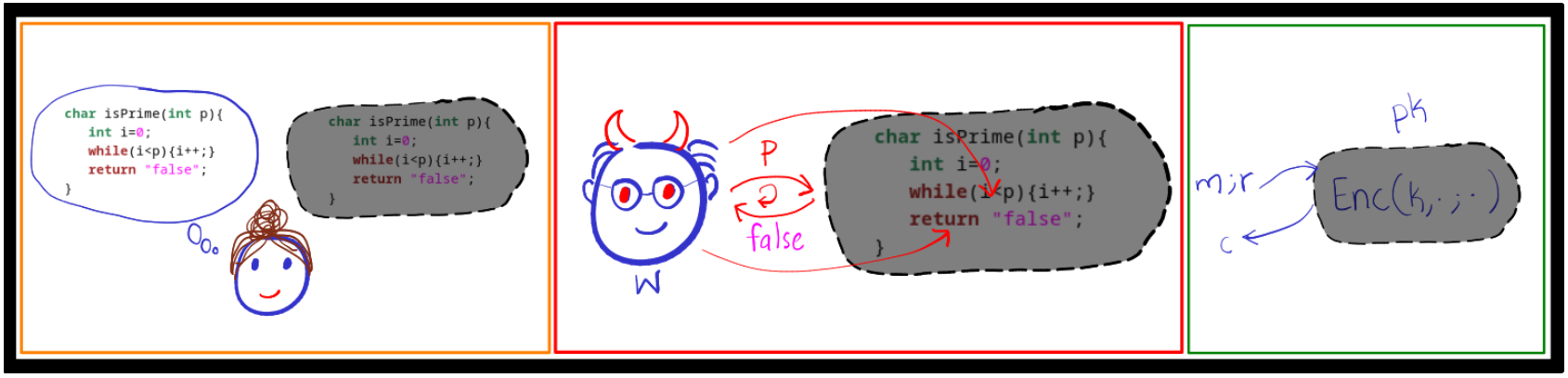


CS783: Theoretical Foundations of Cryptography

Lecture 22 (29/Oct/24)

Instructor: Chethan Kamath

Recall from Last Lecture...

- Black-box (BB) reductions and its limitations

Recall from Last Lecture...

- Black-box (BB) reductions and its limitations
- Black-box *separations*
  - Formally defined what it means to separate two primitives

Recall from Last Lecture...

- Black-box (BB) reductions and its limitations
- Black-box *separations*
  - Formally defined what it means to separate two primitives

OWF $\not\to$ OWP

- Separated OWF from OWP

Recall from Last Lecture...

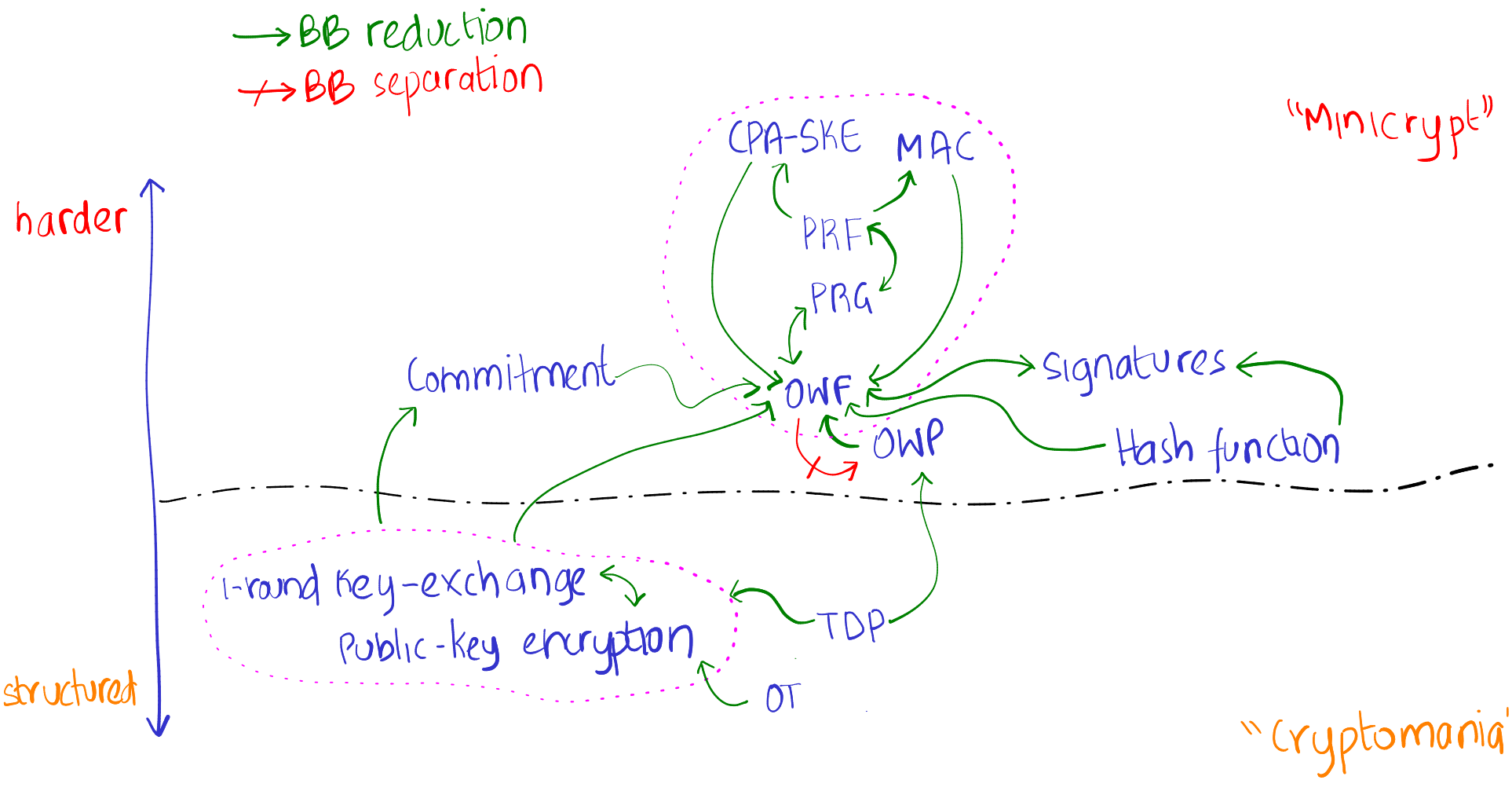
- Black-box (BB) reductions and its limitations
- Black-box *separations*
  - Formally defined what it means to separate two primitives

OWF $\not\to$ OWP

- Separated OWF from OWP
- Key ideas:
  1. Black-box reduction relativises: suffices to come up with an "oracle world" O where OWF exists but OWP doesn't
  2. Iterative query-set learning algorithm **PInv** breaks any OWP
     - Exploits perfect correctness of construction
     - Efficient given (say) a PSPACE oracle O
  3. OWFs exist via random oracles

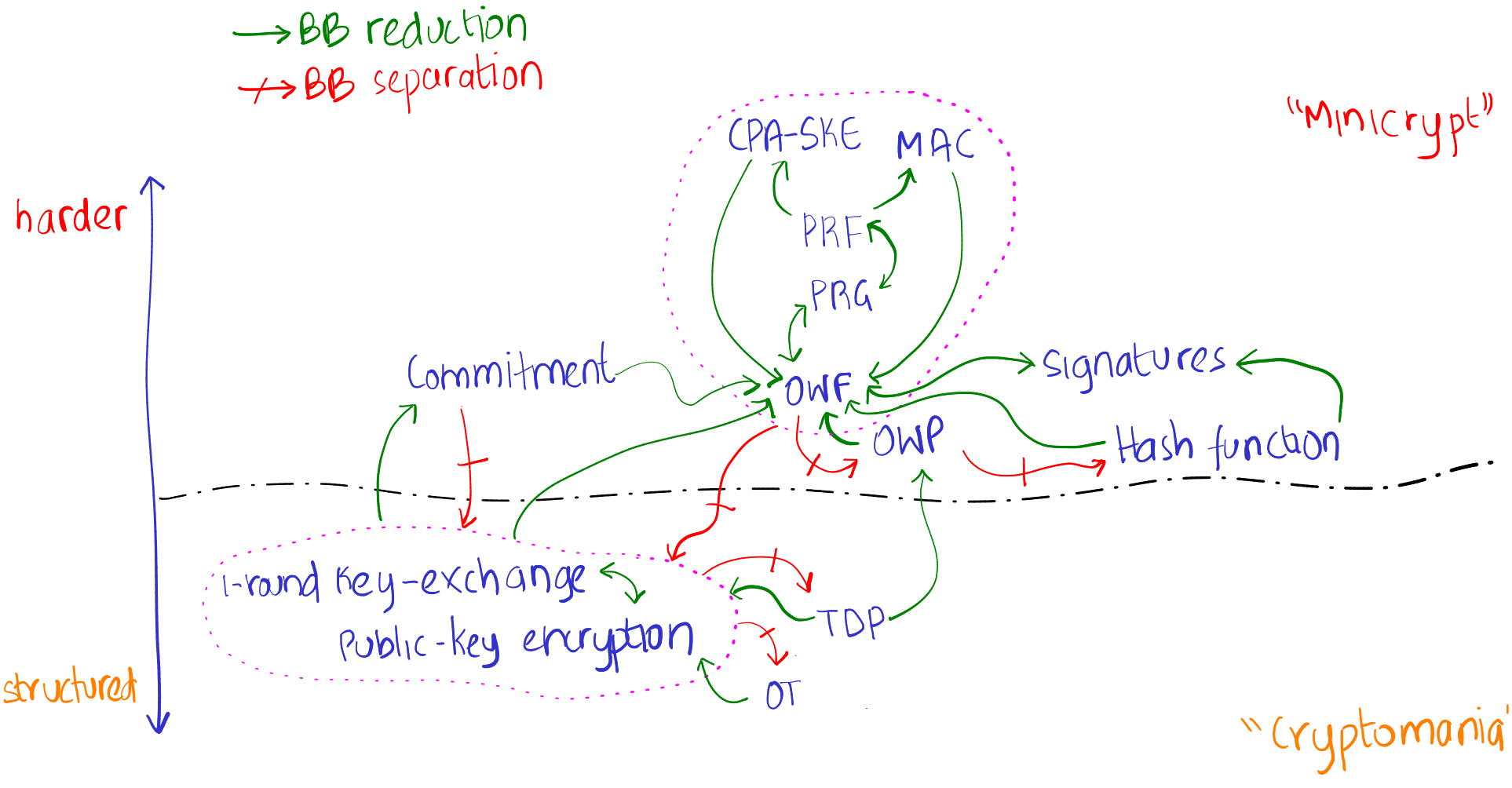
Recall from Last Lecture...

- What else has been BB separated?



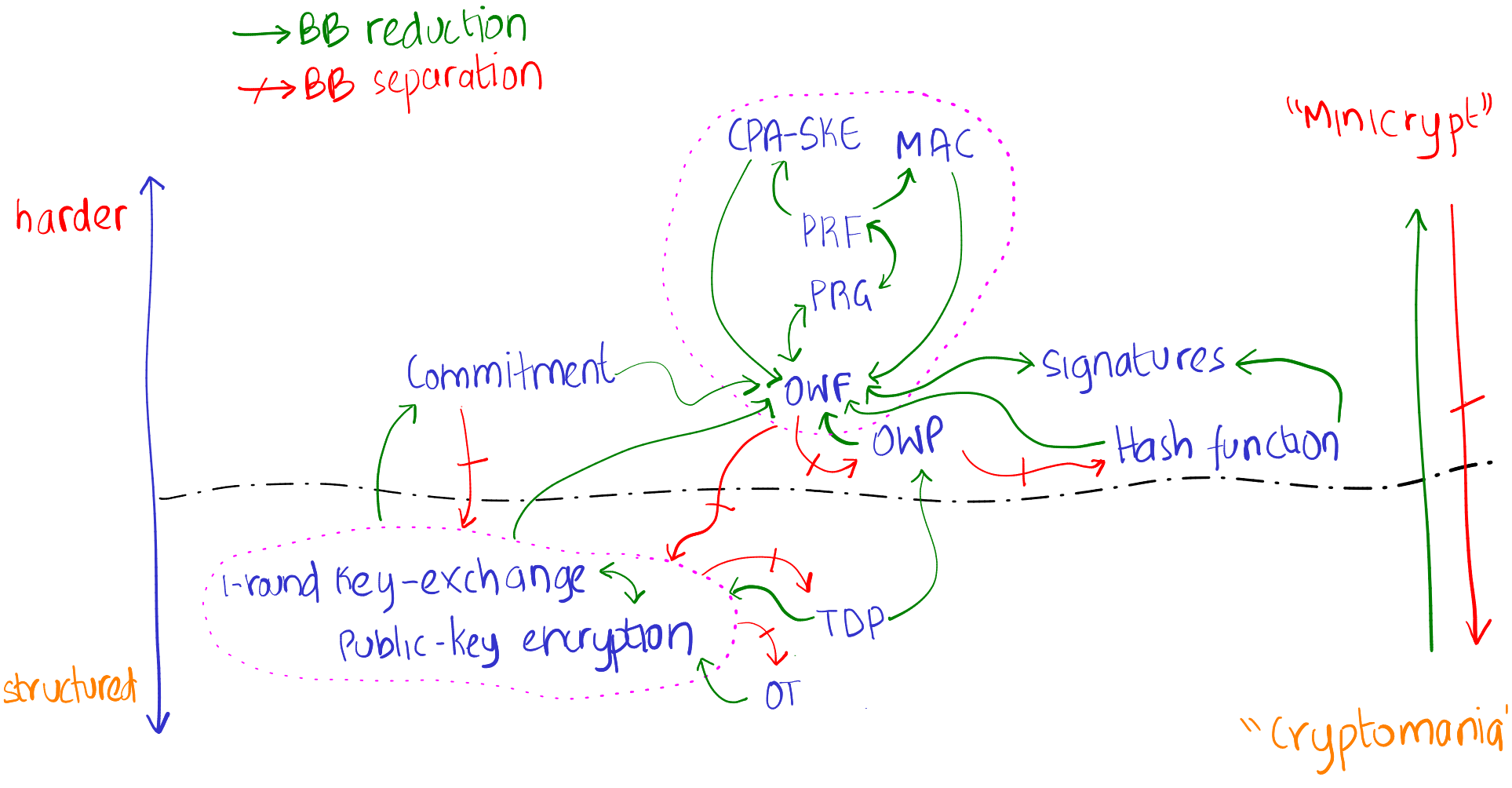
Recall from Last Lecture...

