C be the **lexicographically first circuit** C' that computes the same function as C. Then, we have found a **canonical** representation of each C.

This can be done in P if P = NP. □

Remark 1: In fact, one can think of iO as a “pseudo-canonicaliser”.

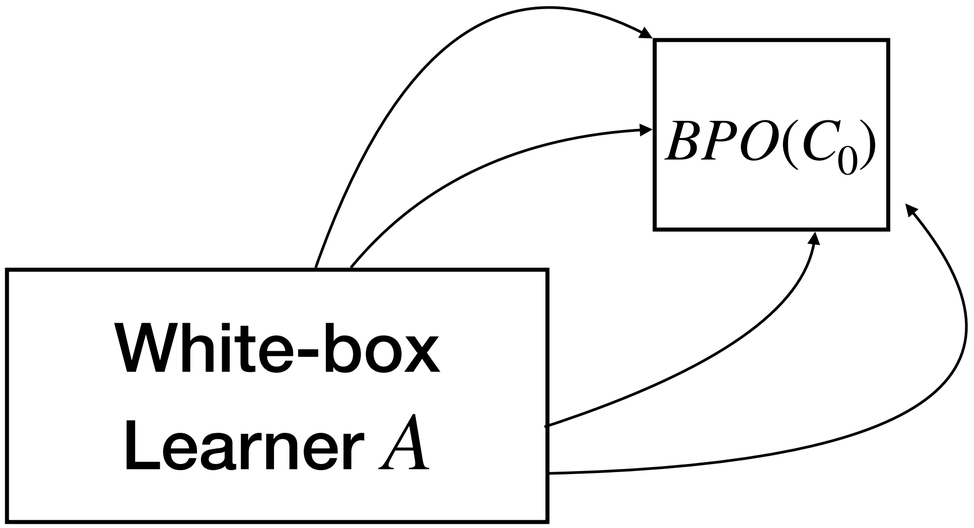
Remark 2: This fact means that it is hard to show iO implies OWF (if it did, P ≠ NP).

Best possible obfuscation

Anything you can learn from a BPO of C_0 can be learned from **any** circuit C_1 computing an identical function.

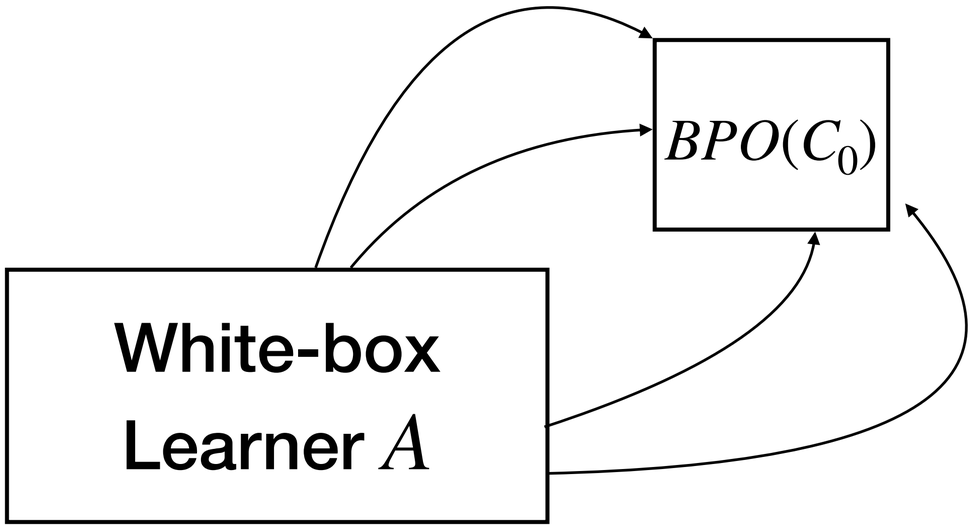
Best possible obfuscation

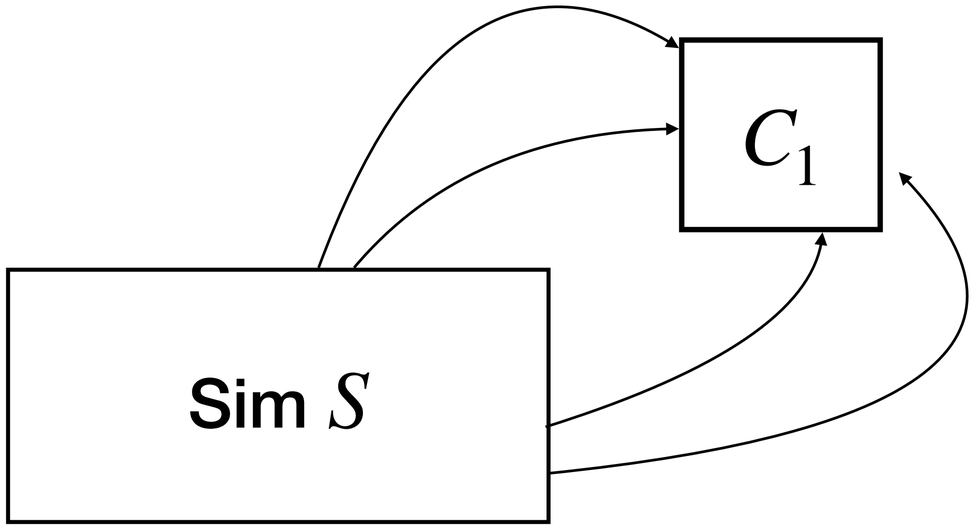
Anything you can learn from a BPO of C_0 can be learned from **any** circuit C_1 computing an identical function.



Best possible obfuscation

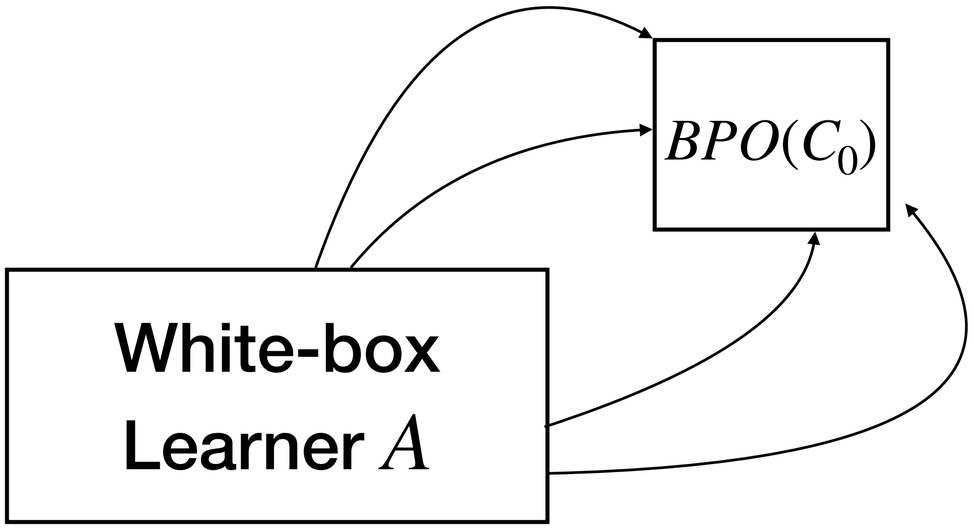
Anything you can learn from a BPO of C_0 can be learned from **any** circuit C_1 computing an identical function.

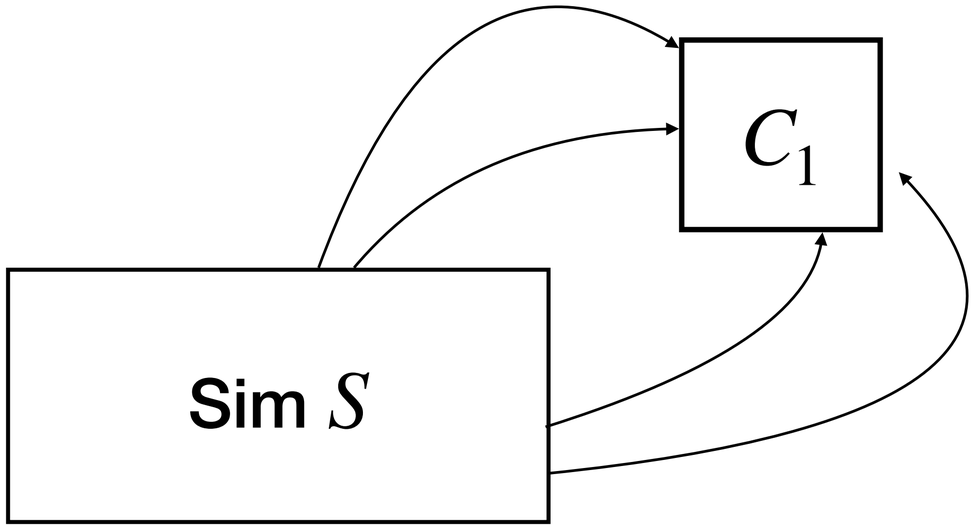




Best possible obfuscation

Anything you can learn from a BPO of C_0 can be learned from **any** circuit C_1 computing an identical function.





For all white-box learners A, there exists a simulator S such that

$$|\Pr[A(BPO(C_0)) = 1] - \Pr[S(C_1) = 1]| \leq \mathsf{negl}(|C|)$$

iO is “as good as it gets”

iO is “as good as it gets”

Theorem [GR07]. iO is a *best possible obfuscation.*

iO is “as good as it gets”

Theorem [GR07]. iO is a *best possible obfuscation.*

Proof: Let $BPO = i\mathcal{O}$. For any learner A, let the simulator S be the algorithm that first computes the $i\mathcal{O}$ of the input, and then runs A.

iO is “as good as it gets”

Theorem [GR07]. iO is a *best possible obfuscation.*

Proof: Let $BPO = i\mathcal{O}$. For any learner A, let the simulator S be the algorithm that first computes the $i\mathcal{O}$ of the input, and then runs A.

$$\begin{aligned} & |\Pr[A(BPO(C_0)) = 1] - \Pr[S(C_1) = 1]| \\ \leq\ & |\Pr[A(i\mathcal{O}(C_0)) = 1] - \Pr[A(i\mathcal{O}(C_1)) = 1]| = \mathsf{negl}(|C|) \end{aligned}$$

iO is “as good as it gets”

Theorem [GR07]. iO is a *best possible obfuscation.*

Proof: Let $BPO = i\mathcal{O}$. For any learner A, let the simulator S be the algorithm that first computes the $i\mathcal{O}$ of the input, and then runs A.

$$
\begin{aligned}
& |\Pr[A(BPO(C_0)) = 1] - \Pr[S(C_1) = 1]| \\
\leq\ & |\Pr[A(i\mathcal{O}(C_0)) = 1] - \Pr[A(i\mathcal{O}(C_1)) = 1]| = \mathsf{negl}(|C|)
\end{aligned}
$$

By indistinguishability!

iO is “as good as it gets”

Theorem [GR07]. iO is a *best possible obfuscation.*

Proof: Let $BPO = i\mathcal{O}$. For any learner A, let the simulator S be the algorithm that first computes the $i\mathcal{O}$ of the input, and then runs A.

$$
\begin{aligned}
& |\Pr[A(BPO(C_0)) = 1] - \Pr[S(C_1) = 1]| \\
\leq\ & |\Pr[A(i\mathcal{O}(C_0)) = 1] - \Pr[A(i\mathcal{O}(C_1)) = 1]| = \mathsf{negl}(|C|)
\end{aligned}
$$

□

iO is “as good as it gets”

Theorem [GR07]. iO is a *best possible obfuscation.*

Proof: Let $BPO = i\mathcal{O}$. For any learner A, let the simulator S be the algorithm that first computes the $i\mathcal{O}$ of the input, and then runs A.

$$\begin{aligned} & |\Pr[A(BPO(C_0)) = 1] - \Pr[S(C_1) = 1]| \\ \leq\ & |\Pr[A(i\mathcal{O}(C_0)) = 1] - \Pr[A(i\mathcal{O}(C_1)) = 1]| = \mathsf{negl}(|C|) \end{aligned}$$

