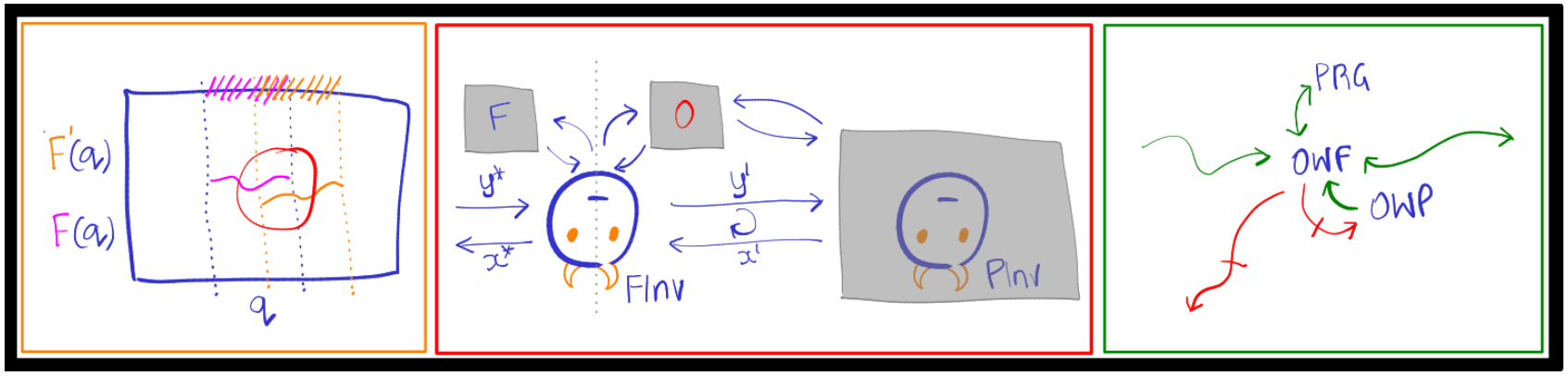


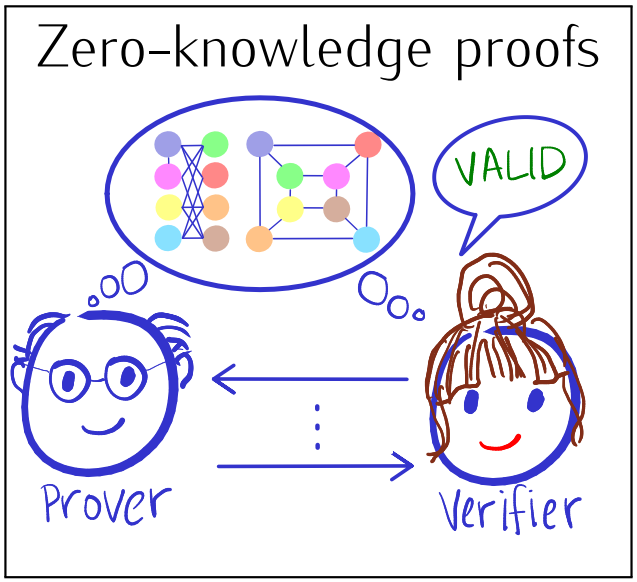
CS783: Theoretical Foundations of Cryptography