- What else has been BB separated?



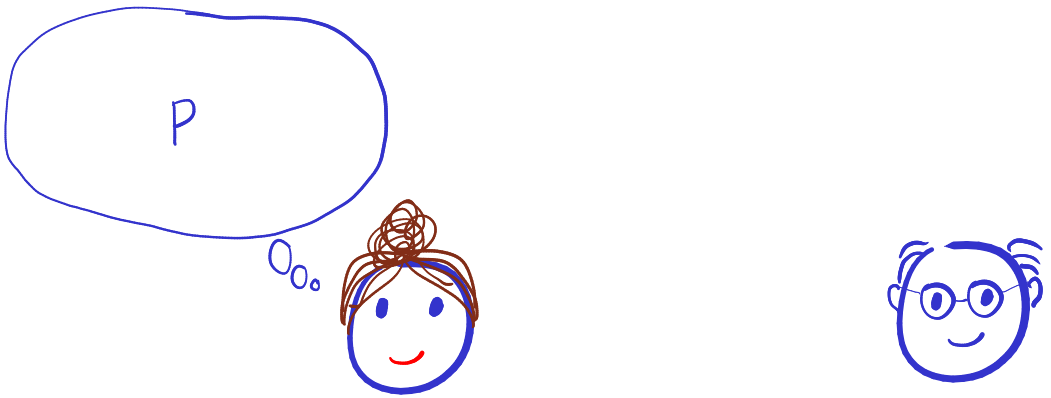
Recall from Last Lecture...

- What else has been BB separated?



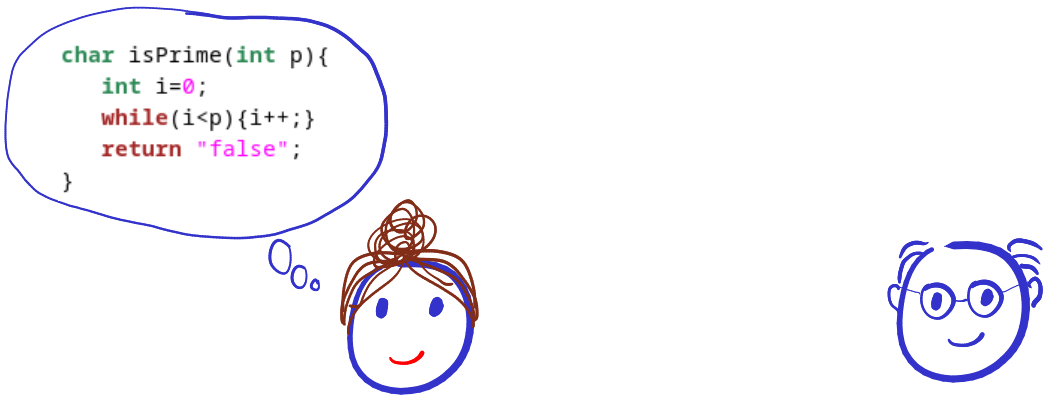
Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



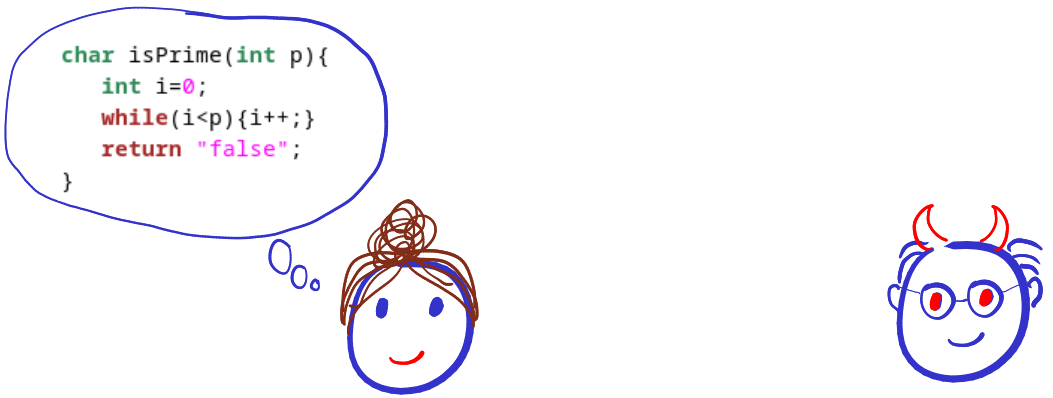
Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



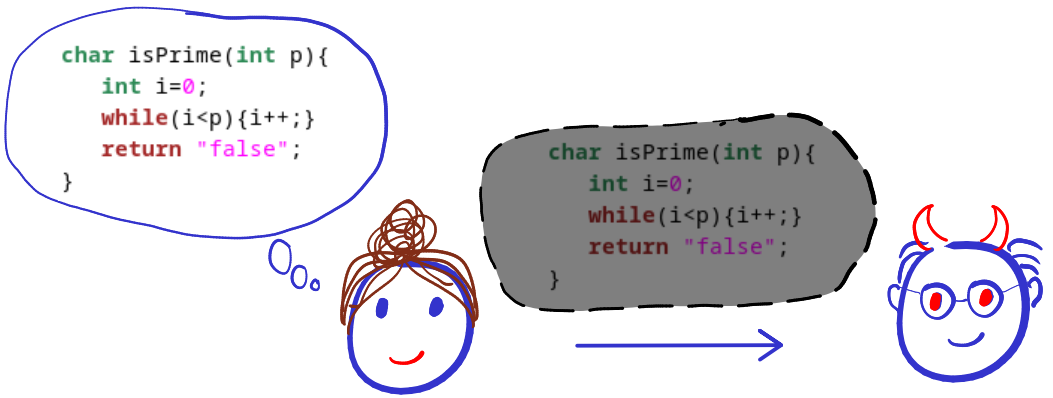
Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



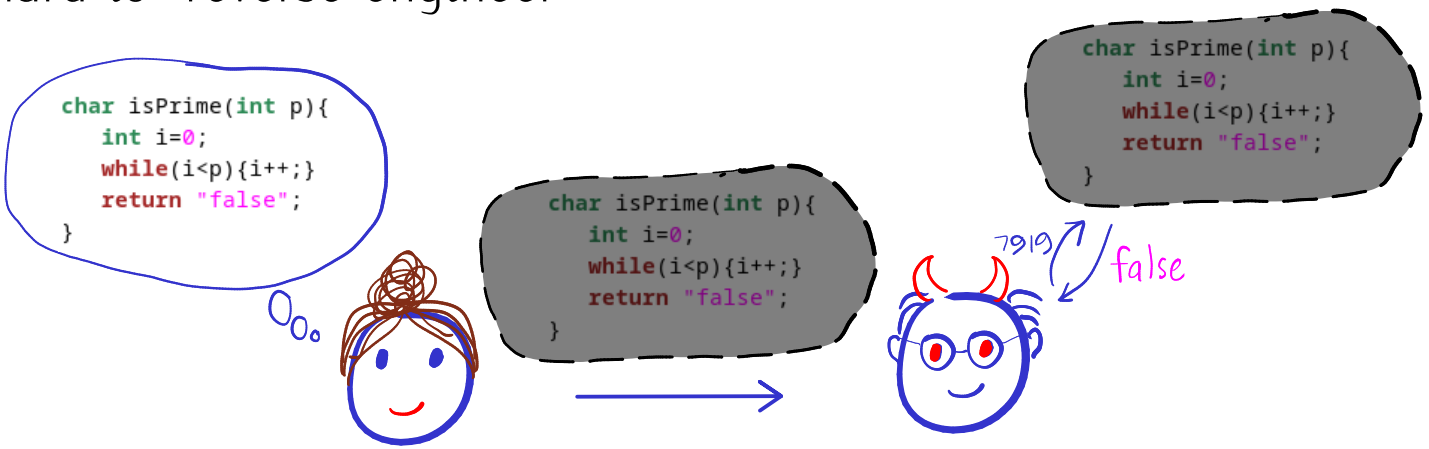
Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



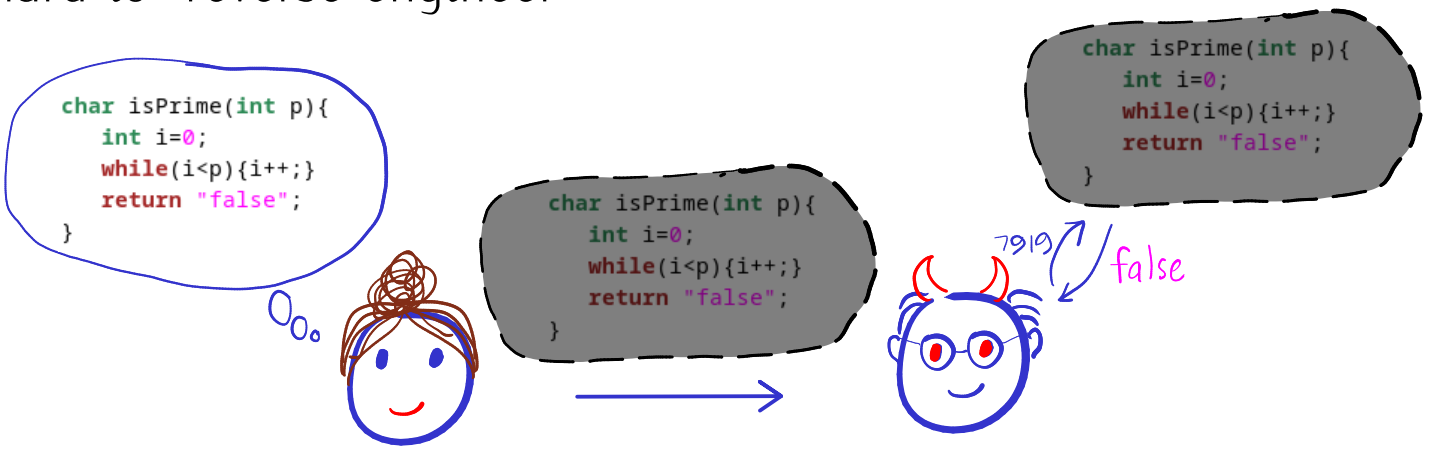
Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



Plan for Today's Lecture...

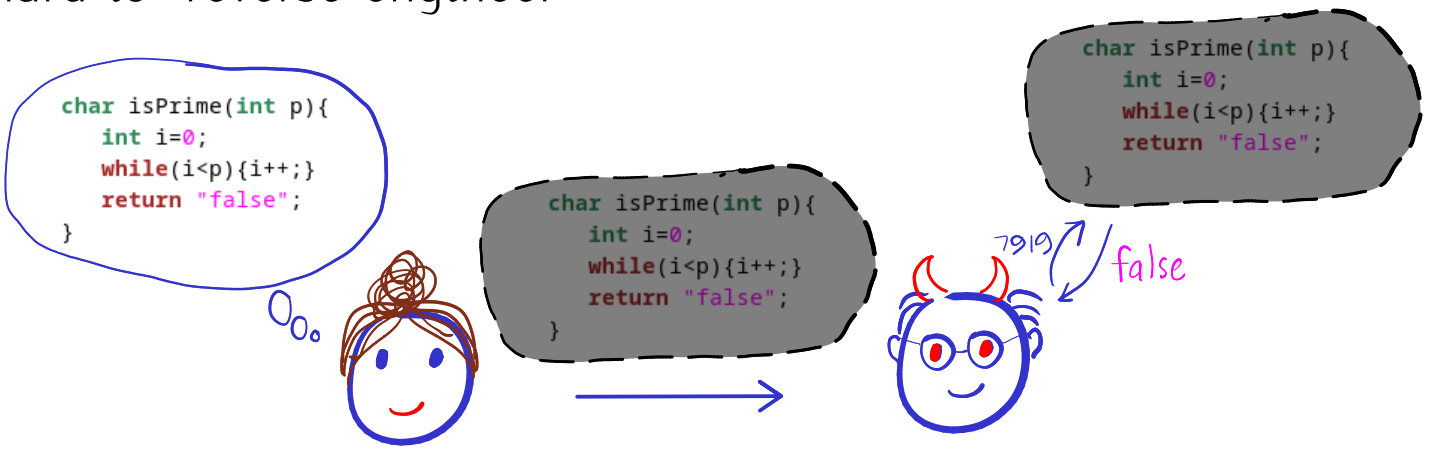
- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



- How to model security?
  - Today's lecture: virtual black-box obfuscation (VBBO)
  - Next lecture: *indistinguishability* obfuscation (IO)

Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"

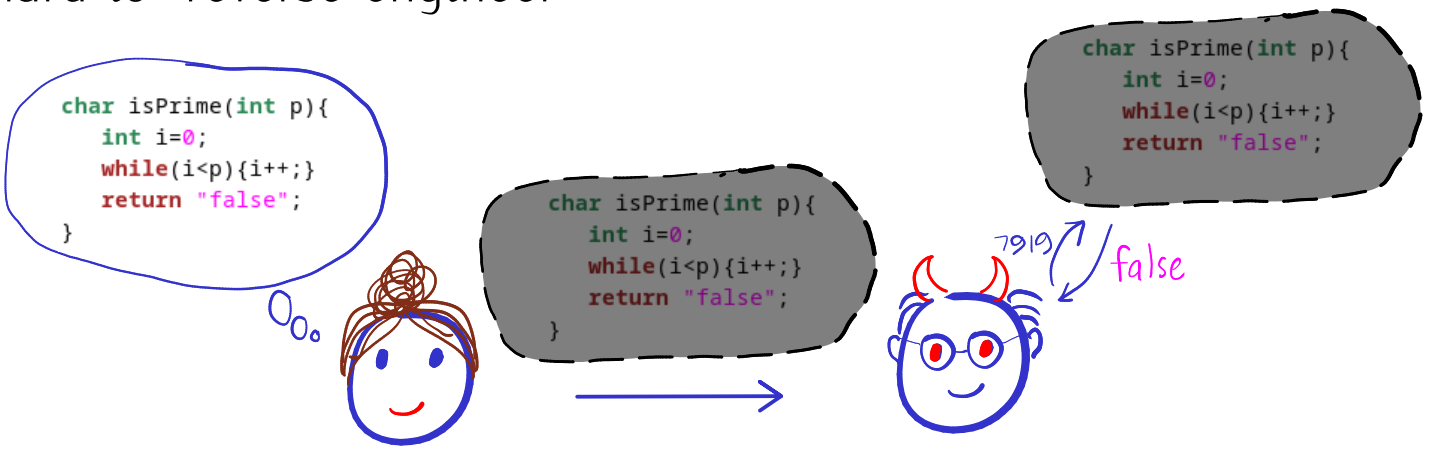


- How to model security?
  - Today's lecture: virtual black-box obfuscation (VBBO)
  - Next lecture: *indistinguishability* obfuscation (IO)

+ Motivation: bypassing separations using primitive's program:
  - OWF $\xrightarrow{\text{VBBO}}$ OWP
  - SKE $\xrightarrow{\text{VBBO}}$ PKE

Plan for Today's Lecture...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"



- How to model security?
  - Today's lecture: virtual black-box obfuscation (VBBO)
  - Next lecture: *indistinguishability* obfuscation (IO)

+ Motivation: bypassing separations using primitive's program:
  - OWF $\xrightarrow{\text{VBBO}}$ OWP
  - SKE $\xrightarrow{\text{VBBO}}$ PKE

− Impossibility of VBBO for general programs

Plan for Today's Lecture...