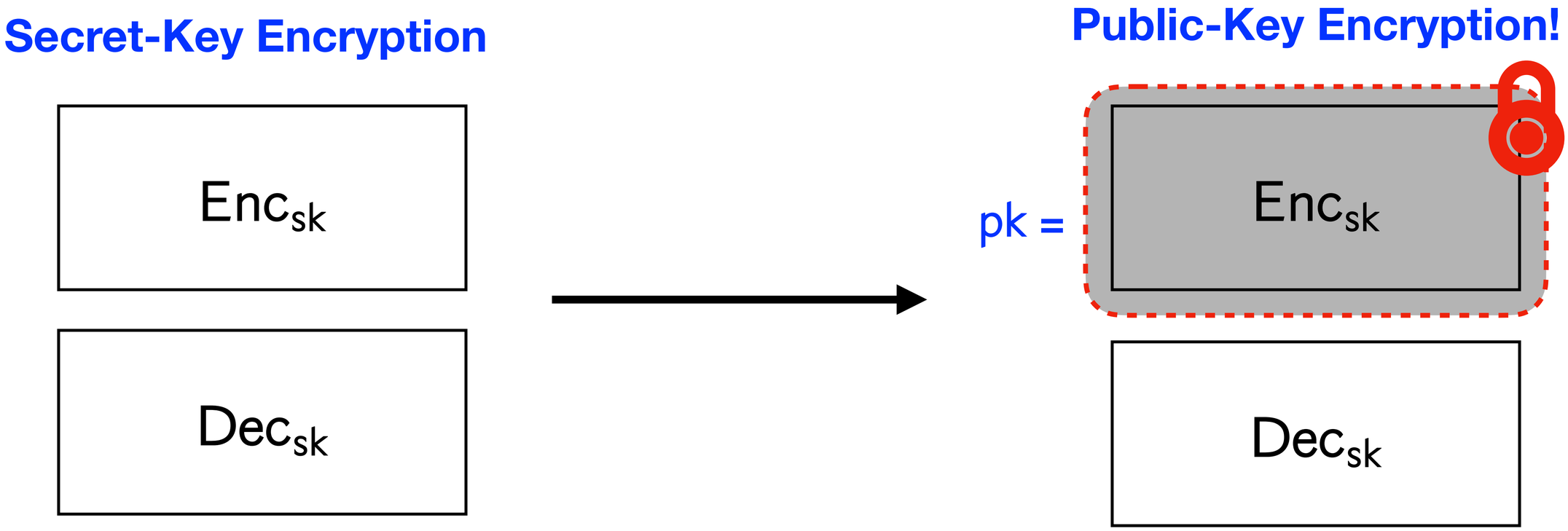
□

Corollary. If a circuit family has VBB obfuscation, then iO is a VBB obfuscation for this family.

Ok… But what can you do with iO?

iO Gymnastics

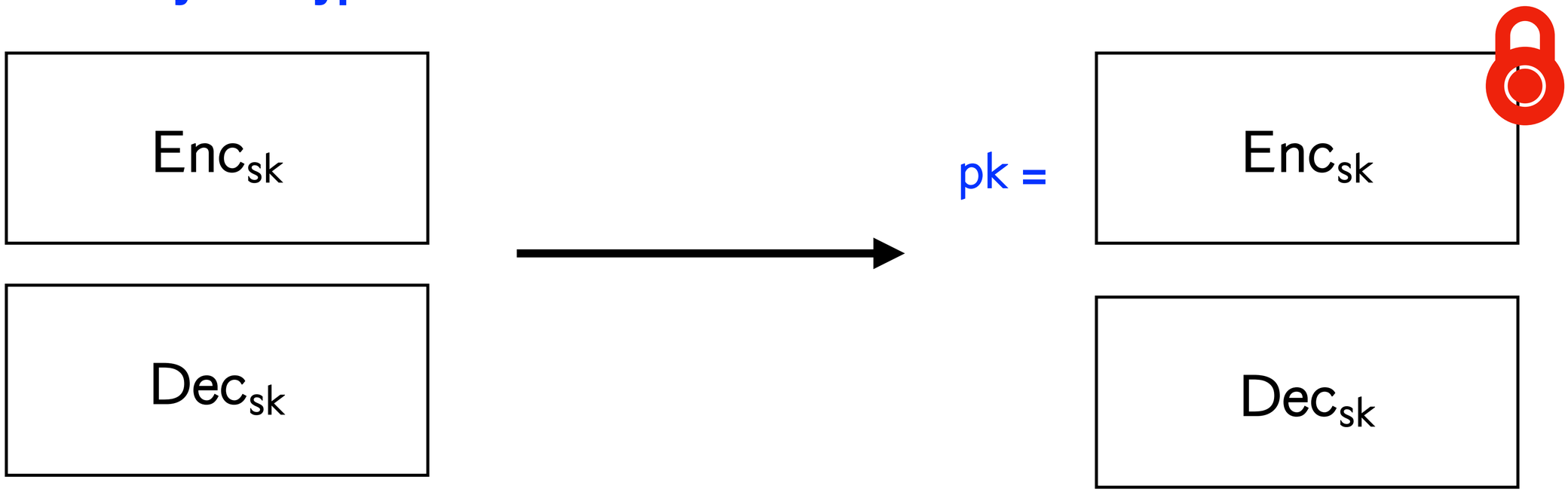
Recall: PKE from SKE!



OWF + VBB gives public key encryption!!

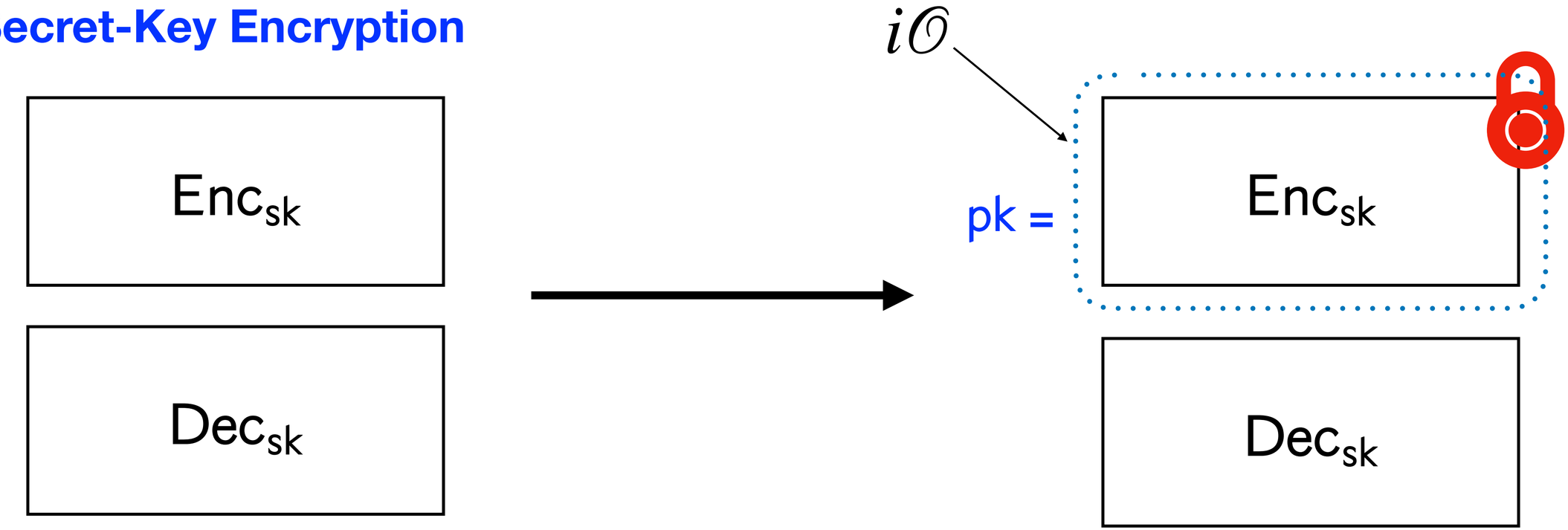
Recall: PKE from SKE!

Secret-Key Encryption

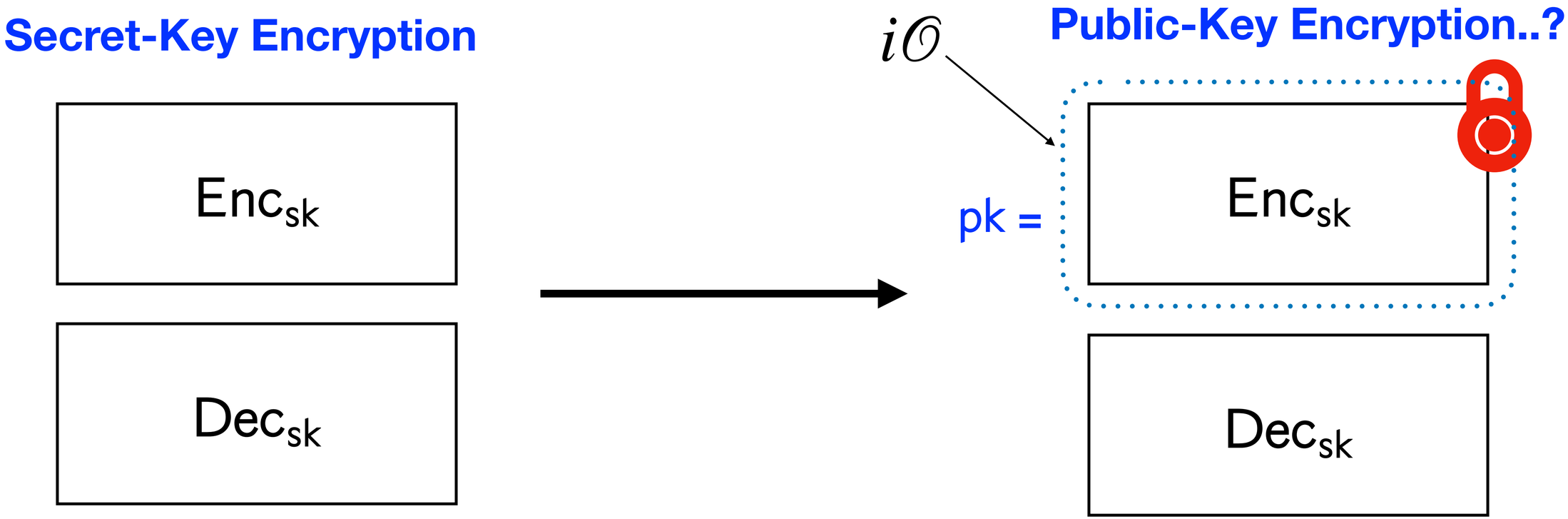


Recall: PKE from SKE!

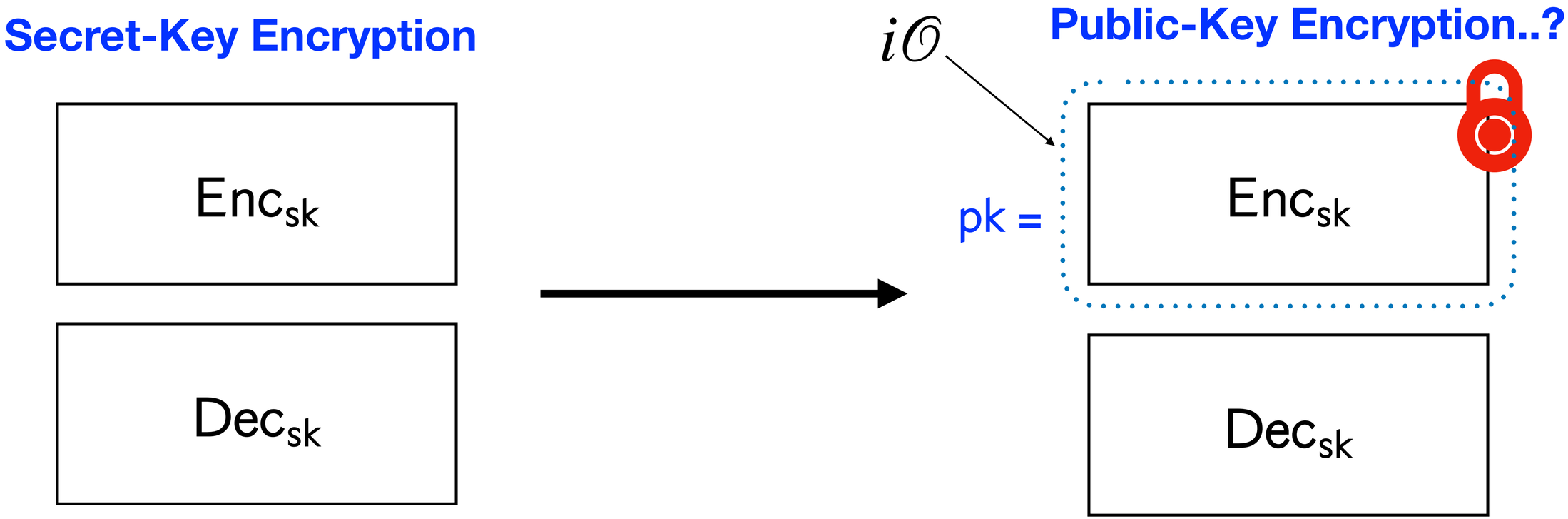
Secret-Key Encryption



Recall: PKE from SKE!