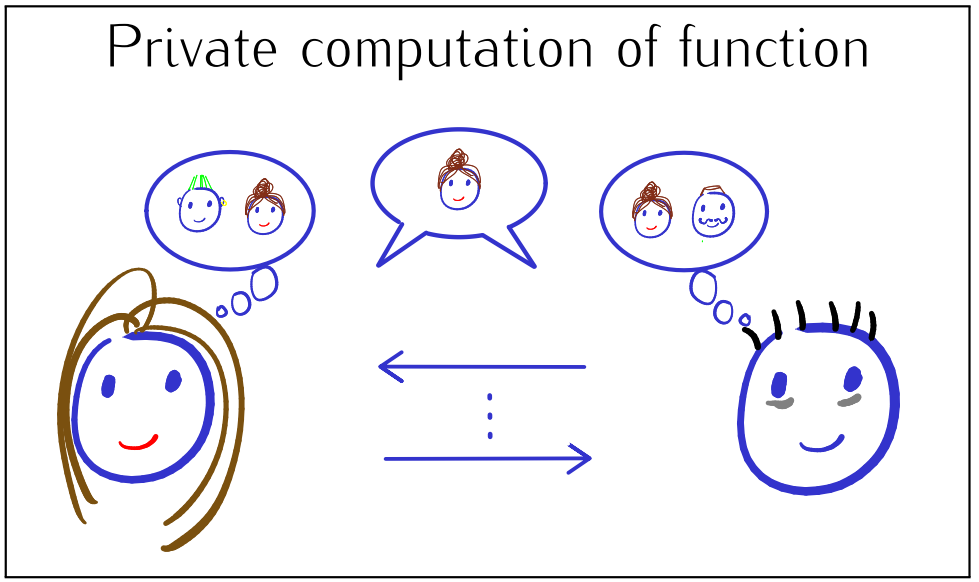
Lecture 21 (22/Oct/24)

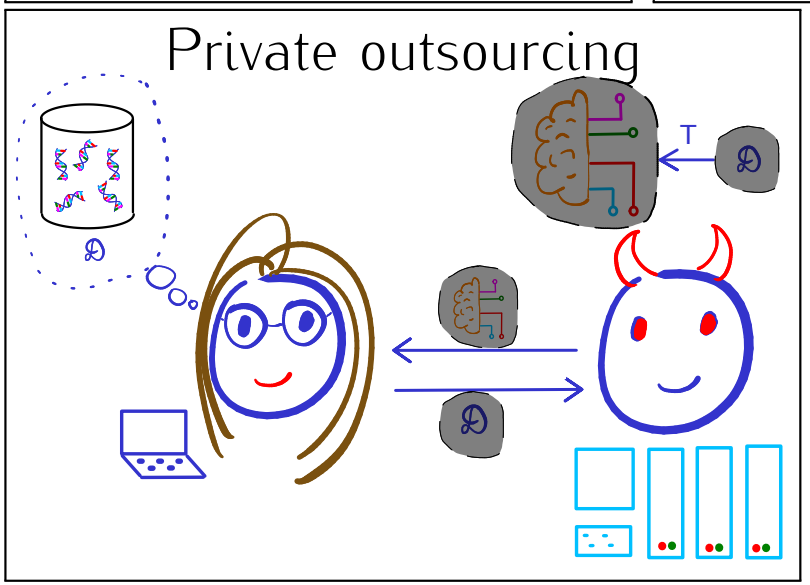
Instructor: Chethan Kamath

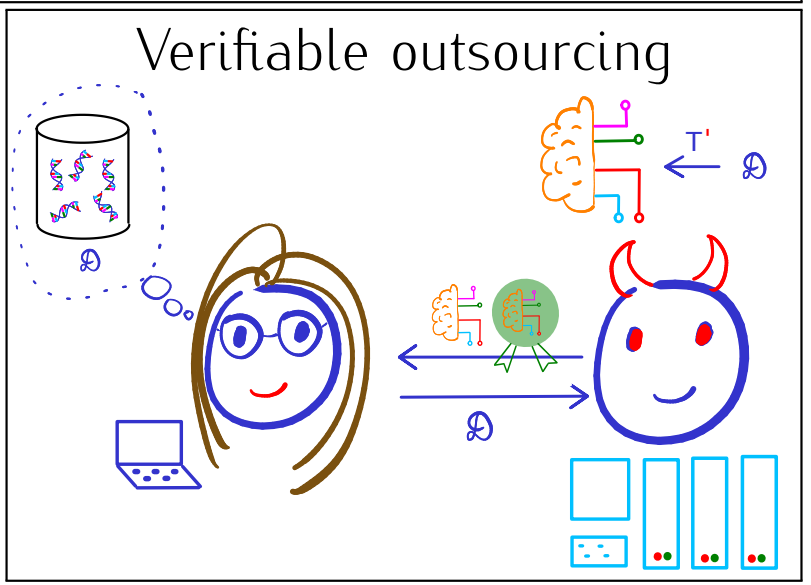
Recall from Last Module

- We saw several avatars of secure computation









Plan for Next Three Lectures...