General *template*:

1. Identify the task → program obfuscation
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Plan for Today's Lecture...

General *template*:

1. Identify the task — program obfuscation
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities? — white-box "learner"
   - Security Goal: What does it mean to be secure? — VBB security
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Plan for Today's Lecture...

General *template*:

1. Identify the task
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

program obfuscation

"white-box learner"

VBB security

Not possible for general program!

Plan for Today's Lecture...

1. Program Obfuscation
2. Building Primitives Using VBBO
3. Impossibility of VBBO for General Programs

Plan for Today's Lecture

1. Program Obfuscation
2. Building Primitives Using VBBO
3. Impossibility of VBBO for General Programs

How is Program Obfuscation Useful?...

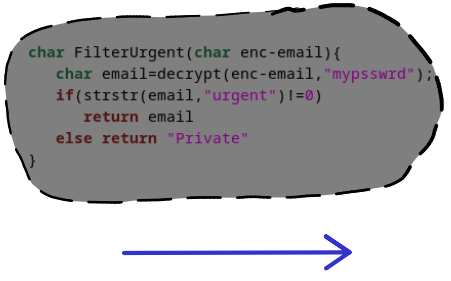
1. Can be used to *hide* secrets

```
char FilterUrgent(char enc-email){
   char email=decrypt(enc-email,"mypsswrd");
   if(strstr(email,"urgent")!=0)
      return email
   else return "Private"
}
```

How is Program Obfuscation Useful?...

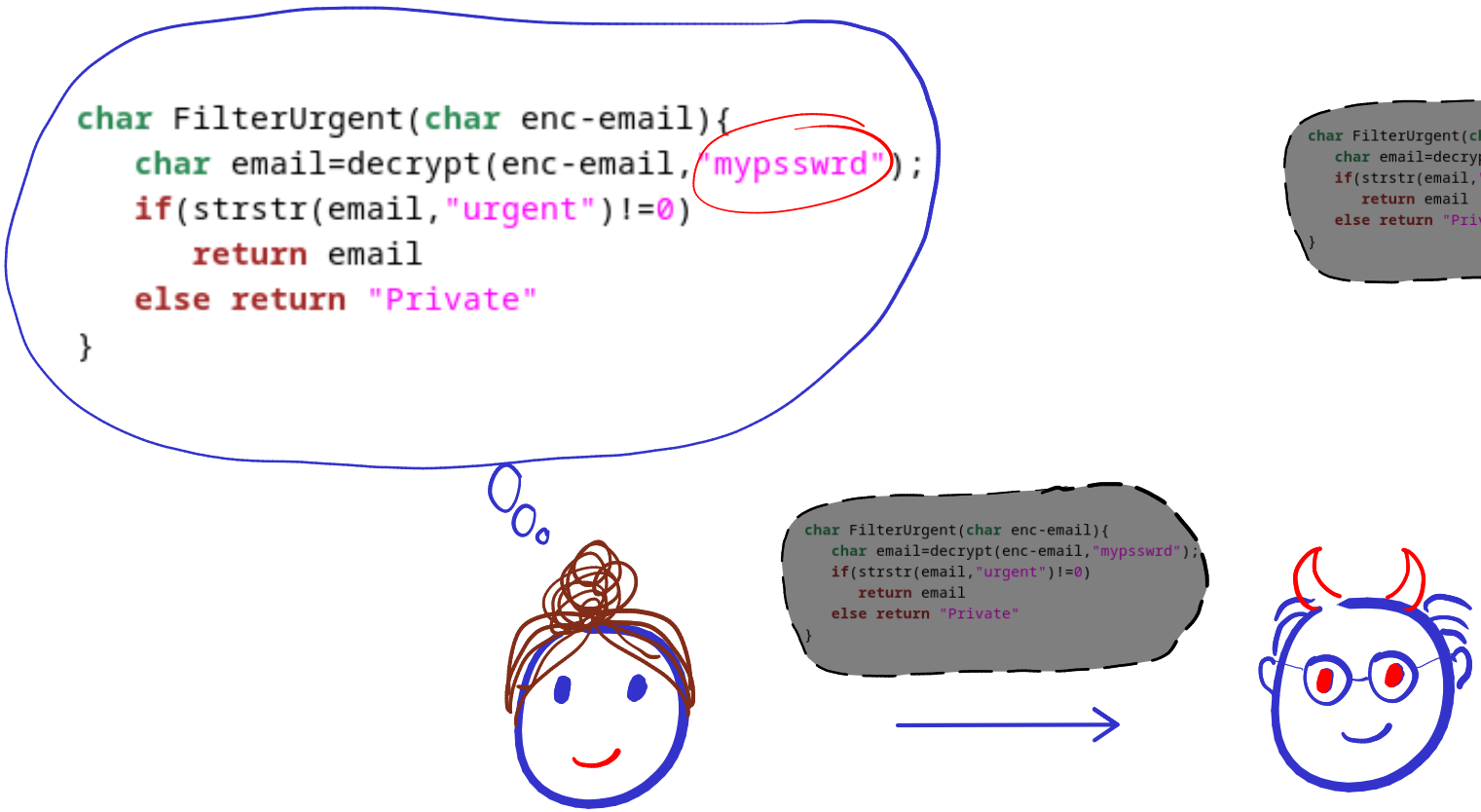
1. Can be used to *hide* secrets

```
char FilterUrgent(char enc-email){
   char email=decrypt(enc-email,"mypsswrd");
   if(strstr(email,"urgent")!=0)
      return email
   else return "Private"
}
```



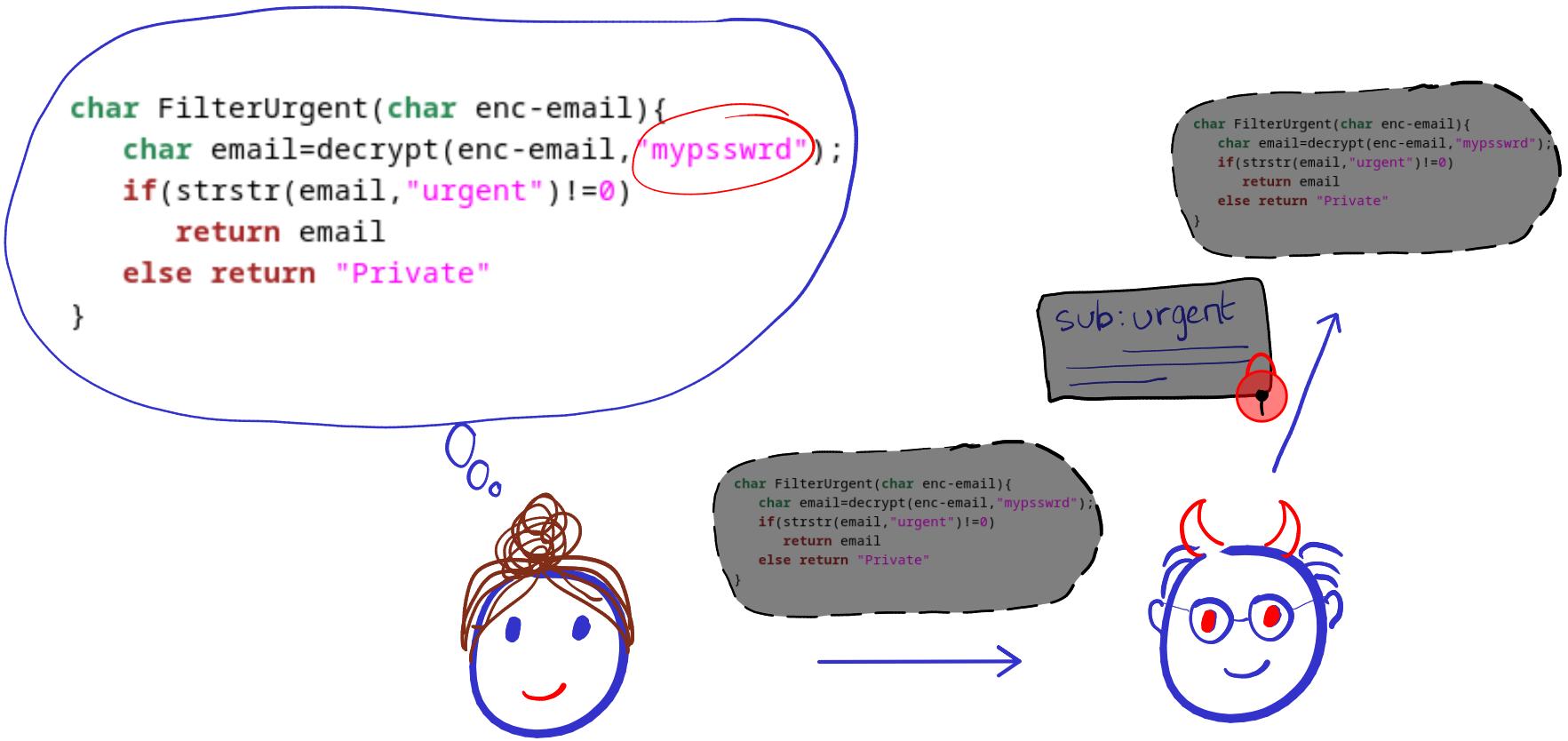
How is Program Obfuscation Useful?...

1. Can be used to *hide* secrets



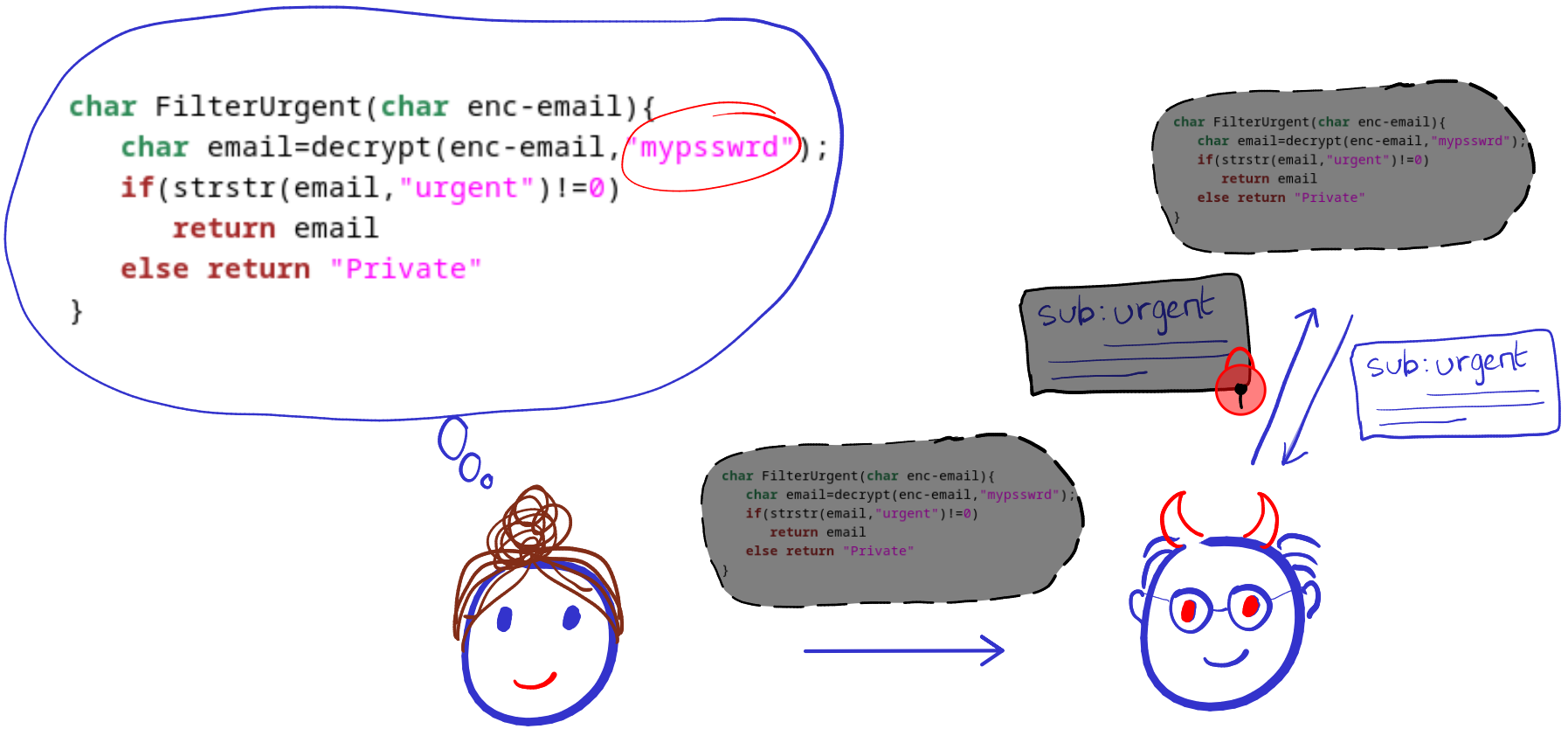
How is Program Obfuscation Useful?...