Recall: PKE from SKE!



But… iO doesn't really gives us much to work with in this picture…

Theorem. iO + OWF gives us PKE :)

Theorem. iO + OWF gives us PKE :)

- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG.

Theorem. iO + OWF gives us PKE :)

- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG.
- $\mathsf{Gen}(1^n)$: Sample $s \leftarrow \{0,1\}^n$. Output the key-pair $\mathsf{sk} = s$ and $\mathsf{pk} = G(s)$.

Theorem. iO + OWF gives us PKE :)

- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG.
- Gen(1^n): Sample $s \leftarrow \{0,1\}^n$. Output the key-pair $\mathsf{sk} = s$ and $\mathsf{pk} = G(s)$.
- Enc(pk, m): Let $P = P_{\mathsf{pk},m}$ be the following program. Output $\widehat{P_{\mathsf{pk},m}} = i\mathcal{O}(P_{\mathsf{pk},m})$.

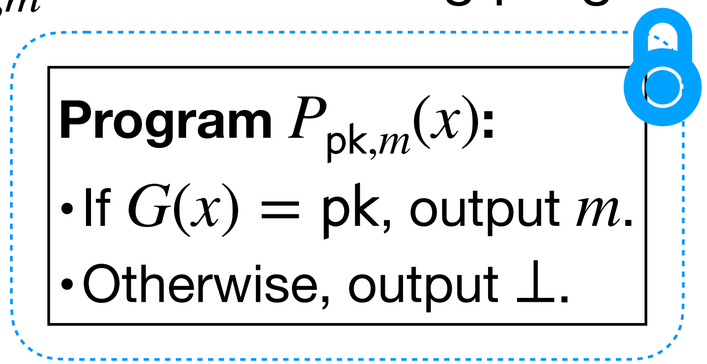
Theorem. iO + OWF gives us PKE :)

- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG.
- $\mathsf{Gen}(1^n)$: Sample $s \leftarrow \{0,1\}^n$. Output the key-pair $\mathsf{sk} = s$ and $\mathsf{pk} = G(s)$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: Let $P = P_{\mathsf{pk},m}$ be the following program. Output $\widehat{P_{\mathsf{pk},m}} = i\mathcal{O}(P_{\mathsf{pk},m})$.

Program $P_{\mathsf{pk},m}(x)$:
- If $G(x) = \mathsf{pk}$, output m.
- Otherwise, output $\perp$.

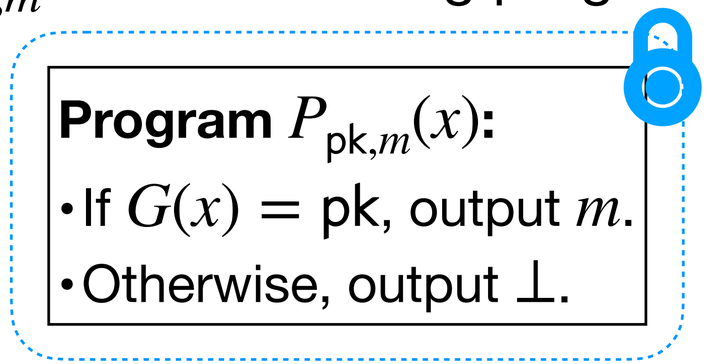
Theorem. iO + OWF gives us PKE :)

- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG.
- Gen(1^n): Sample $s \leftarrow \{0,1\}^n$. Output the key-pair $\mathsf{sk} = s$ and $\mathsf{pk} = G(s)$.
- Enc(pk, m): Let $P = P_{\mathsf{pk},m}$ be the following program. Output $\widehat{P_{\mathsf{pk},m}} = i\mathcal{O}(P_{\mathsf{pk},m})$.



Theorem. iO + OWF gives us PKE :)

- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG.
- Gen(1^n): Sample $s \leftarrow \{0,1\}^n$. Output the key-pair $\mathsf{sk} = s$ and $\mathsf{pk} = G(s)$.
- Enc(pk, m): Let $P = P_{\mathsf{pk},m}$ be the following program. Output $\widehat{P_{\mathsf{pk},m}} = i\mathcal{O}(P_{\mathsf{pk},m})$.



- Dec(sk, c): Interpret c as a program, run it on input $\mathsf{sk} = s$, and output the result.

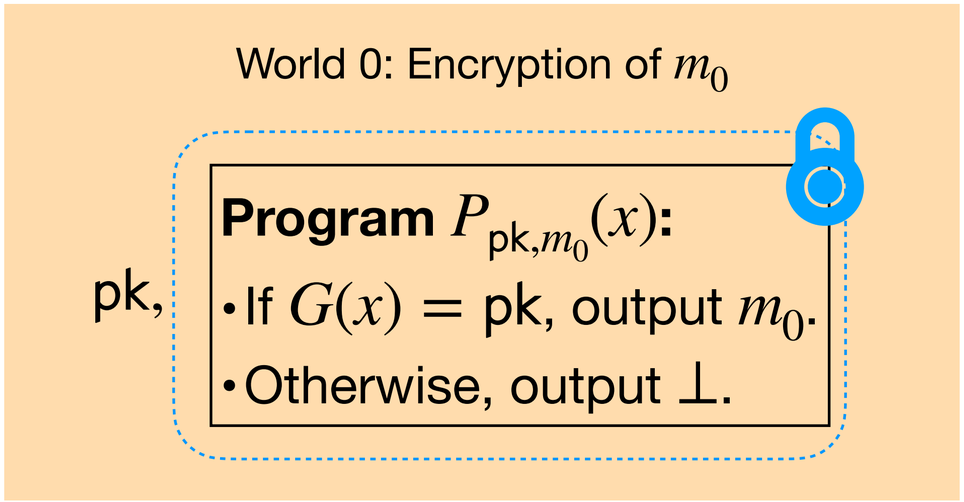
Theorem. iO + OWF gives us PKE :)

Theorem. iO + OWF gives us PKE :)

Proof: Want to show that $(\mathsf{pk}, Enc_{\mathsf{pk}}(m_0)) \approx_c (\mathsf{pk}, Enc_{\mathsf{pk}}(m_1))$.

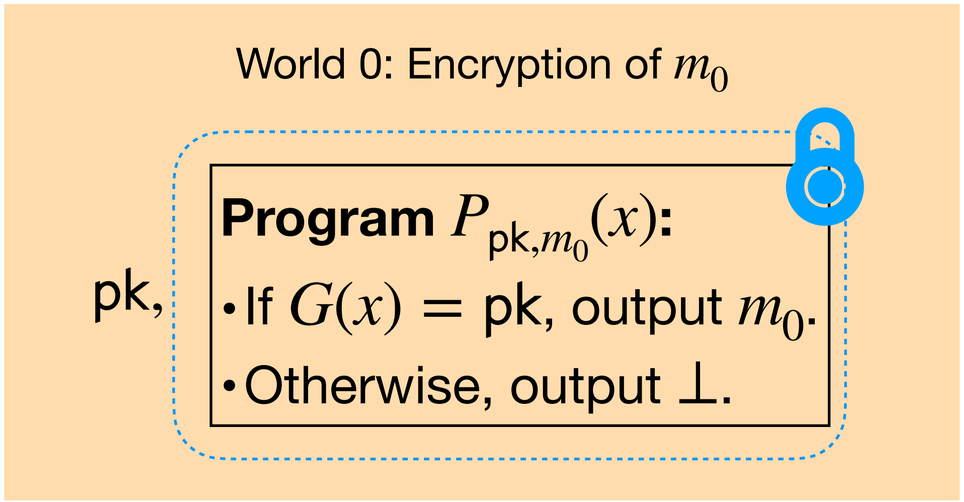
Theorem. iO + OWF gives us PKE :)

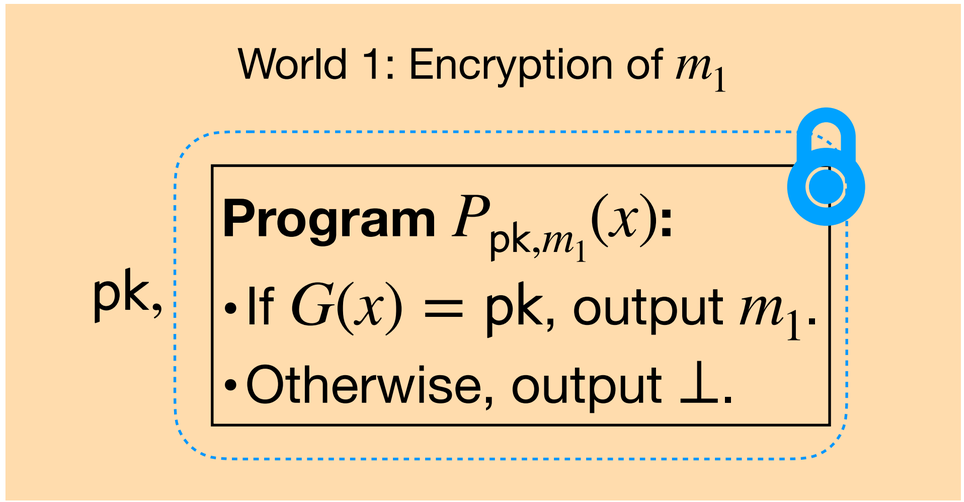
Proof: Want to show that $(\mathsf{pk}, Enc_{\mathsf{pk}}(m_0)) \approx_c (\mathsf{pk}, Enc_{\mathsf{pk}}(m_1))$.



Theorem. iO + OWF gives us PKE :)

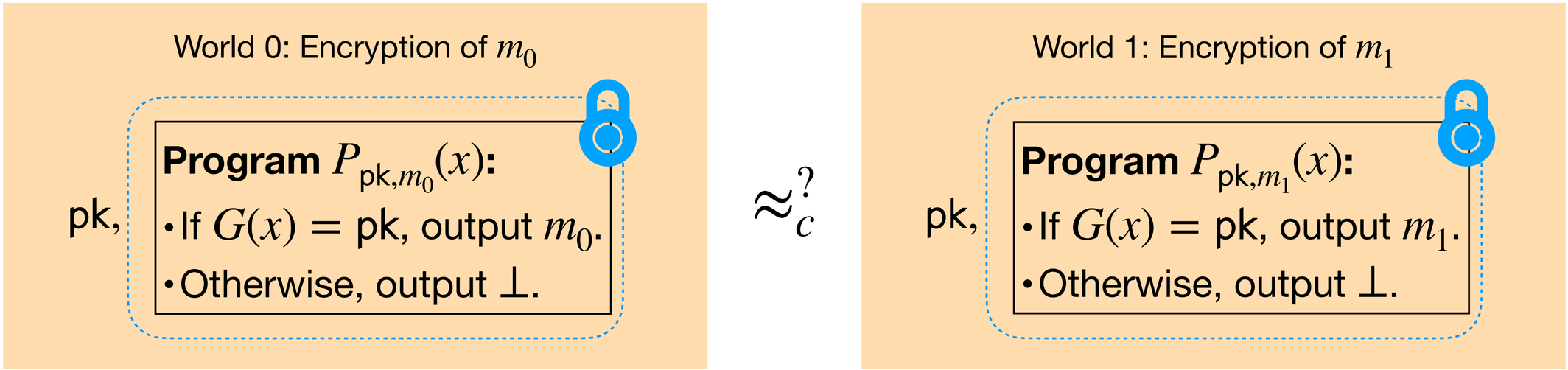
Proof: Want to show that $(\mathsf{pk}, Enc_{\mathsf{pk}}(m_0)) \approx_c (\mathsf{pk}, Enc_{\mathsf{pk}}(m_1))$.