MODULE 1
(Shared keys)

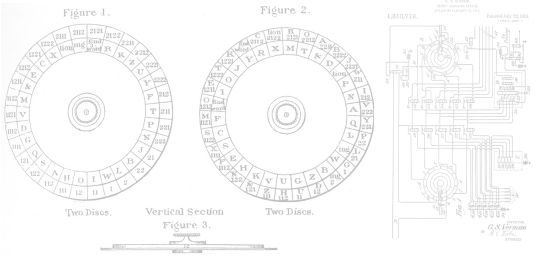
MODULE 2
(Public keys)

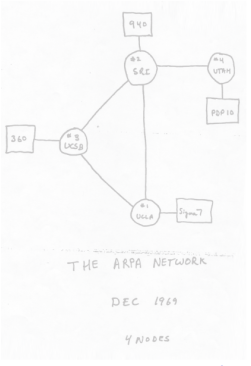
MODULE 3
(Secure comp.)

MODULE 4
(Adv. tasks)

For a large part of history

↷ Advent of internet ↷ Ubiquity of computing





*

**

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "Not prime.\n";
}
```

Plan for Next Three Lectures...

- Focus today: limitations of *black-box reductions*

Plan for Next Three Lectures...

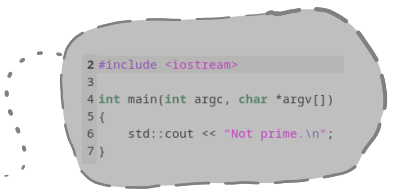
- Focus today: limitations of *black-box reductions*
  - Formalise black-box reduction of one primitive (e.g., PRF) to another (e.g., PRG) PRG → PRF
- Black-box *separations*
  - Certain primitives (e.g., PKE) *cannot* be "black-box reduced" to others (e.g., OWF) OWF ↛ PKE

Plan for Next Three Lectures...

- Focus today: limitations of *black-box reductions*
  - Formalise black-box reduction of one primitive (e.g., PRF) to another (e.g., PRG) PRG $\rightarrow$ PRF
- Black-box *separations*
  - Certain primitives (e.g., PKE) *cannot* be "black-box reduced" to others (e.g., OWF) OWF $\not\rightarrow$ PKE
  - Formalise black-box separation
  - Separate OWF from OWP OWF $\not\rightarrow$ OWP

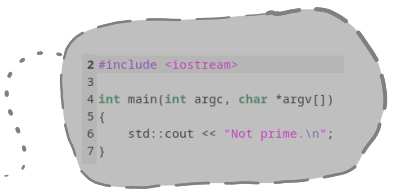
Plan for Next Three Lectures...

- Focus today: limitations of *black-box reductions*
  - Formalise black-box reduction of one primitive (e.g., PRF) to another (e.g., PRG) PRG → PRF