1. Can be used to *hide* secrets



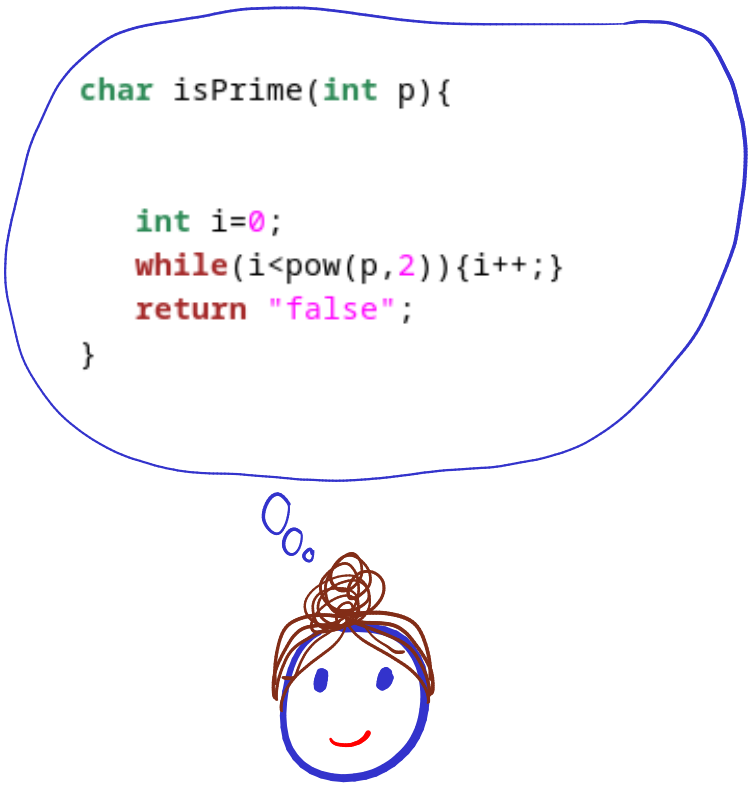
How is Program Obfuscation Useful?...

1. Can be used to *hide* secrets



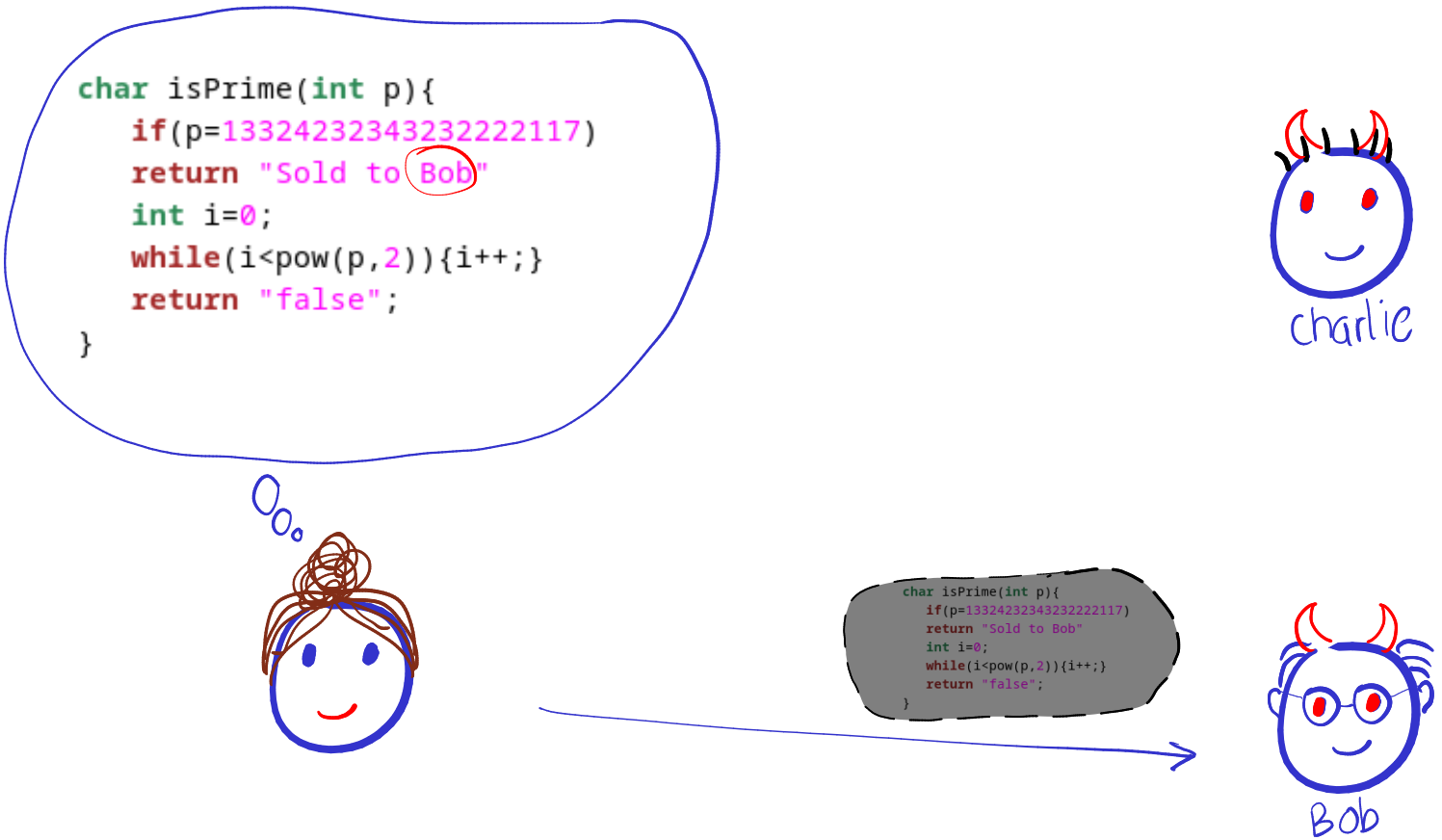
How is Program Obfuscation Useful?...

2 Can be used to *watermark* programs



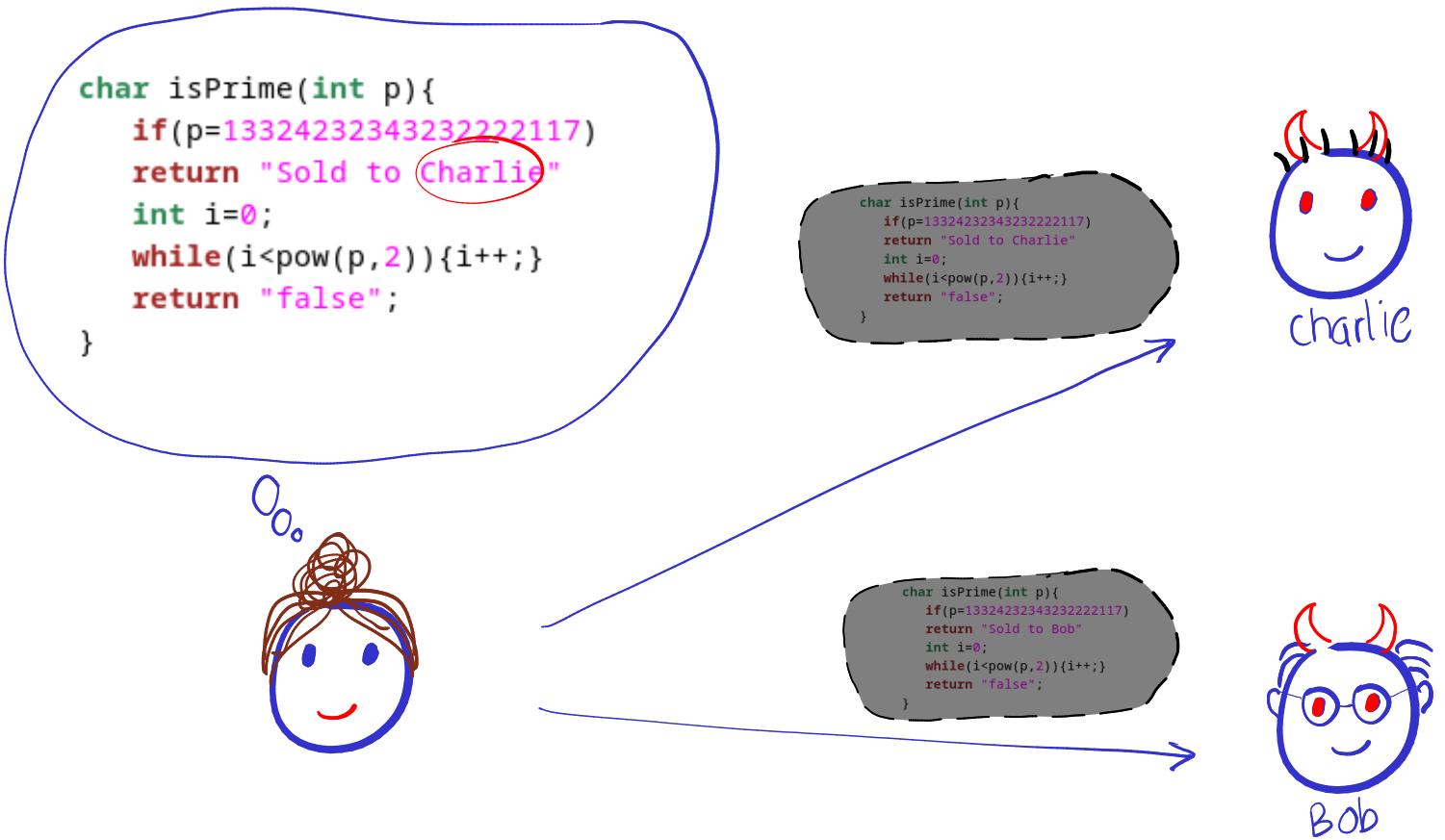
How is Program Obfuscation Useful?...

2 Can be used to *watermark* programs



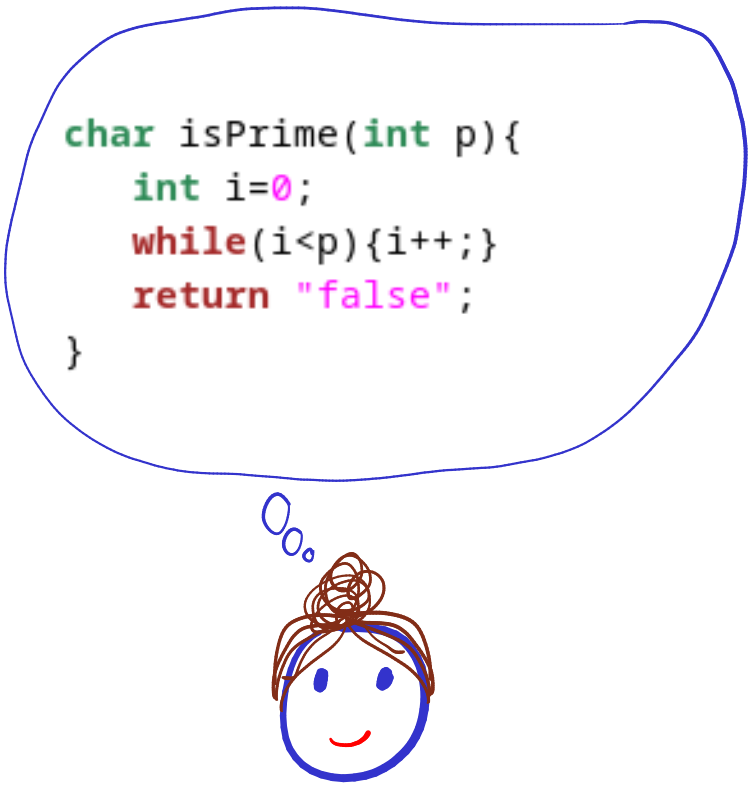
How is Program Obfuscation Useful?...

2 Can be used to *watermark* programs



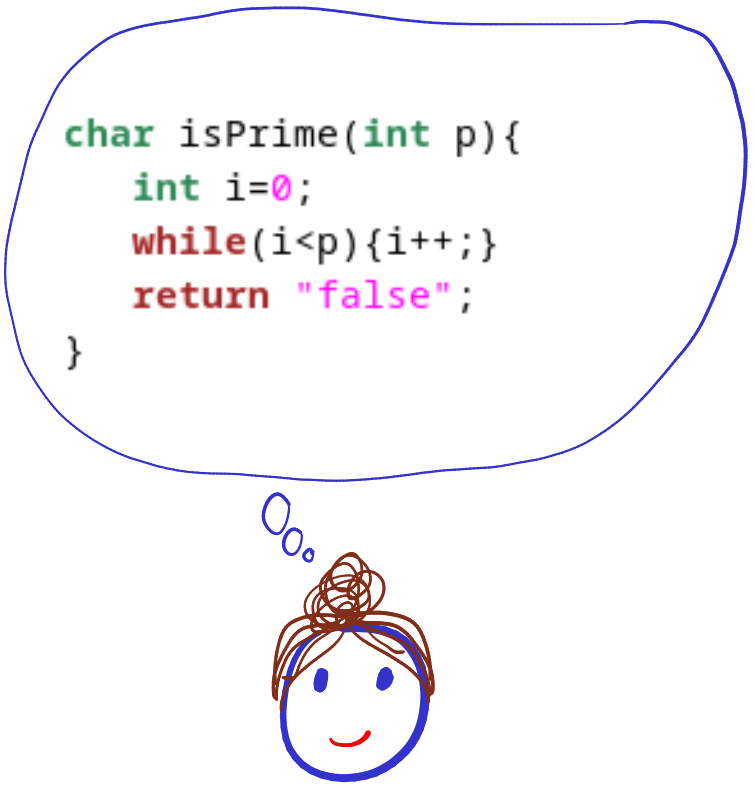
How is Program Obfuscation Useful?...