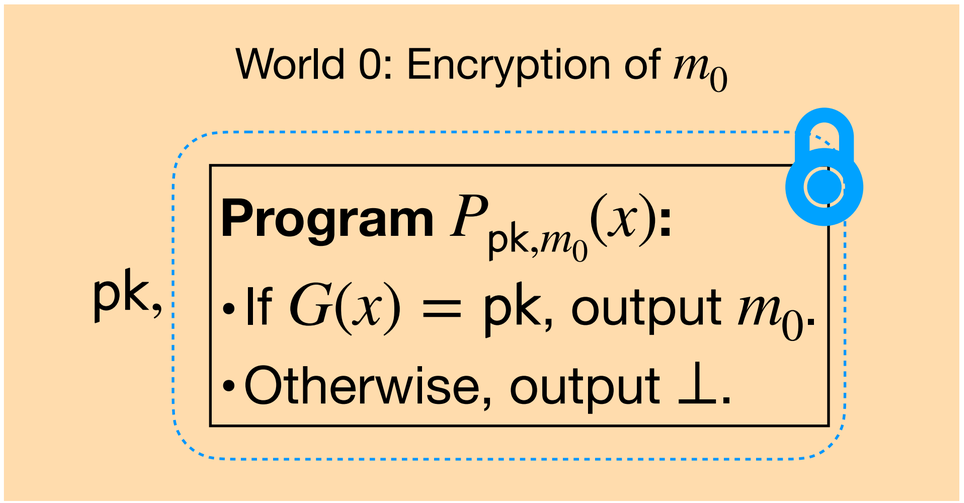


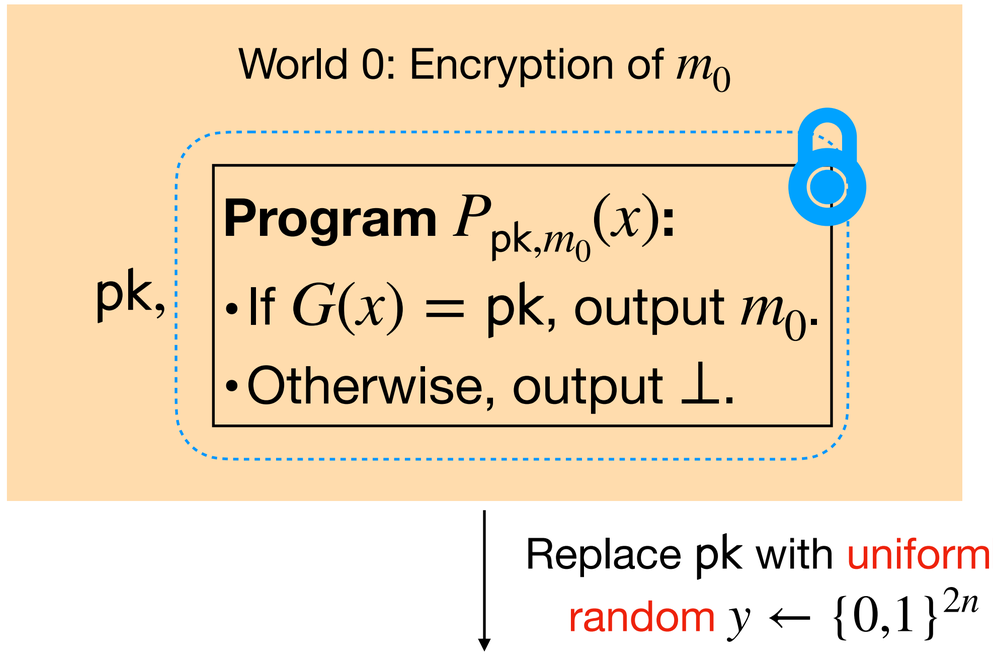


Theorem. iO + OWF gives us PKE :)

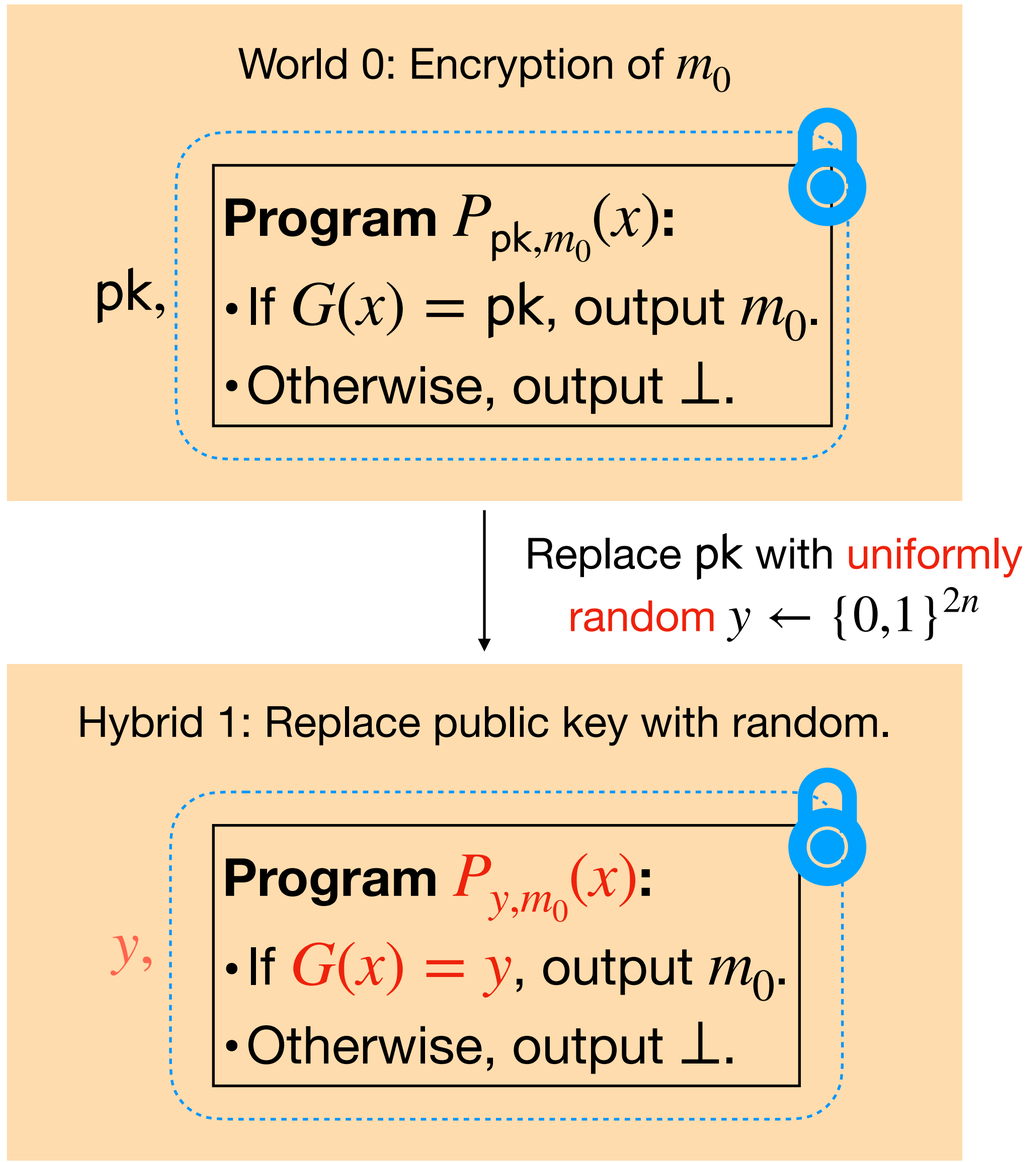
Proof: Want to show that $(\mathsf{pk}, Enc_{\mathsf{pk}}(m_0)) \approx_c (\mathsf{pk}, Enc_{\mathsf{pk}}(m_1))$.

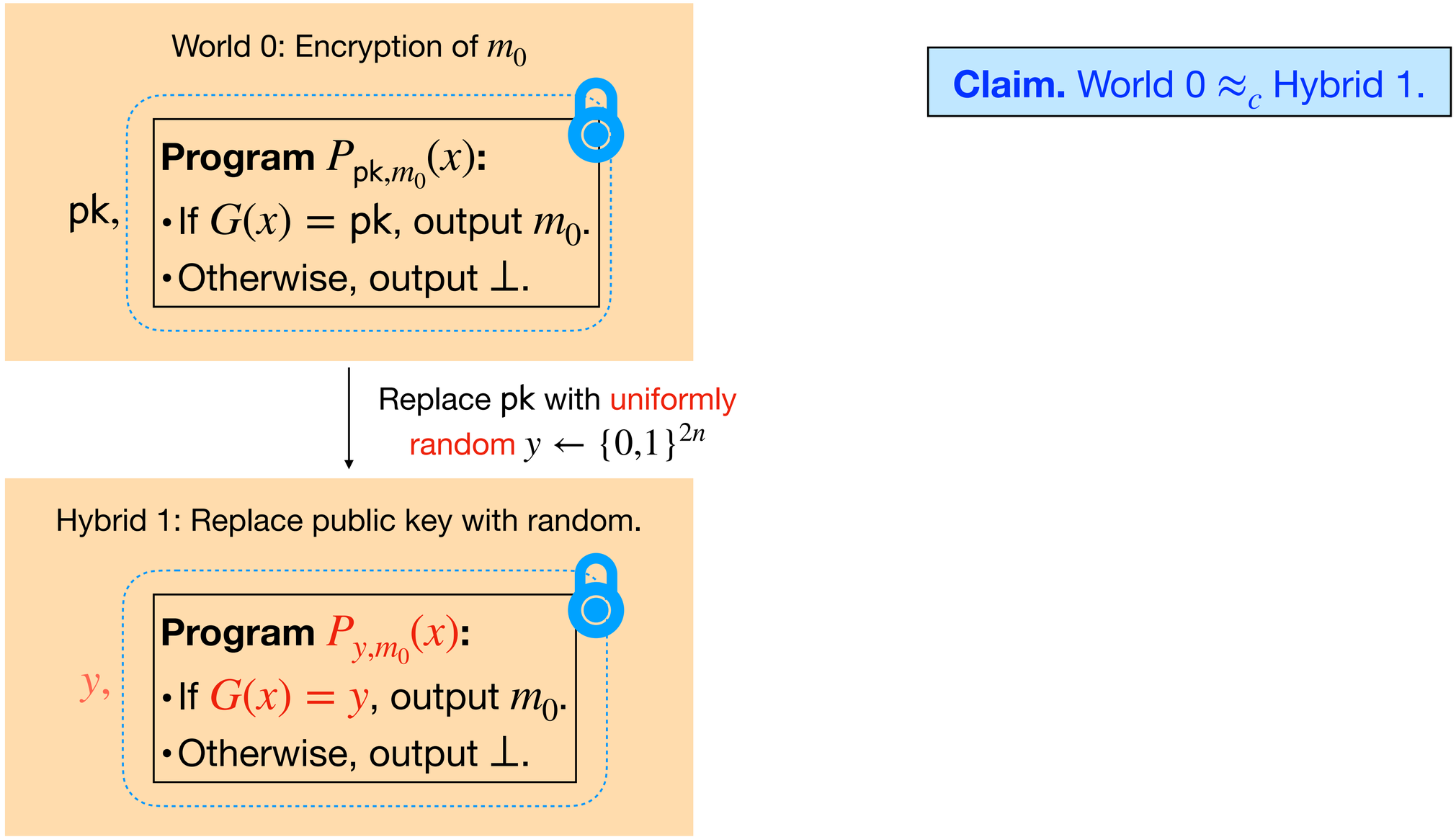



World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$**:**
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.