- Black-box *separations*
  - Certain primitives (e.g., PKE) *cannot* be "black-box reduced" to others (e.g., OWF) OWF ↛ PKE
  - Formalise black-box separation
  - Separate OWF from OWP OWF ↛ OWP
- Later:
  - Code obfuscation



Plan for Next Three Lectures...

- Focus today: limitations of *black-box reductions*
  - Formalise black-box reduction of one primitive (e.g., PRF) to another (e.g., PRG) PRG → PRF
- Black-box *separations*
  - Certain primitives (e.g., PKE) *cannot* be "black-box reduced" to others (e.g., OWF) OWF ↛ PKE
  - Formalise black-box separation
  - Separate OWF from OWP OWF ↛ OWP
- Later:
  - Code obfuscation



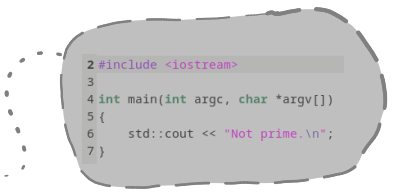
```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "Not prime.\n";
}
```

  - How to formalise security?
    - Virtual black-box (VBB) and indistinguishability obfuscator (IO)

Plan for Next Three Lectures...

- Focus today: limitations of *black-box reductions*
  - Formalise black-box reduction of one primitive (e.g., PRF) to another (e.g., PRG) PRG $\rightarrow$ PRF
- Black-box *separations*
  - Certain primitives (e.g., PKE) *cannot* be "black-box reduced" to others (e.g., OWF) OWF $\not\rightarrow$ PKE
  - Formalise black-box separation
  - Separate OWF from OWP OWF $\not\rightarrow$ OWP
- Later:
  - Code obfuscation



  - How to formalise security?