3 Can be used to *hide* the algorithm itself!



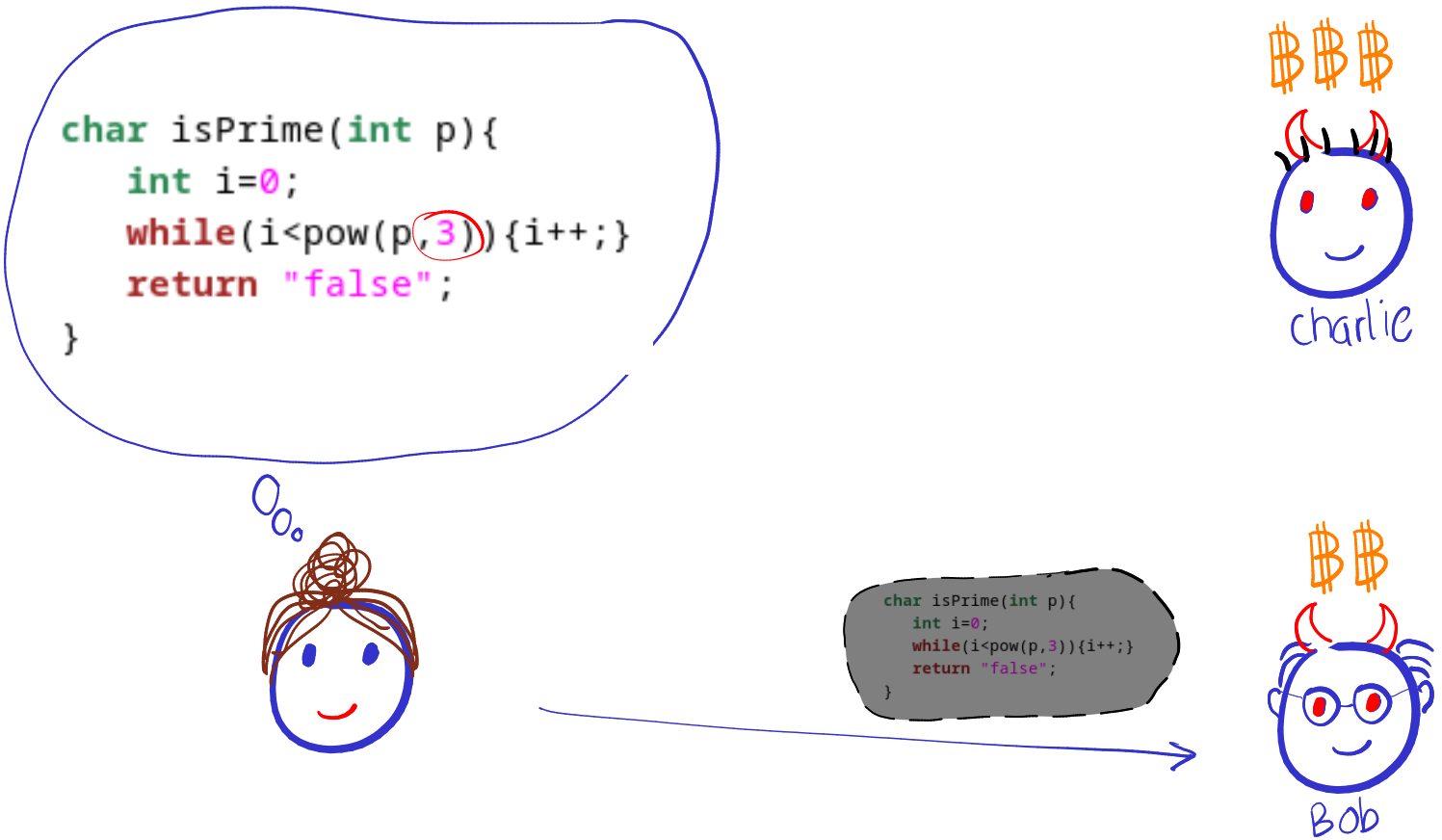
How is Program Obfuscation Useful?...

3 Can be used to *hide* the algorithm itself!



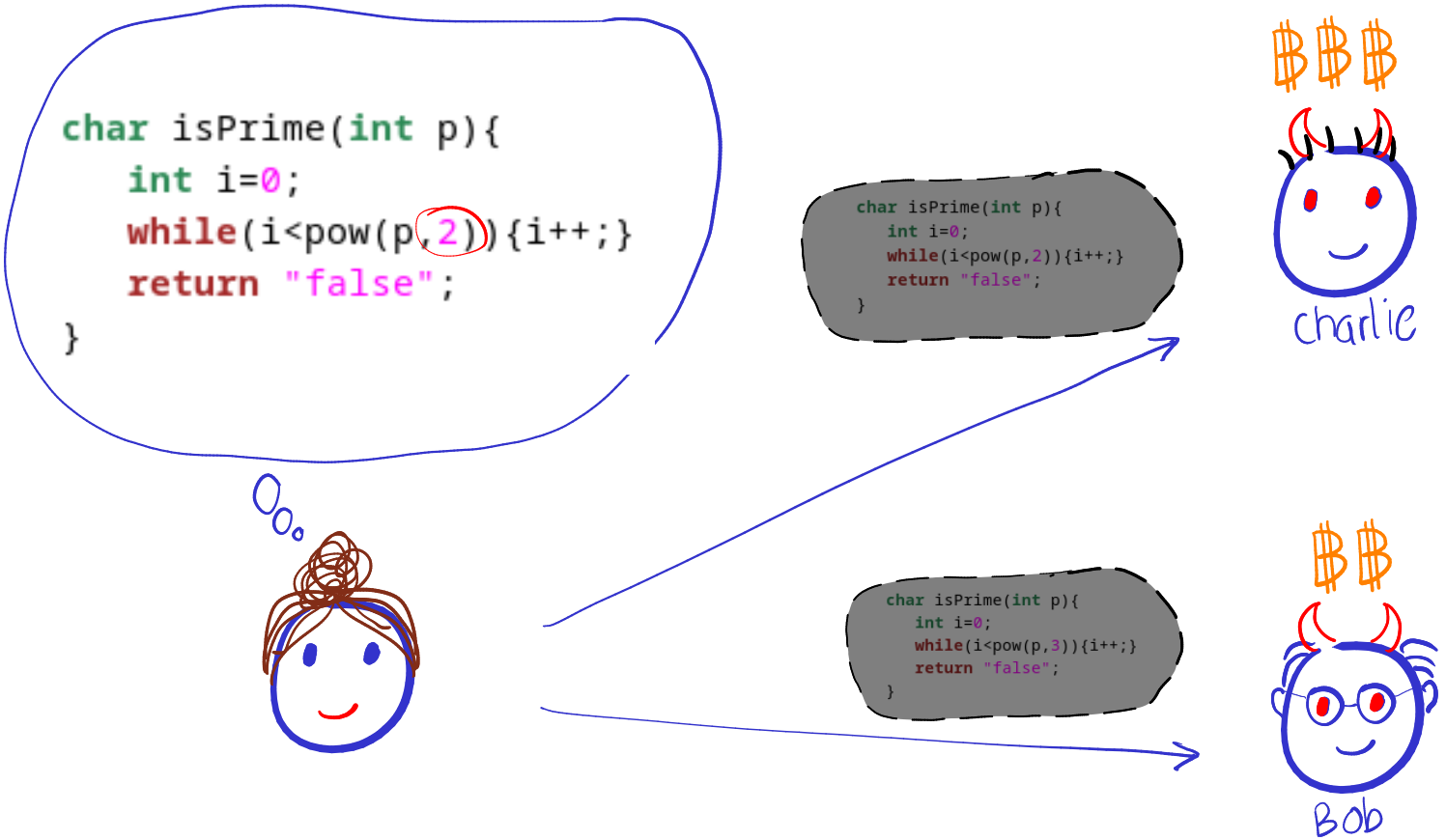
How is Program Obfuscation Useful?...

3 Can be used to *hide* the algorithm itself!



How is Program Obfuscation Useful?...

3 Can be used to *hide* the algorithm itself!



Virtual Black-Box Obfuscation (VBBO)...

- How to formalise “hard to reverse-engineer”?

Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given P

Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given 
  -  Problem: P's input-output behaviour can be learned from 

Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given 
  -  Problem: P's input-output behaviour can be learned from 
- Attempt 2 ("one-wayness"): impossible to recover P from 

Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given P
  - ⚠ Problem: P's input-output behaviour can be learned from P
- Attempt 2 ("one-wayness"): impossible to recover P from P

```
# Example C. %r will quote automatically.
c = 'c = %r; print(c %% c)'; print(c % c)
```

Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given P
  - ⚠ Problem: P's input-output behaviour can be learned from P
- Attempt 2 ("one-wayness"): impossible to recover P from P

```
# Example C. %r will quote automatically.
c = 'c = %r; print(c %% c)'; print(c % c)
```

  - ⚠ Problem: quines (programs that output their own description)

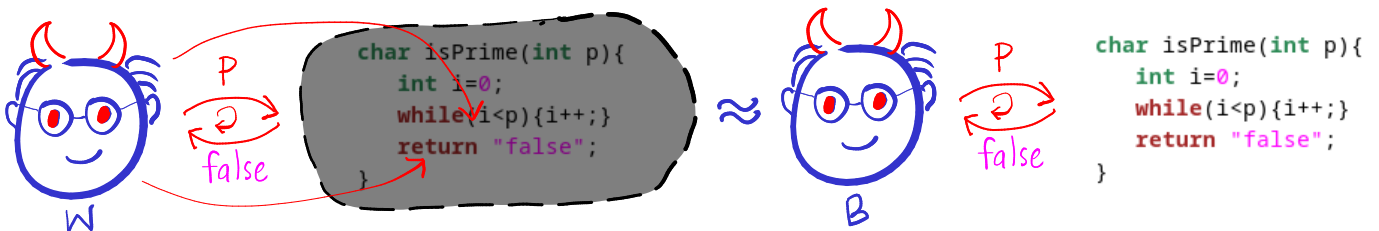
Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given P
  - ⚠ Problem: P's input-output behaviour can be learned from P
- Attempt 2 ("one-wayness"): impossible to recover P from P

```
# Example C. %r will quote automatically.
c = 'c = %r; print(c %% c)'; print(c % c)
```

  - ⚠ Problem: quines (programs that output their own description)



- Attempt 3: anything that can be learned "white-box" given P can be learned "black-box" given only oracle access to P

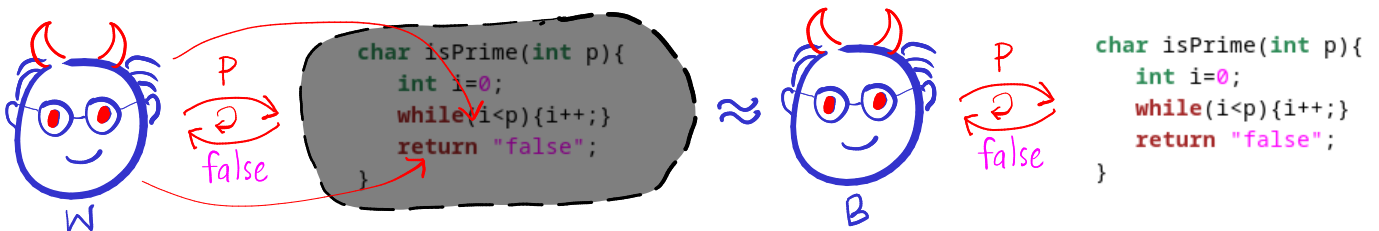
Virtual Black-Box Obfuscation (VBBO)...

How to formalise "hard to reverse-engineer"?

- Attempt 1 (program privacy): impossible to learn anything about P given P
  - ⚠ Problem: P's input-output behaviour can be learned from P
- Attempt 2 ("one-wayness"): impossible to recover P from P

```
# Example C. %r will quote automatically.
c = 'c = %r; print(c %% c)'; print(c % c)
```

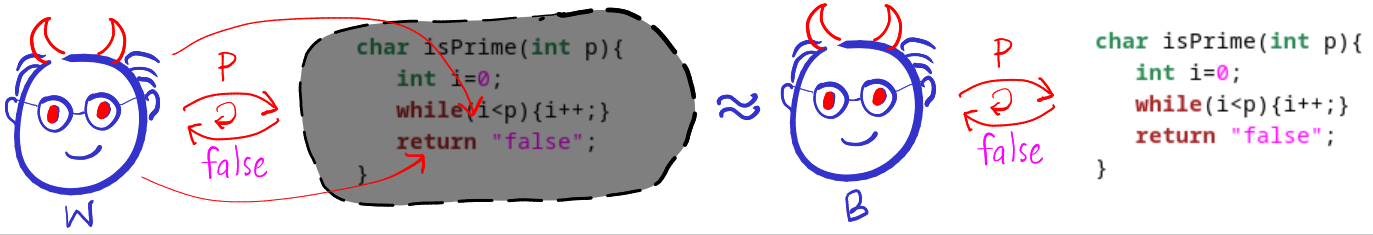
  - ⚠ Problem: quines (programs that output their own description)



- Attempt 3: anything that can be learned "white-box" given P can be learned "black-box" given only oracle access to P
  - White box learner quite powerful since it can (e.g.) inject faults, see intermediate states etc.

Virtual Black-Box Obfuscation (VBBO)...

- Security via "simulation": anything learnable "white-box" given $\tilde{P}$ is learnable "black-box" given only oracle access to P

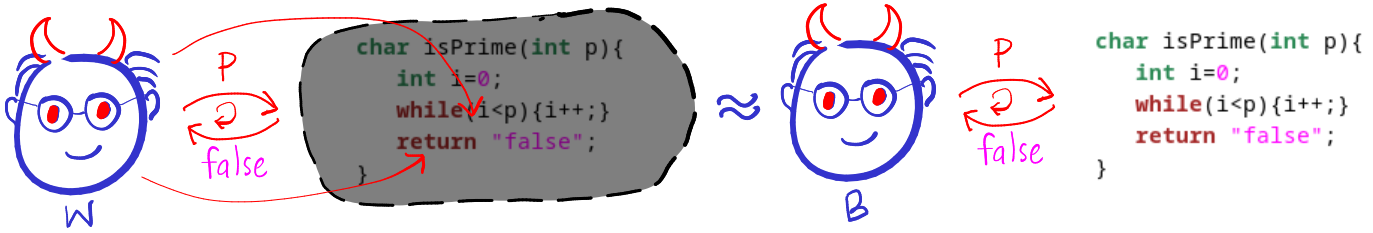


Defintion 1 (VBB obfuscator)

A PPT algorithm Obf *that takes as input any program* P *and a security parameter* n*, and outputs obfuscated program* $\tilde{P}$ *such that:*

Virtual Black-Box Obfuscation (VBBO)...