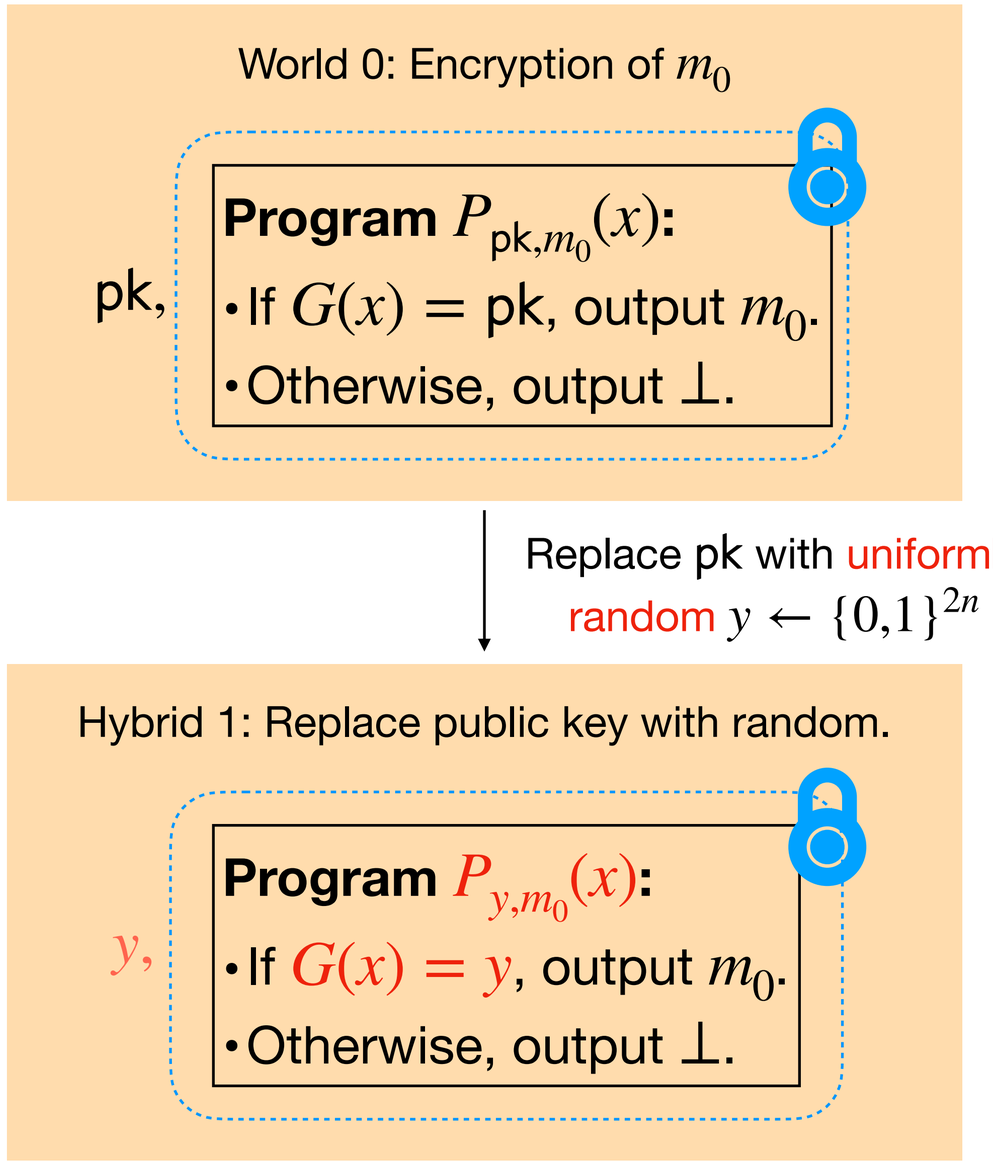


Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.

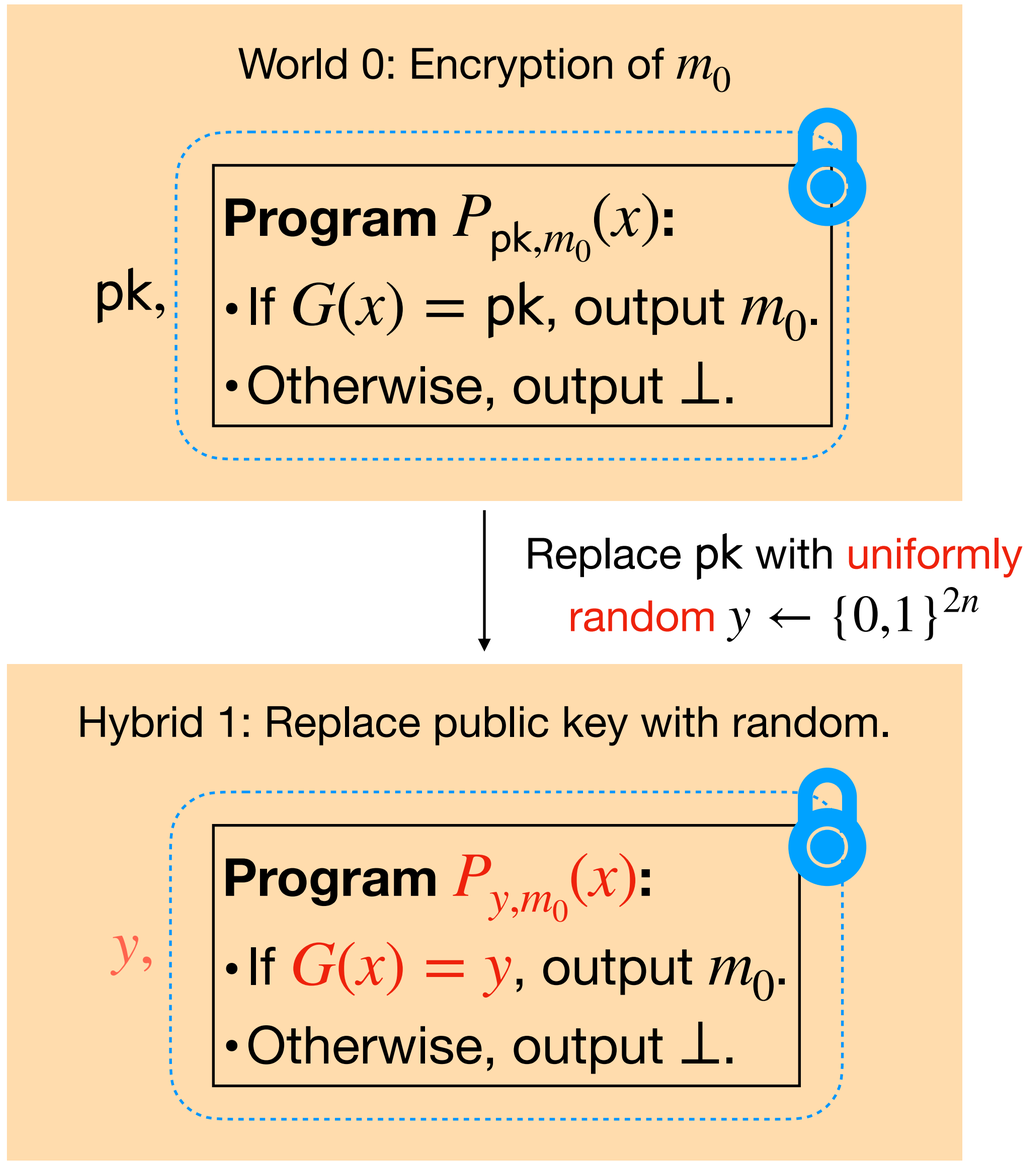


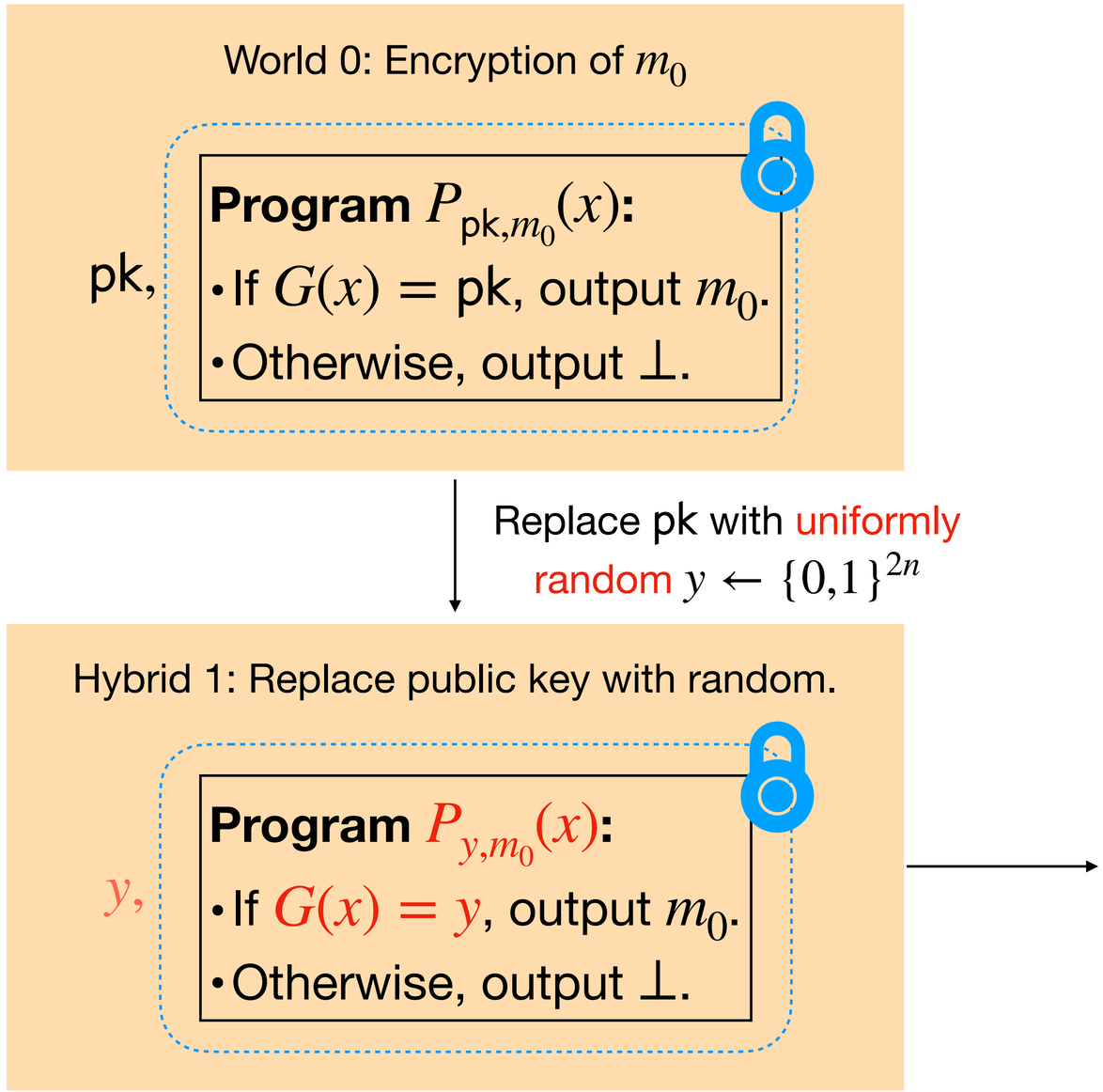
Claim. World 0 $\approx_c$ Hybrid 1.