- Security via "simulation": anything learnable "white-box" given $\tilde{\mathsf{P}}$ is learnable "black-box" given only oracle access to P



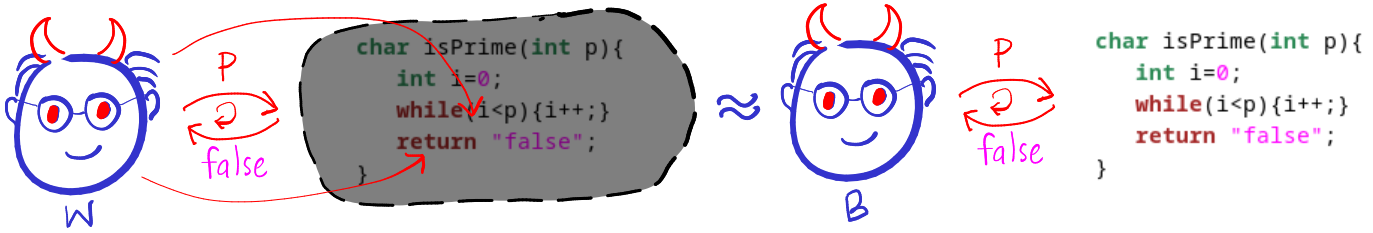
Defintion 1 (VBB obfuscator)

A PPT algorithm Obf *that takes as input any program* P *and a security parameter* n*, and outputs obfuscated program* $\tilde{\mathsf{P}}$ *such that:*

1. *Functionality preserved: for all inputs* x, $\tilde{\mathsf{P}}(x) = \mathsf{P}(x)$

Virtual Black-Box Obfuscation (VBBO)...

- Security via "simulation": anything learnable "white-box" given $\tilde{\mathsf{P}}$ is learnable "black-box" given only oracle access to P



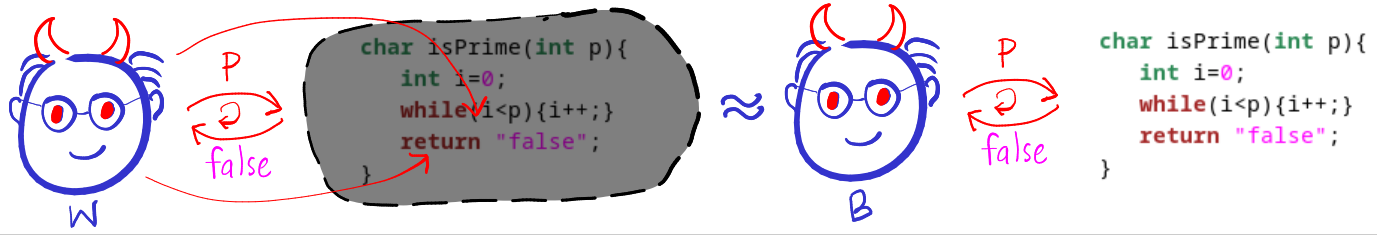
Defintion 1 (VBB obfuscator)

A PPT algorithm Obf *that takes as input any program* P *and a security parameter* n*, and outputs obfuscated program* $\tilde{\mathsf{P}}$ *such that:*

1. *Functionality preserved: for all inputs* x, $\tilde{\mathsf{P}}(x) = \mathsf{P}(x)$
2. *Small slowdown: run-time of* $\tilde{\mathsf{P}}$ *is poly. in* n *and run-time of* P

Virtual Black-Box Obfuscation (VBBO)...

- Security via "simulation": anything learnable "white-box" given P is learnable "black-box" given only oracle access to P



Defintion 1 (VBB obfuscator)

A PPT algorithm Obf *that takes as input any program* P *and a security parameter* n*, and outputs obfuscated program* $\tilde{P}$ *such that:*

1. *Functionality preserved: for all inputs* x, $\tilde{P}(x) = P(x)$
2. *Small slowdown: run-time of* $\tilde{P}$ *is poly. in* n *and run-time of* P
3. *VBBO security: for every PPT* W*, there exists PPT* B *that can simulate* W*'s output on input* $\tilde{P}$ *using only oracle access to* P*. That is, the following is negligible:*

$$\left| \Pr\left[W(\tilde{P}) = 1\right] - \Pr\left[B^{P}(1^n, 1^{|P|}) = 1\right] \right|$$

$$\tilde{P} \leftarrow \mathrm{Obf}(1^n, P)$$

Plan for Today's Lecture

1 Program Obfuscation

2 Building Primitives Using VBBO

3 Impossibility of VBBO for General Programs

OWF $\xrightarrow{\text{VBBO}}$ OWP

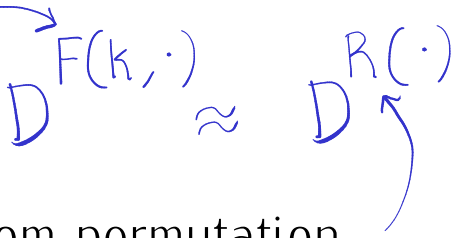
- Recall that OWF $\rightarrow$ PRG $\rightarrow$ PRF

OWF $\xrightarrow{\text{VBBO}}$ OWP

- Recall that OWF $\rightarrow$ PRG $\rightarrow$ PRF $\rightarrow$ PRP
  - PRP: pseudo-random *permutation*
  - Computationally indistinguishable from random permutation

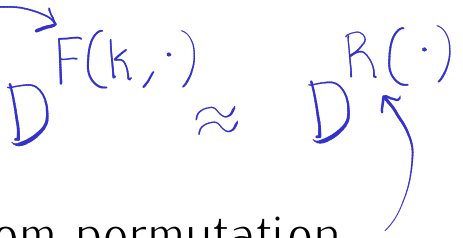
OWF $\xrightarrow{\text{VBBO}}$ OWP

- Recall that OWF $\rightarrow$ PRG $\rightarrow$ PRF $\rightarrow$ PRP
  - PRP: pseudo-random *permutation*
  - Computationally indistinguishable from random permutation



OWF $\xrightarrow{\text{VBBO}}$ OWP

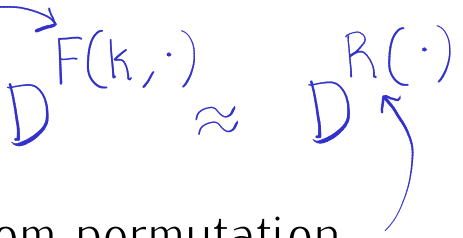
- Recall that OWF $\to$ PRG $\to$ PRF $\to$ PRP
  - PRP: pseudo-random *permutation*
  - Computationally indistinguishable from random permutation



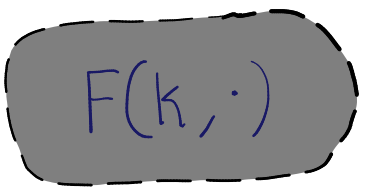
❓ How to construct (keyed) OWP Π from PRP F?

OWF $\xrightarrow{\text{VBBO}}$ OWP

- Recall that OWF $\to$ PRG $\to$ PRF $\to$ PRP
  - PRP: pseudo-random *permutation*
  - Computationally indistinguishable from random permutation

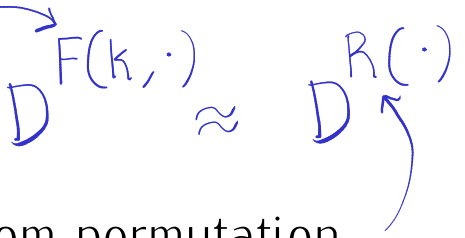


- How to construct (keyed) OWP Π from PRP F?
  - VBB obfuscate the PRP F!

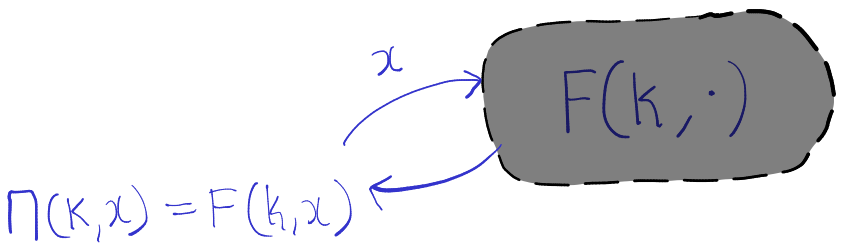


OWF $\xrightarrow{\text{VBBO}}$ OWP

- Recall that OWF $\to$ PRG $\to$ PRF $\to$ PRP
  - PRP: pseudo-random *permutation*
  - Computationally indistinguishable from random permutation



- ? How to construct (keyed) OWP Π from PRP F?
  - VBB obfuscate the PRP F!



Construction 1 (PRP $\mathsf{F} \to$ OWP Π)

- *Sample* $k \leftarrow \{0,1\}^n$ *and output* $K \leftarrow \mathsf{Obf}(\mathsf{F}(k,\cdot))$ *as key for* Π
- $\Pi(K,x) := K(x)$

$\mathrm{OWF} \xrightarrow{\mathrm{VBBO}} \mathrm{OWP}$

Claim 1

If Obf *is VBB obfuscator and* F *is PRP then* Π *is a OWP*

OWF $\xrightarrow{\text{VBBO}}$ OWP

Claim 1

If **Obf** *is VBB obfuscator and* **F** *is PRP then* Π *is a OWP*

Proof sketch. (**Obf** not VBBO or **F** not PRP $\Leftarrow \exists$**Inv** for OWP Π).

Let $W = \mathsf{Inv}$ be WB learner for $F(k,\cdot)$ that inverts $y \leftarrow \{0,1\}^n$ under Π

□

OWF $\xrightarrow{\text{VBBO}}$ OWP

Claim 1

If **Obf** *is VBB obfuscator and* **F** *is PRP then* Π *is a OWP*

Proof sketch. (**Obf** not VBBO or **F** not PRP $\Leftarrow \exists$**Inv** for OWP Π).

Let $W = \text{Inv}$ be WB learner for $F(k,\cdot)$ that inverts $y \leftarrow \{0,1\}^n$ under Π

Case 1: $\nexists$ BB learner B s.t

$$\left| \Pr\left[W(F(k,\cdot), \Pi(F(k,\cdot), y)) = x\right] - \Pr\left[B^{F(k,\cdot)}(1^n, y) = x\right] \right| = \text{negl}(n)$$

$F(k,\cdot) \leftarrow \text{Obf}(1^n, F(k,\cdot))$, $y \leftarrow \{0,1\}^n$

$y \leftarrow \{0,1\}^n$

□

OWF $\xrightarrow{\text{VBBO}}$ OWP

Claim 1

If **Obf** *is VBB obfuscator and* **F** *is PRP then* Π *is a OWP*

Proof sketch. (**Obf** not VBBO or **F** not PRP $\Leftarrow \exists$**Inv** for OWP Π).

Let $W = \mathsf{Inv}$ be WB learner for $F(k,\cdot)$ that inverts $y \leftarrow \{0,1\}^n$ under Π

Case 1: $\nexists$ BB learner B s.t

$$\left| \Pr\left[W(F(k,\cdot), \Pi(F(k,\cdot), y)) = x\right] - \Pr\left[B^{F(k,\cdot)}(1^n, y) = x\right] \right| = \mathrm{negl}(n)$$

$F(k,\cdot) \leftarrow \mathsf{Obf}(1^n, F(k,\cdot))$ $\quad$ $y \leftarrow \{0,1\}^n$

$y \leftarrow \{0,1\}^n$

$\Rightarrow$ Obf is not VBBO (w/ auxiliary inputs)

□

OWF $\xrightarrow{\text{VBBO}}$ OWP

Claim 1

If **Obf** *is VBB obfuscator and* **F** *is PRP then* Π *is a OWP*

Proof sketch. (**Obf** not VBBO or **F** not PRP $\Leftarrow \exists$**Inv** for OWP Π).

Let $W = \mathsf{Inv}$ be WB learner for $F(k,\cdot)$ that inverts $y \leftarrow \{0,1\}^n$ under Π

Case 1: $\exists$ BB learner B s.t

$$\left| \Pr\left[W(F(k,\cdot), \Pi(F(k,\cdot), y)) = x\right] - \Pr\left[B^{F(k,\cdot)}(1^n, y) = x\right] \right| = \mathrm{negl}(n)$$

$F(k,\cdot) \leftarrow \mathsf{Obf}(1^n, F(k,\cdot))$, $y \leftarrow \{0,1\}^n$; $y \leftarrow \{0,1\}^n$

OWF $\xrightarrow{\text{VBBO}}$ OWP

Claim 1

If **Obf** *is VBB obfuscator and* **F** *is PRP then* Π *is a OWP*

Proof sketch. (**Obf** not VBBO or **F** not PRP $\Leftarrow \exists$**Inv** for OWP Π).

Let $W = \mathsf{Inv}$ be WB learner for $F(k,\cdot)$ that inverts $y \leftarrow \{0,1\}^n$ under Π

Case 1: $\exists$ BB learner B s.t

$$\left| \Pr\left[W(F(k,\cdot), \Pi(F(k,\cdot), y)) = x\right] - \Pr\left[B^{F(k,\cdot)}(1^n, y) = x\right] \right| = \mathrm{negl}(n)$$

$F(k,\cdot) \leftarrow \mathsf{Obf}(1^n, F(k,\cdot))$ $\quad$ $y \leftarrow \{0,1\}^n$

$y \leftarrow \{0,1\}^n$

since F PRP $\not\Rightarrow$ B inverts random permutation $R(\cdot)$ w/ non-negl. probability ↯

OWF $\xrightarrow{\text{VBBO}}$ OWP

Claim 1

If **Obf** *is VBB obfuscator and* **F** *is PRP then* Π *is a OWP*

Proof sketch. (**Obf** not VBBO or **F** not PRP $\Leftarrow \exists$**Inv** for OWP Π).

Let $W = \mathsf{Inv}$ be WB learner for $F(k,\cdot)$ that inverts $y \leftarrow \{0,1\}^n$ under Π

Case 1: $\exists$ BB learner B s.t

$$\left| \Pr\left[W(F(k,\cdot), \Pi(F(k,\cdot), y)) = x\right] - \Pr\left[B^{F(k,\cdot)}(1^n, y) = x\right] \right| = \mathrm{negl}(n)$$

$F(k,\cdot) \leftarrow \mathsf{Obf}(1^n, F(k,\cdot))$ $\qquad$ $y \leftarrow \{0,1\}^n$

$y \leftarrow \{0,1\}^n$

since F PRP $\Rightarrow$ B inverts random permutation $R(\cdot)$ w/ non-negl. probability ↯

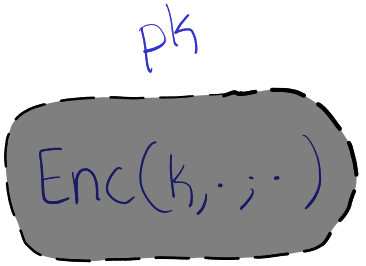
? Does Claim 1 contradict OWF $\not\to$ OWP from Lecture 21?

- No, Π is a *non-black-box* construction (it uses F's program)

How to construct PKE from SKE?