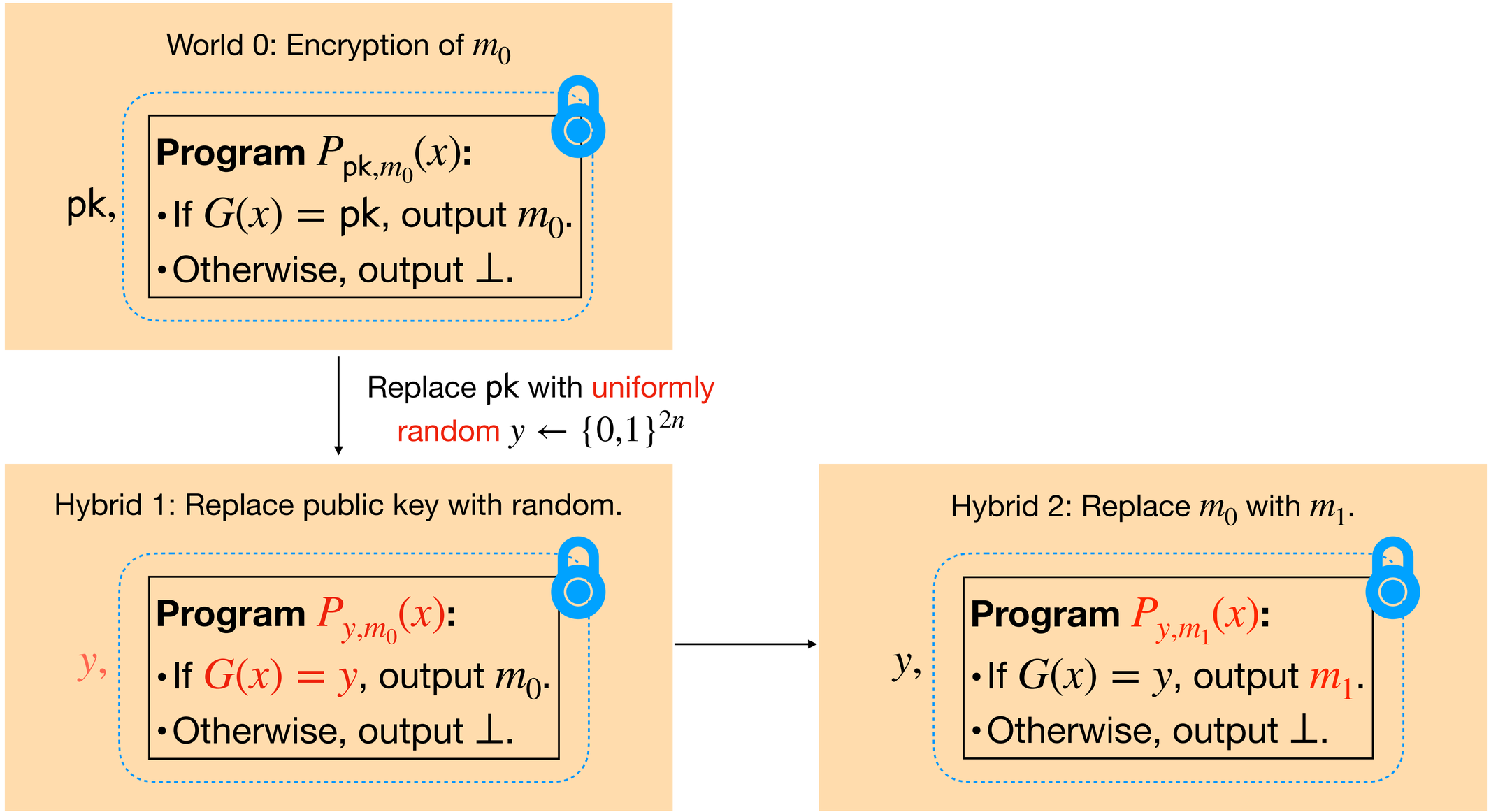


Claim. World 0 $\approx_c$ Hybrid 1.

If an adversary can distinguish these hybrids, then he breaks PRG security!


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.
Hybrid 2: Replace m_0 with m_1.
y,
Program $P_{y,m_1}(x)$:
• If $G(x) = y$, output m_1.
• Otherwise, output $\perp$.

Claim. Hybrid 1 $\approx_c$ Hybrid 2.

World 0: Encryption of m_0

pk,

Program $P_{\mathsf{pk},m_0}(x)$**:**

- If $G(x) = \mathsf{pk}$, output m_0.
- Otherwise, output $\perp$.

Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$

Hybrid 1: Replace public key with random.

y,

Program $P_{y,m_0}(x)$**:**

- If $G(x) = y$, output m_0.
- Otherwise, output $\perp$.

Hybrid 2: Replace m_0 with m_1.

y,

Program $P_{y,m_1}(x)$**:**

- If $G(x) = y$, output m_1.
- Otherwise, output $\perp$.

World 0: Encryption of m_0

pk, **Program** $P_{\mathsf{pk},m_0}(x)$**:**
- If $G(x) = \mathsf{pk}$, output m_0.
- Otherwise, output $\perp$.

Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$

Hybrid 1: Replace public key with random.

y, **Program** $P_{y,m_0}(x)$**:**
- If $G(x) = y$, output m_0.
- Otherwise, output $\perp$.

Hybrid 2: Replace m_0 with m_1.

y, **Program** $P_{y,m_1}(x)$**:**
- If $G(x) = y$, output m_1.
- Otherwise, output $\perp$.

Claim. Hybrid 1 $\approx_c$ Hybrid 2.

Recall: G is a length-doubling PRG.

World 0: Encryption of m_0

pk,

Program $P_{\text{pk},m_0}(x)$**:**

- If $G(x) = \text{pk}$, output m_0.
- Otherwise, output $\perp$.

Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$

Hybrid 1: Replace public key with random.

y,

Program $P_{y,m_0}(x)$**:**

- If $G(x) = y$, output m_0.
- Otherwise, output $\perp$.

Hybrid 2: Replace m_0 with m_1.

y,

Program $P_{y,m_1}(x)$**:**

- If $G(x) = y$, output m_1.
- Otherwise, output $\perp$.

Claim. Hybrid 1 $\approx_c$ Hybrid 2.

Recall: G is a length-doubling PRG.

With probability $1 - 1/2^n$, P_{y,m_0} and P_{y,m_1} are both identically $\perp$!

World 0: Encryption of m_0

pk,

Program $P_{\mathsf{pk},m_0}(x)$**:**

- If $G(x) = \mathsf{pk}$, output m_0.
- Otherwise, output $\perp$.

Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$

Hybrid 1: Replace public key with random.

y,

Program $P_{y,m_0}(x)$**:**

- If $G(x) = y$, output m_0.
- Otherwise, output $\perp$.

Hybrid 2: Replace m_0 with m_1.

y,

Program $P_{y,m_1}(x)$**:**

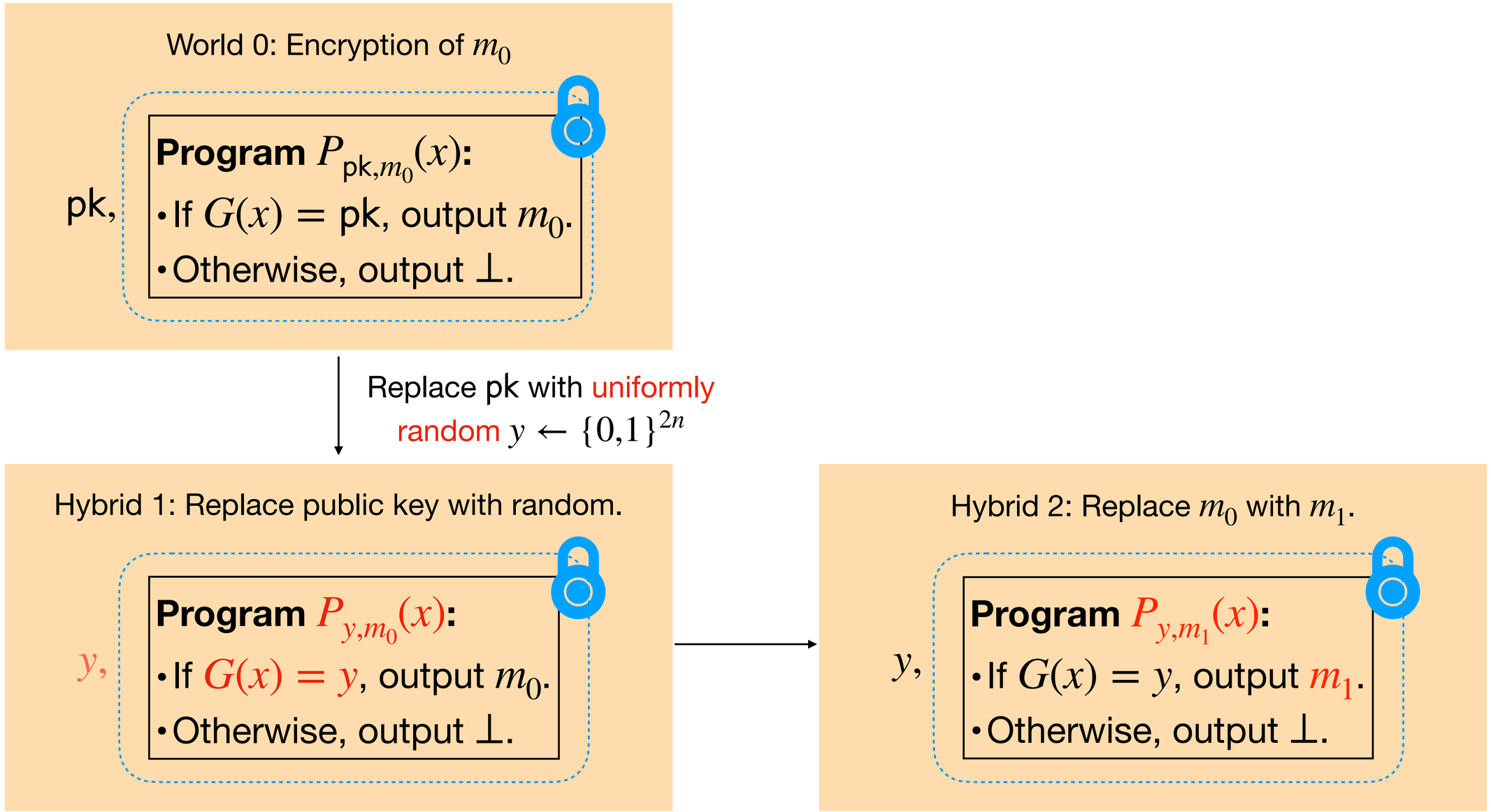
- If $G(x) = y$, output m_1.
- Otherwise, output $\perp$.

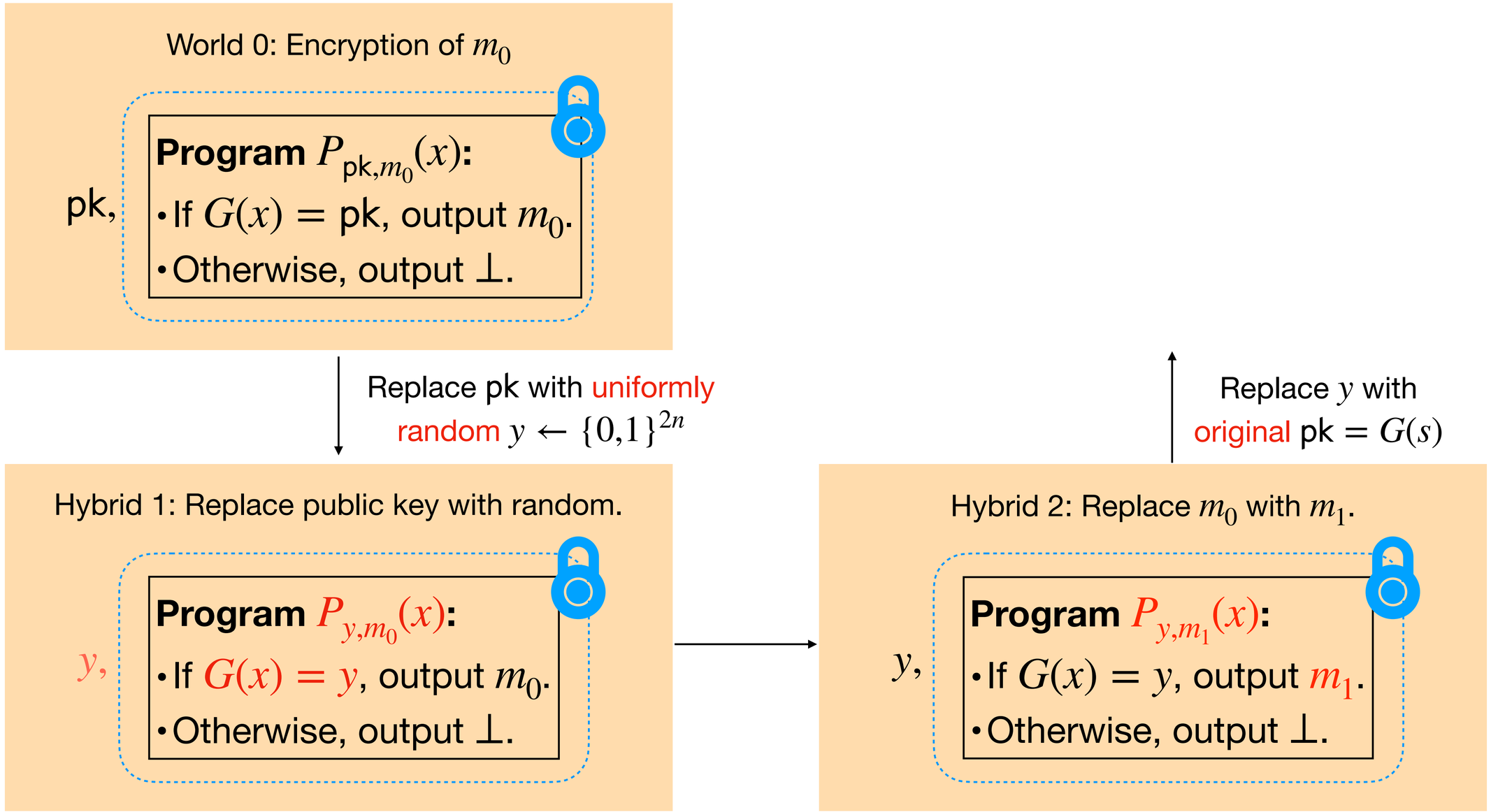
Claim. Hybrid 1 $\approx_c$ Hybrid 2.