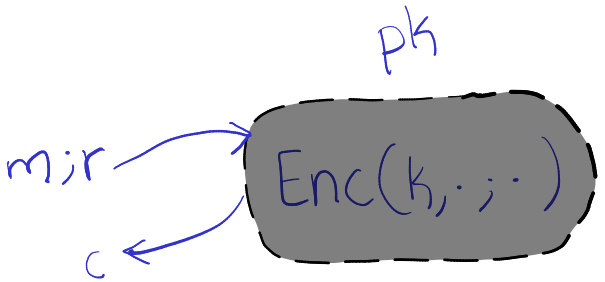
❓ How to construct PKE from SKE?

- What if the PKE's public key is an obfuscation of SKE's encrypt algorithm **Enc** with secret key k *hardcoded*?



How to construct PKE from SKE?

- What if the PKE's public key is an obfuscation of SKE's encrypt algorithm **Enc** with secret key k *hardcoded*?



Construction 2 ($\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \rightarrow \Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$)

- $\mathsf{Gen}'(1^n)$:
  - *Sample* $k \leftarrow \mathsf{Gen}(1^n)$
  - *Output* $\mathsf{Obf}(\mathsf{Enc}(k, \cdot; \cdot))$ *as public key* pk *and* k *as secret key*
- $\mathsf{Enc}'(pk, m; r)$: *output* $c := pk(m; r)$
- $\mathsf{Dec}'(k, c)$: *output* $m := \mathsf{Dec}(k, c)$

SKE $\xrightarrow{\text{VBBO}}$ PKE...

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.

[Diffie, Hellman 76]

SKE $\xrightarrow{\text{VBBO}}$ PKE...

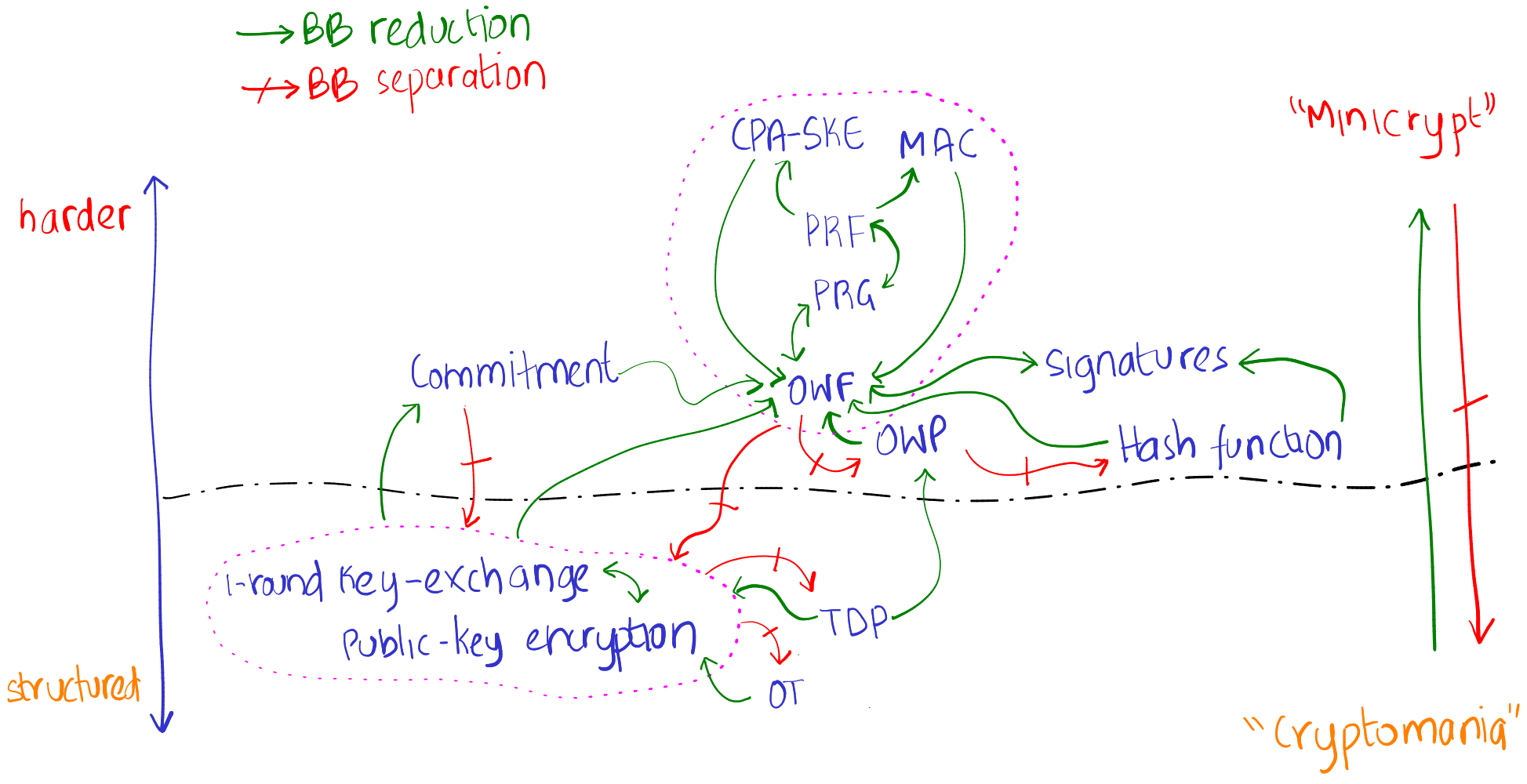
Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-way because it must be feasible to do the compilation, but infeasible to reverse 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.

[Diffie, Hellman 76]

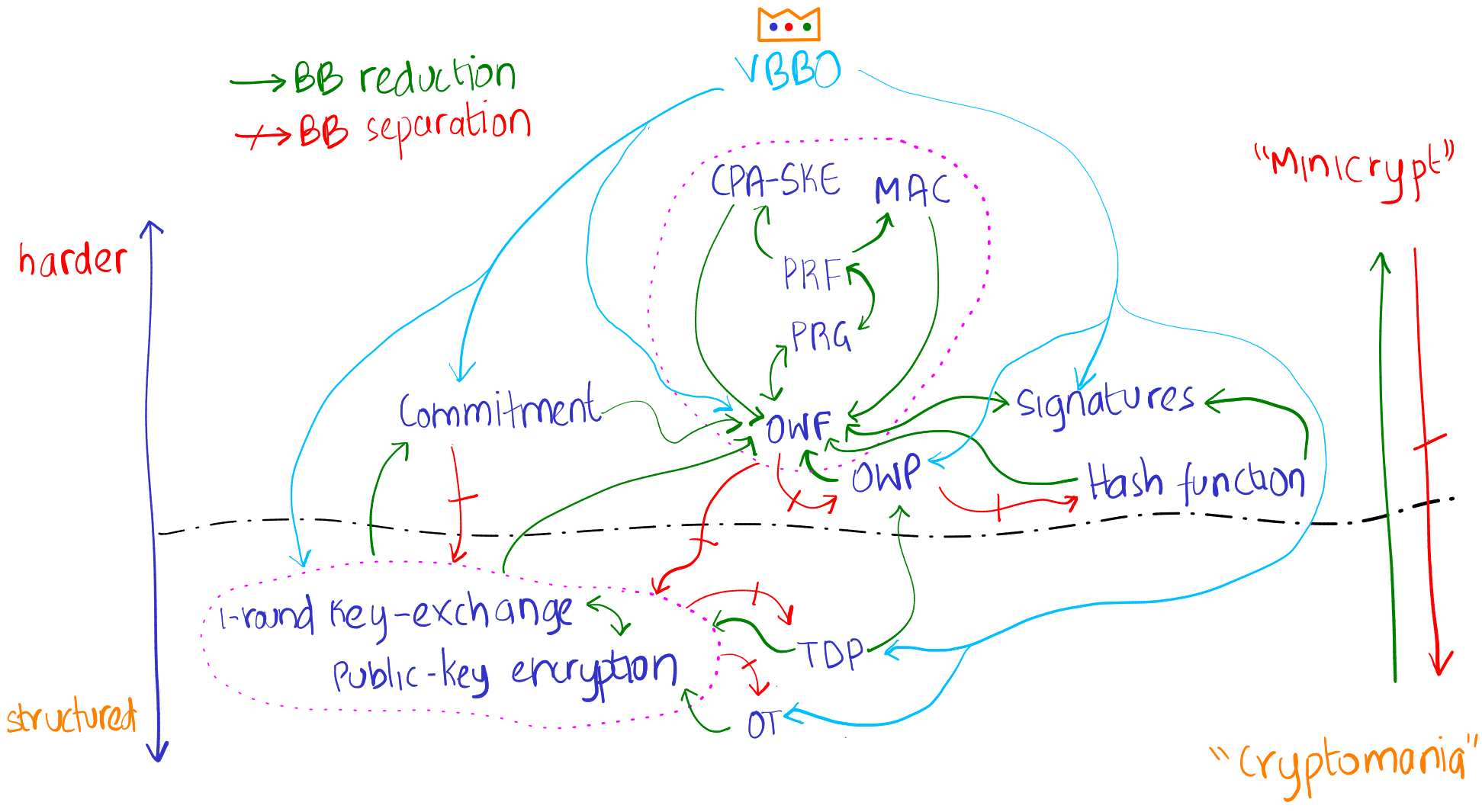
Exercise 1

- *Come up with an attack against Construction 2 (Hint: substitute a concrete SKE we have seen in this course such that the construction fails.)*
- *Define VBBO for randomised programs; fix Construction 2*

VBBO is (Almost) "Crypto Complete"!...



VBBO is (Almost) "Crypto Complete"!...



VBBO is (Almost) "Crypto Complete"!...

Exercise 2

Given VBB obfuscator, construct:

1. *one-way function*
2. *fully-homomorphic encryption (FHE) from any SKE*
3. *non-interactive commitment*
4. *trapdoor permutation*

Plan for this Session

1 Program Obfuscation

2 Building Primitives Using VBBO

3 Impossibility of VBBO for General Programs

Let's Rule out VBBO for General Programs...

- Recall security requirement of VBBO: for every program P and
  1. for every efficient white-box learner W
  2. there exists an efficient black-box learner B such that
  3. B simulates W's output on input P using only oracle access to P

Let's Rule out VBBO for General Programs...

- Recall security requirement of VBBO: for every program P and
  1. for every efficient white-box learner W
  2. there exists an efficient black-box learner B such that
  3. B simulates W's output on input P using only oracle access to P

Theorem 1 (VBBO for general programs is impossible)

For every Obf *there exists a program* P^* *and*

1. *a particular efficient white-box learner* W^* *such that*
2. *for every black-box learner* B,
3. B *fails to simulate* W^**'s output on input* P^* *using only oracle access to* P^*

Let's Rule out VBBO for General Programs...

- Recall security requirement of VBBO: for every program P and
  1. for every efficient white-box learner W
  2. there exists an efficient black-box learner B such that
  3. B simulates W's output on input P using only oracle access to P

Theorem 1 (VBBO for general programs is impossible)

For every Obf *there exists a program* P^* *and*

1. *a particular efficient white-box learner* W^* *such that*
2. *for every black-box learner* B,
3. B *fails to simulate* W^**'s output on input* P^* *using only oracle access to* P^*

- Easy case: when W^* can output arbitrarily-long strings
  - What is W^*'s strategy?

Let's Rule out VBBO for General Programs...

- Recall security requirement of VBBO: for every program P and
  1. for every efficient white-box learner W
  2. there exists an efficient black-box learner B such that
  3. B simulates W's output on input P using only oracle access to P

Theorem 1 (VBBO for general programs is impossible)

For every Obf *there exists a program* P^* *and*

1. *a particular efficient white-box learner* W^* *such that*
2. *for every black-box learner* B,
3. B *fails to simulate* W^**'s output on input* P^* *using only oracle access to* P^*

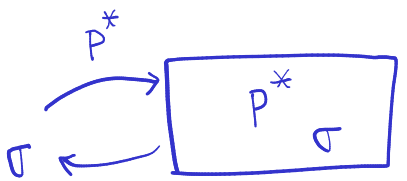
- Easy case: when W^* can output arbitrarily-long strings
  - What is W^*'s strategy? Simply output P^*
    - (Every) B only has black-box access to $\mathsf{P}^* \Rightarrow \mathsf{B}$ cannot output description of P^*

Let's Rule out VBBO for General Programs...

- Harder case: when W^* outputs short strings (e.g., a bit)

Let's Rule out VBBO for General Programs...

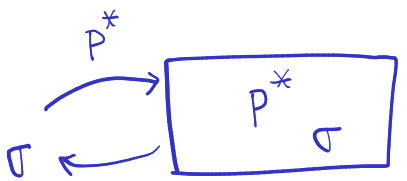
- Harder case: when W^* outputs short strings (e.g., a bit)



- Idea: come up with a program P^* such that
  1. P^* spits out some short secret string σ when run *"on itself"*
     - P^* needs to be a Turing machine

Let's Rule out VBBO for General Programs...

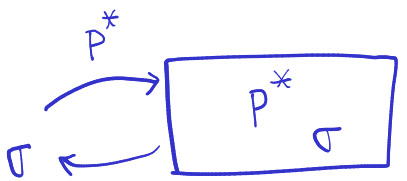
- Harder case: when W^* outputs short strings (e.g., a bit)



- Idea: come up with a program P^* such that
  1. P^* spits out some short secret string σ when run *"on itself"*
     - P^* needs to be a Turing machine
  2. W^* has P^* and can therefore access σ

Let's Rule out VBBO for General Programs...

- Harder case: when W^* outputs short strings (e.g., a bit)



- Idea: come up with a program P^* such that
  1. P^* spits out some short secret string σ when run *"on itself"*
     - P^* needs to be a Turing machine
  2. W^* has P^* and can therefore access σ
  3. (Every) B only has black-box access to $P^* \Rightarrow B$ cannot access σ
- Challenge: avoid circularity when defining P^*

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases}$$

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

delta/point fn. $\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases}$

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

delta/point fn.

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases} \qquad S_{\alpha,\beta,\sigma}(x) := \begin{cases} \sigma & \text{if } x(\alpha) = \beta \\ 0 & \text{if } x(\alpha) \neq \beta \end{cases}$$

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

S interpreted as TM

delta/point fn.

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases}$$

$$\mathsf{S}_{\alpha,\beta,\sigma}(x) := \begin{cases} \sigma & \text{if } x(\alpha) = \beta \\ 0 & \text{if } x(\alpha) \neq \beta \end{cases}$$

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

S interpreted as TM

delta/point fn.

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases} \qquad \mathsf{S}_{\alpha,\beta,\sigma}(x) := \begin{cases} \sigma & \text{if } x(\alpha) = \beta \\ 0 & \text{if } x(\alpha) \neq \beta \end{cases}$$

  - Given Δ and S, W^* runs $\mathsf{S}(\Delta)$ to obtain ...

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

S interpreted as TM

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases} \qquad \mathsf{S}_{\alpha,\beta,\sigma}(x) := \begin{cases} \sigma & \text{if } x(\alpha) = \beta \\ 0 & \text{if } x(\alpha) \neq \beta \end{cases}$$

delta/point fn.

  - Given Δ and S, W^* runs $\mathsf{S}(\Delta)$ to obtain ... σ
  - Without Δ and S, no B can access σ

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

delta/point fn.

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases} \qquad S_{\alpha,\beta,\sigma}(x) := \begin{cases} \sigma & \text{if } x(\alpha) = \beta \\ 0 & \text{if } x(\alpha) \neq \beta \end{cases}$$

interpreted as TM

  - Given Δ and S, W^* runs $\mathsf{S}(\Delta)$ to obtain ... σ
  - Without Δ and S, no B can access σ
- To get a single program P^*, simply "MUX" Δ and S:

$$P^*_{\alpha,\beta,\sigma}(b,x) := \begin{cases} \Delta_{\alpha,\beta}(x) & \text{if } b = 0 \\ S_{\alpha,\beta,\sigma}(x) & \text{if } b = 1 \end{cases}$$

Let's Rule out VBBO for General Programs...

- Consider the following Δ and S for $\alpha, \beta, \sigma \in \{0,1\}^n$:

"simulates" Δ

interpreted as TM

delta/point fn.

$$\Delta_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases} \qquad \mathsf{S}_{\alpha,\beta,\sigma}(x) := \begin{cases} \sigma & \text{if } x(\alpha) = \beta \\ 0 & \text{if } x(\alpha) \neq \beta \end{cases}$$

  - Given Δ and S, W^* runs $\mathsf{S}(\Delta)$ to obtain ... σ
  - Without Δ and S, no B can access σ

- To get a single program P^*, simply "MUX" Δ and S:

$$\mathsf{P}^*_{\alpha,\beta,\sigma}(b,x) := \begin{cases} \Delta_{\alpha,\beta}(x) & \text{if } b = 0 \\ \mathsf{S}_{\alpha,\beta,\sigma}(x) & \text{if } b = 1 \end{cases}$$

  - Given P^*, W^* defines $\Delta := \mathsf{P}^*(0,\cdot)$ and $\mathsf{S} := \mathsf{P}^*(1,\cdot)$
  - W^* runs Δ on S to obtain σ
  - Without P^*, B cannot access σ

Exercise 3

Come up with your own P^*!

Way Around: Relax to Indistinguishability Obfuscation

- Security: obfuscations of two *functionally-equivalent* programs are computationally indistinguishable

Defintion 2 (Indistinguishability obfuscator (IO))

A PPT algorithm Obf *that takes as input any program* P *and security parameter* n*, and outputs obfuscated program* P *such that:*

1. *Functionality preserved: for all inputs* x, $\mathsf{P}(x) = \mathsf{P}(x)$
2. *Slowdown is polynomial: run-time of* P *is polynomial in* n *and run-time of* P

Way Around: Relax to Indistinguishability Obfuscation

- Security: obfuscations of two *functionally-equivalent* programs are computationally indistinguishable

Defintion 2 (Indistinguishability obfuscator (IO))

A PPT algorithm Obf *that takes as input any program* P *and security parameter* n*, and outputs obfuscated program* P *such that:*

1. *Functionality preserved: for all inputs* x, $\mathsf{P}(x) = \mathsf{P}(x)$
2. *Slowdown is polynomial: run-time of* P *is polynomial in* n *and run-time of* P
3. *IO security: for every equivalent* P_1 *and* P_2 *and PPT distinguisher* D*, the following is negligible:*

$$\left| \Pr[\mathsf{D}(P_1) = 1] - \Pr[\mathsf{D}(P_2) = 1] \right|$$

$$P_1 \leftarrow \mathsf{Obf}(1^n, P_1) \qquad P_2 \leftarrow \mathsf{Obf}(1^n, P_2)$$

Way Around: Relax to Indistinguishability Obfuscation

- Security: obfuscations of two *functionally-equivalent* programs are computationally indistinguishable

Defintion 2 (Indistinguishability obfuscator (IO))

A PPT algorithm Obf *that takes as input any program* P *and security parameter* n*, and outputs obfuscated program* P *such that:*

1. *Functionality preserved: for all inputs* x, $\mathsf{P}(x) = \mathsf{P}(x)$
2. *Slowdown is polynomial: run-time of* P *is polynomial in* n *and run-time of* P
3. *IO security: for every equivalent* P_1 *and* P_2 *and PPT distinguisher* D*, the following is negligible:*

$$\left| \Pr[D(P_1) = 1] - \Pr[D(P_2) = 1] \right|$$

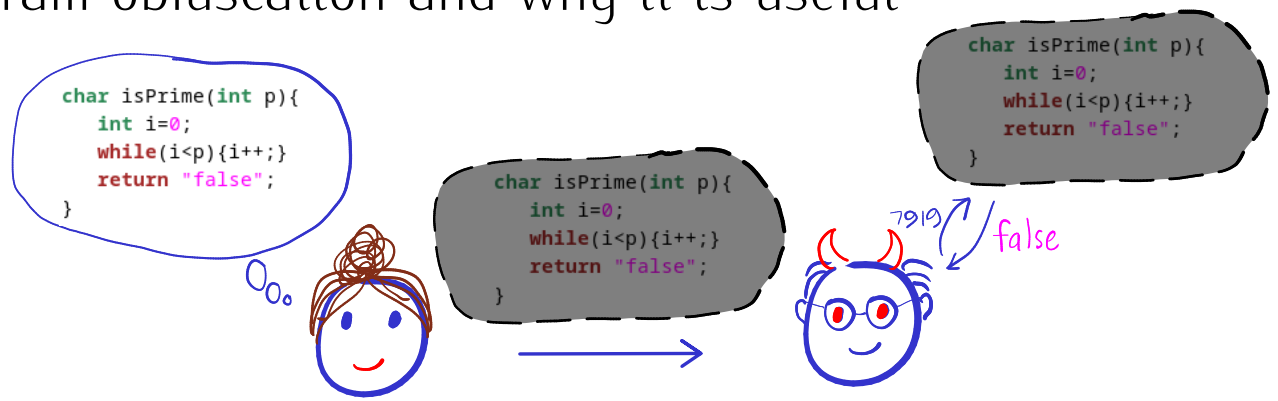
$P_1 \leftarrow Obf(1^n, P_1)$ $\quad$ $P_2 \leftarrow Obf(1^n, P_2)$

Exercise 4

1) Show that VBBO $\rightarrow$ *IO 2) Figure out why Theorem 1 fails for IO*

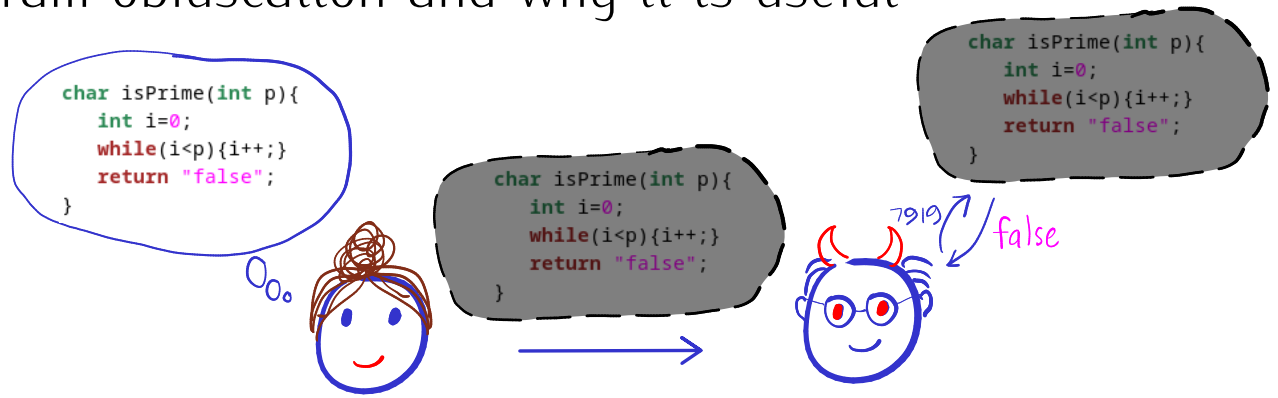
To Recap Today's Lecture

- Program obfuscation and why it is useful



To Recap Today's Lecture

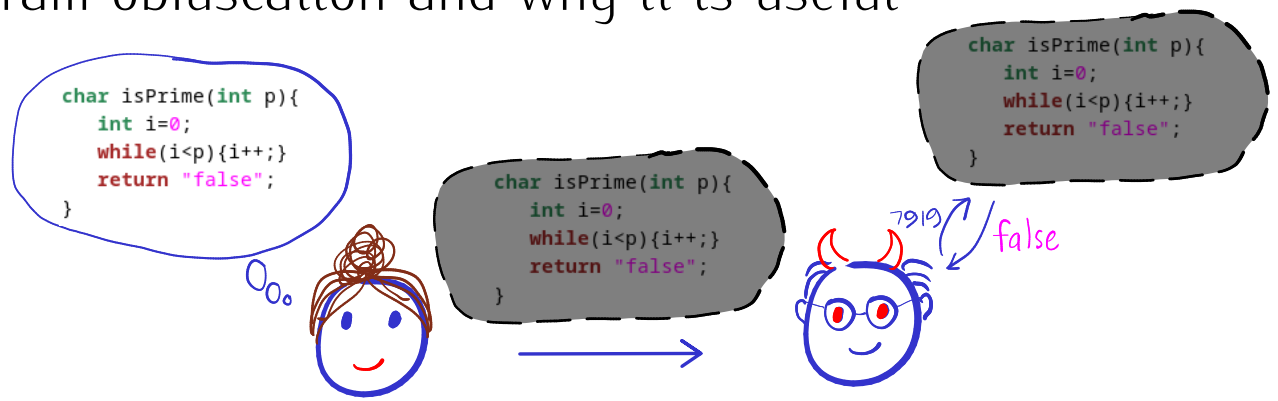
- Program obfuscation and why it is useful



- How to model security?
  - Today's lecture: virtual black-box (VBB) obfuscation
  - Next lecture: *indistinguishability* obfuscation (IO)

To Recap Today's Lecture

- Program obfuscation and why it is useful

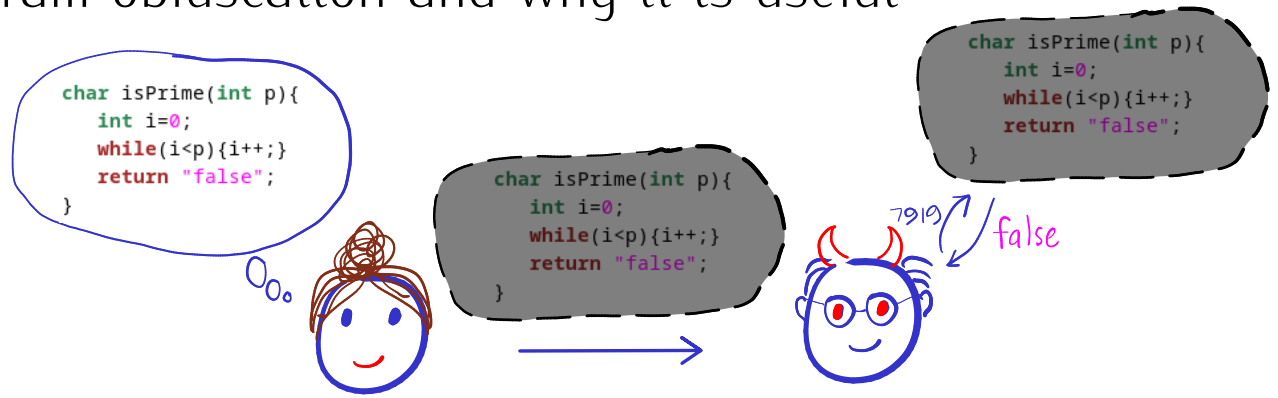


- How to model security?
  - Today's lecture: virtual black-box (VBB) obfuscation
  - Next lecture: *indistinguishability* obfuscation (IO)

- Bypassed black-box separations exploiting primitive's program:
  - OWF $\xrightarrow{\text{VBBO}}$ OWP
  - SKE $\xrightarrow{\text{VBBO}}$ PKE

To Recap Today's Lecture

- Program obfuscation and why it is useful



- How to model security?
  - Today's lecture: virtual black-box (VBB) obfuscation
  - Next lecture: *indistinguishability* obfuscation (IO)

- Bypassed black-box separations exploiting primitive's program:
  - OWF $\xrightarrow{\text{VBBO}}$ OWP
  - SKE $\xrightarrow{\text{VBBO}}$ PKE

- Impossibility of VBBO for general programs
  - Key idea: programs that spit out secret when run "on itself"

Next Lecture

- Friday (31/Oct): no lecture (Diwali eve)

Next Lecture

- Friday (31/Oct): no lecture (Diwali eve)
- Tuesday (04/Nov): More on indistinguishability obfuscator (IO)
  - Theorem 1 does not apply to IO
  - How to use IO?
    - SKE $\xrightarrow{\text{IO}}$ PKE

References

1. Most of this lecture is based on
   - Lectures 1 and 2 of Mark Zhandry's COS 597C course (Autumn 2016); and
   - Lecture 25 of Vinod Vaikuntanathan's MIT6875
2. VBBO was studied rigorously in [BGI+01], which is where Theorem 1 was also proved. It is the same paper that also introduces IO.
3. The problem of constructing cryptographic primitives using IO was studied much later in [SW14]
4. Mark Zhandry's COS 597C course (Autumn 2016) is an excellent source to learn further about program obfuscation


Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang.

On the (im)possibility of obfuscating programs.

In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.


Amit Sahai and Brent Waters.

How to use indistinguishability obfuscation: deniable encryption, and more.

In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.