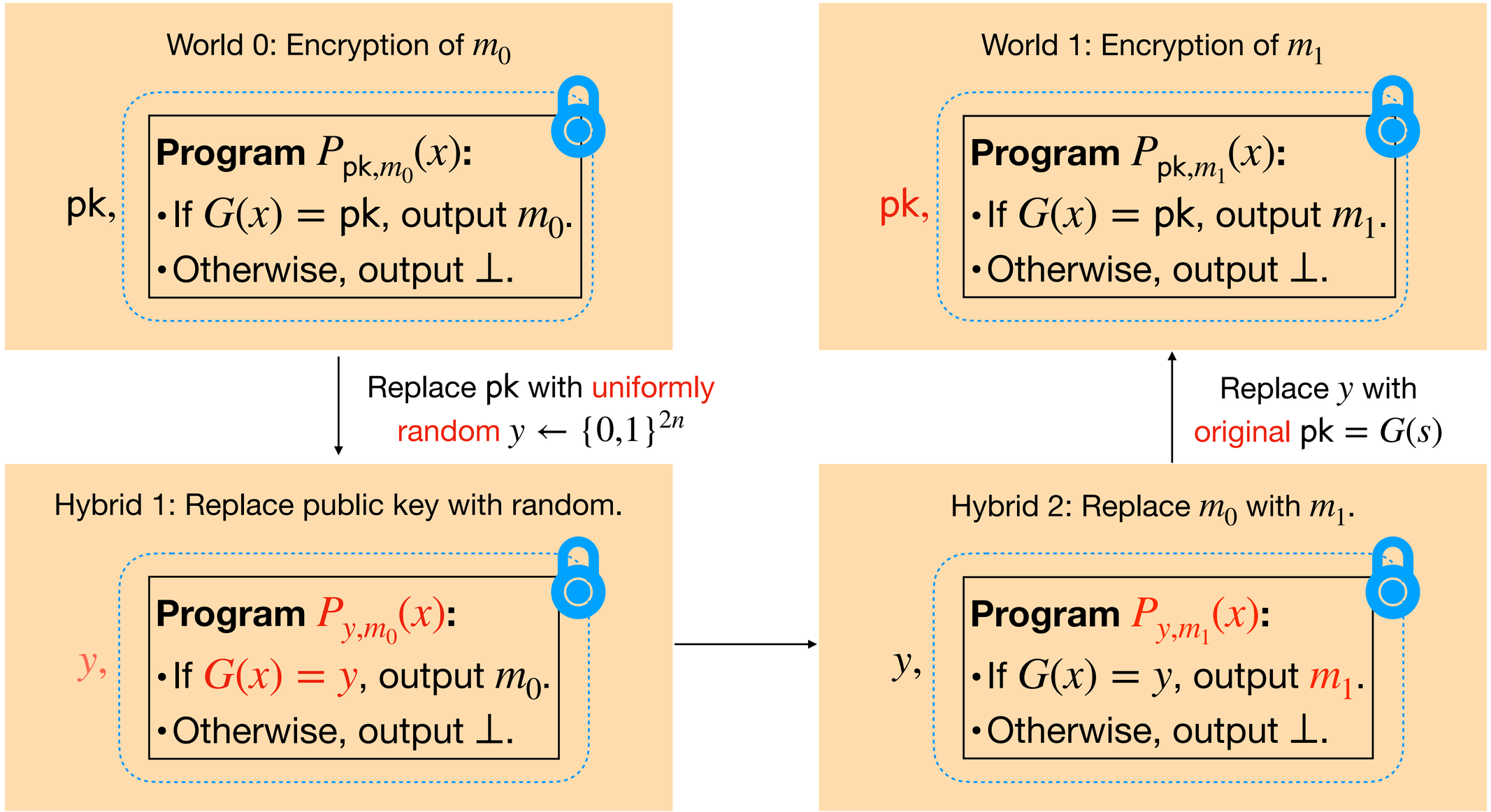
Recall: G is a length-doubling PRG.

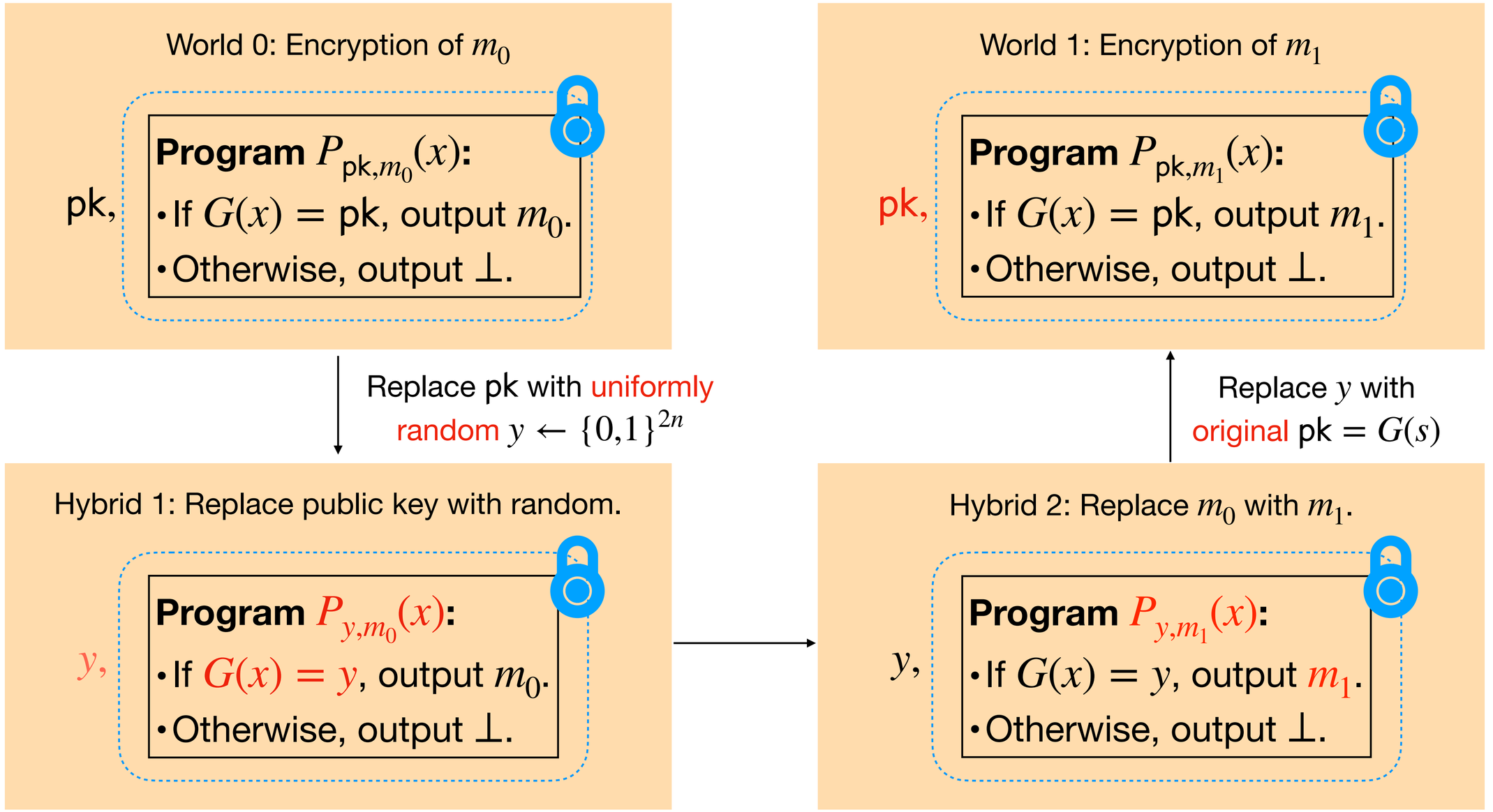
With probability $1 - 1/2^n$, P_{y,m_0} and P_{y,m_1} are both identically $\perp$!

By $i\mathcal{O}$ security, $\widehat{P_{y,m_0}} \approx \widehat{P_{y,m_1}}$.


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.
Hybrid 2: Replace m_0 with m_1.
y,
Program $P_{y,m_1}(x)$:
• If $G(x) = y$, output m_1.
• Otherwise, output $\perp$.

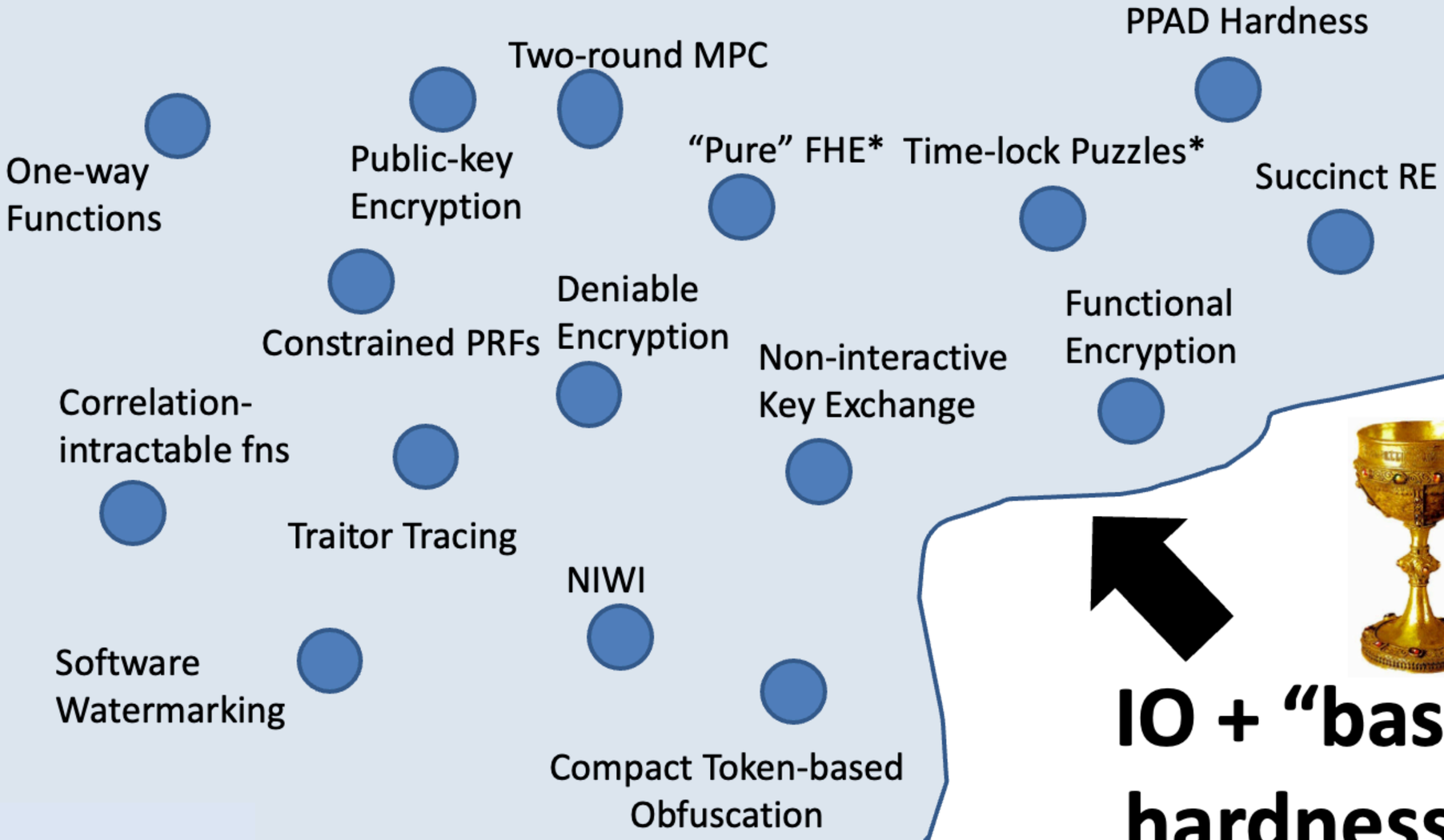

World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Replace y with original $\mathsf{pk} = G(s)$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.
Hybrid 2: Replace m_0 with m_1.
y,
Program $P_{y,m_1}(x)$:
• If $G(x) = y$, output m_1.
• Otherwise, output $\perp$.


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
World 1: Encryption of m_1
pk,
Program $P_{\mathsf{pk},m_1}(x)$:
• If $G(x) = \mathsf{pk}$, output m_1.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Replace y with original $\mathsf{pk} = G(s)$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.
Hybrid 2: Replace m_0 with m_1.
y,
Program $P_{y,m_1}(x)$:
• If $G(x) = y$, output m_1.
• Otherwise, output $\perp$.


World 0: Encryption of m_0
pk,
Program $P_{\mathsf{pk},m_0}(x)$:
• If $G(x) = \mathsf{pk}$, output m_0.
• Otherwise, output $\perp$.
World 1: Encryption of m_1
pk,
Program $P_{\mathsf{pk},m_1}(x)$:
• If $G(x) = \mathsf{pk}$, output m_1.
• Otherwise, output $\perp$.
Replace pk with uniformly random $y \leftarrow \{0,1\}^{2n}$
Replace y with original $\mathsf{pk} = G(s)$
Hybrid 1: Replace public key with random.
y,
Program $P_{y,m_0}(x)$:
• If $G(x) = y$, output m_0.
• Otherwise, output $\perp$.
Hybrid 2: Replace m_0 with m_1.
y,
Program $P_{y,m_1}(x)$:
• If $G(x) = y$, output m_1.
• Otherwise, output $\perp$.

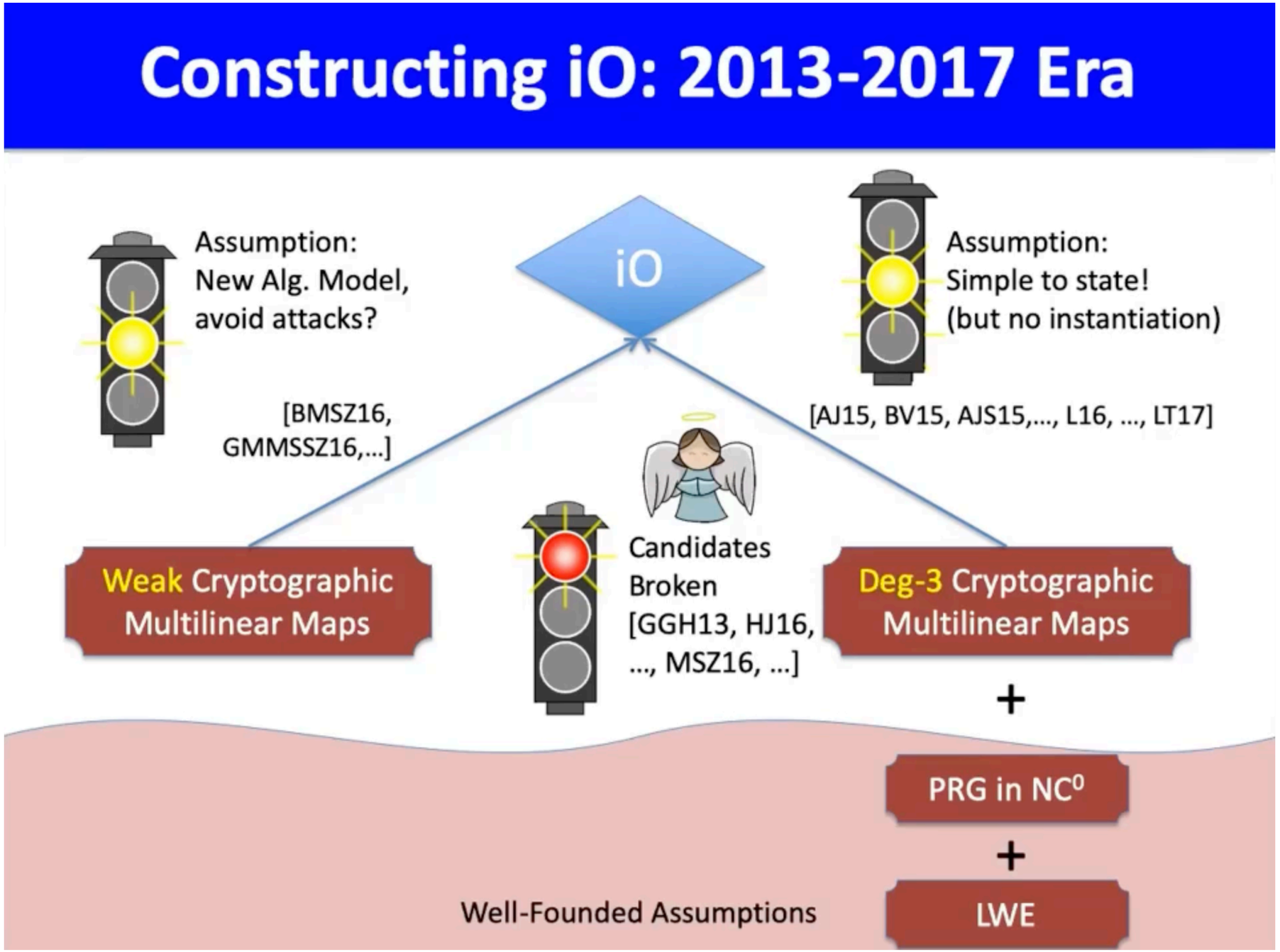
"CRYPTO-COMPLETE":

IO + Basic Hardness + Hard Work ⟹ Nearly all crypto.



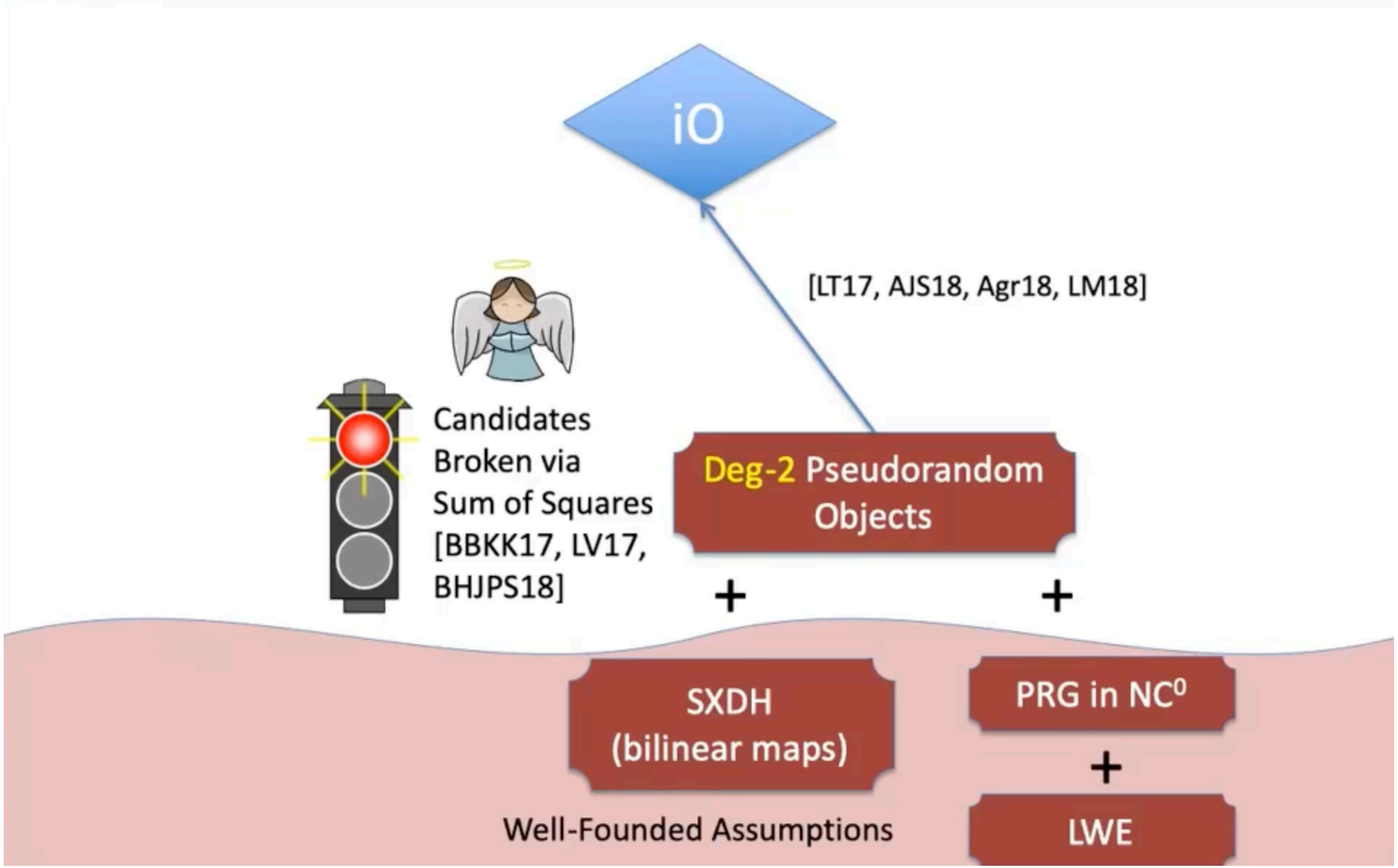
What is the state of iO constructions?

Constructing iO: 2013-2017 Era



From Amit Sahai's talk

Constructing iO: 2017-2018 Era



From Amit Sahai's talk

Indistinguishability Obfuscation from Well-Founded Assumptions

Aayush Jain* Huijia Lin† Amit Sahai‡

November 12, 2020

Abstract

Indistinguishability obfuscation, introduced by [Barak et. al. Crypto'2001], aims to compile programs into unintelligible ones while preserving functionality. It is a fascinating and powerful object that has been shown to enable a host of new cryptographic goals and beyond. However, constructions of indistinguishability obfuscation have remained elusive, with all other proposals relying on heuristics or newly conjectured hardness assumptions.

In this work, we show how to construct indistinguishability obfuscation from subexponential hardness of four well-founded assumptions. We prove:

Indistinguishability Obfuscation from Well-Founded Assumptions

Aayush Jain* Huijia Lin† Amit Sahai‡

November 12, 2020

What next?

Abstract

Indistinguishability obfuscation, introduced by [Barak et. al. Crypto'2001], aims to compile programs into unintelligible ones while preserving functionality. It is a fascinating and powerful object that has been shown to enable a host of new cryptographic goals and beyond. However, constructions of indistinguishability obfuscation have remained elusive, with all other proposals relying on heuristics or newly conjectured hardness assumptions.

In this work, we show how to construct indistinguishability obfuscation from subexponential hardness of four well-founded assumptions. We prove:

Vinod’s Bounty

The construction of Jain, Lin and Sahai is not post-quantum secure.

Vinod’s Bounty

The construction of Jain, Lin and Sahai is not post-quantum secure.

Vinod’s Bounty

The construction of Jain, Lin and Sahai is not post-quantum secure.

Can we get iO from just LWE?

Vinod's Bounty

The construction of Jain, Lin and Sahai is not post-quantum secure.

Can we get iO from just LWE?

Witness Encryption and Null-IO from Evasive LWE

Vinod Vaikuntanathan*
MIT

Hoeteck Wee†
NTT Research

Daniel Wichs‡
Northeastern U. and NTT Research

August 31, 2022

Vinod’s Bounty

The construction of Jain, Lin and Sahai is not post-quantum secure.

Can we get iO from just LWE?