    - Virtual black-box (VBB) and indistinguishability obfuscator (IO)
  - Code obfuscation is powerful
    - Helps bypass certain black-box separations (e.g., OWF $\not\rightarrow$ PKE)
    - IO $\rightarrow$ most cryptographic primitives!

Plan for Today's Lecture

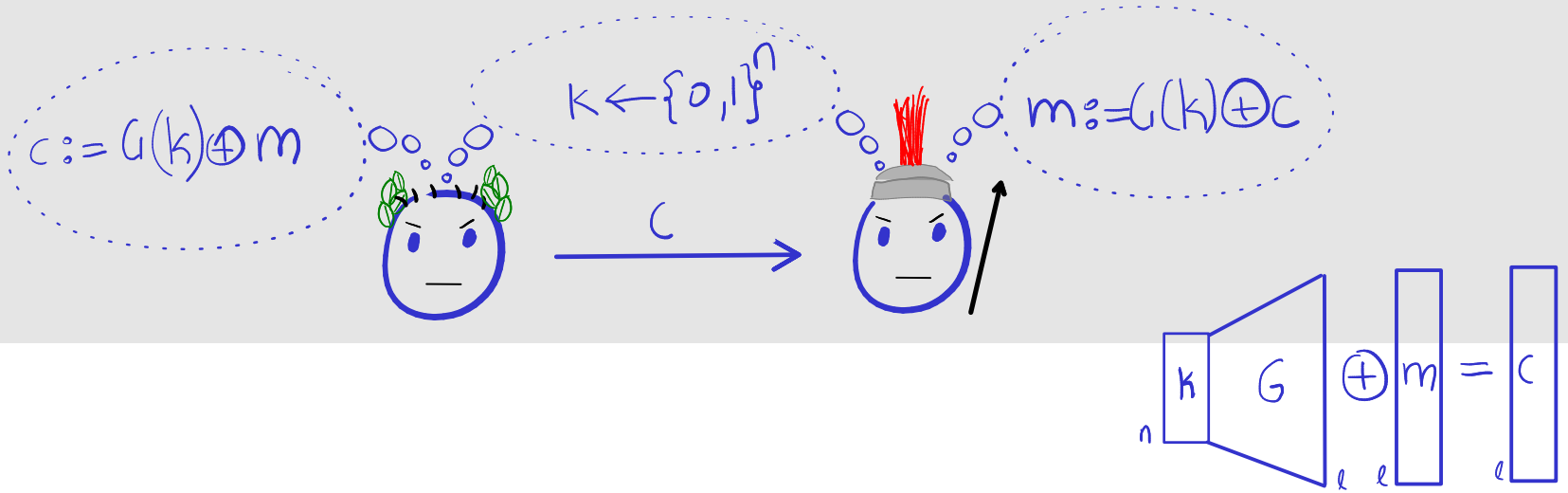
1. Black-Box Reduction $\rightarrow$
2. Black-Box Separation $\not\rightarrow$
3. Black-Box Separating OWF from OWP $\mathrm{OWF} \not\rightarrow \mathrm{OWP}$

Plan for Today's Lecture

1. Black-Box Reduction $\rightarrow$
2. Black-Box Separation $\not\rightarrow$
3. Black-Box Separating OWF from OWP $\mathsf{OWF} \not\rightarrow \mathsf{OWP}$

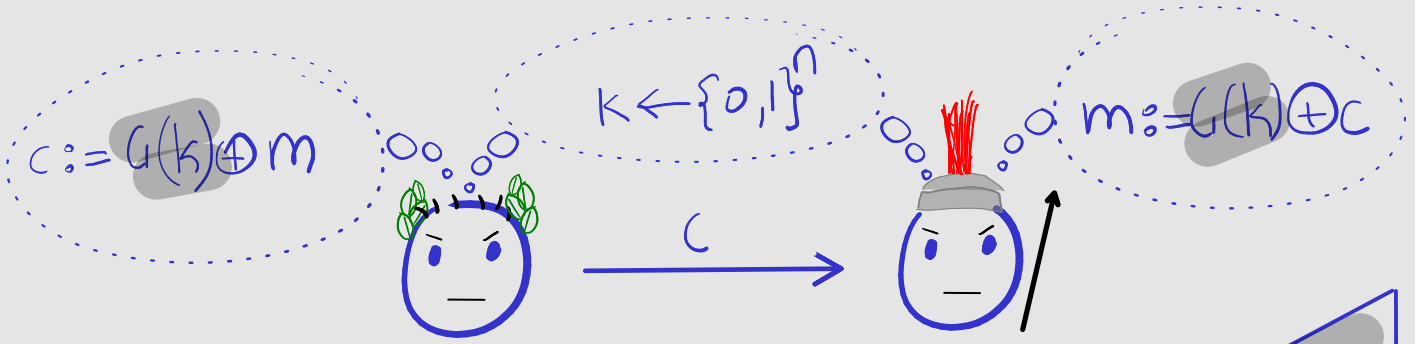
Recall Our First Cryptographic Reduction...

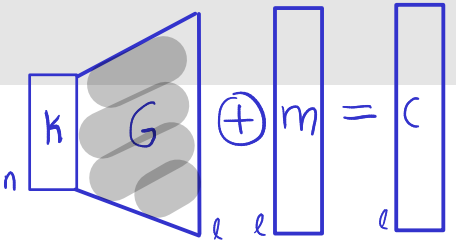
Construction 1 (Computational OTP Π from PRG G)



Recall Our First Cryptographic Reduction...

Construction 1 (Computational OTP Π from PRG G)

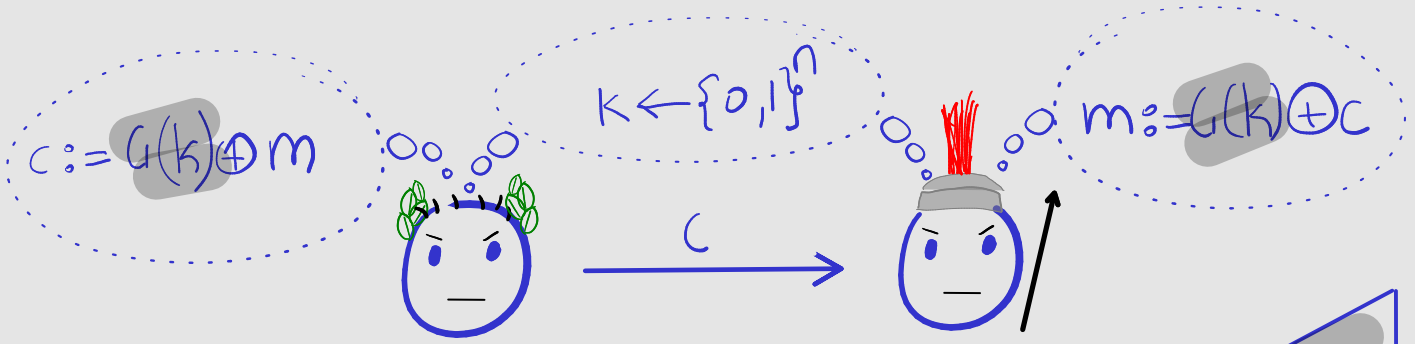


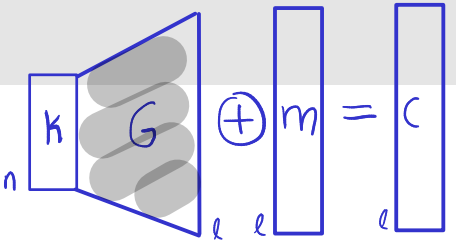


- Construction Π uses PRG G as a "black-box"
  - Only needs query access to G
  - Does not depend on exact implementation of G

Recall Our First Cryptographic Reduction...

Construction 1 (Computational OTP Π from PRG G)





- Construction Π uses PRG G as a "black-box"
  - Only needs query access to G
  - Does not depend on exact implementation of G

Pseudocode 1 (of Π^G) ↪ denotes oracle/black-box access to G

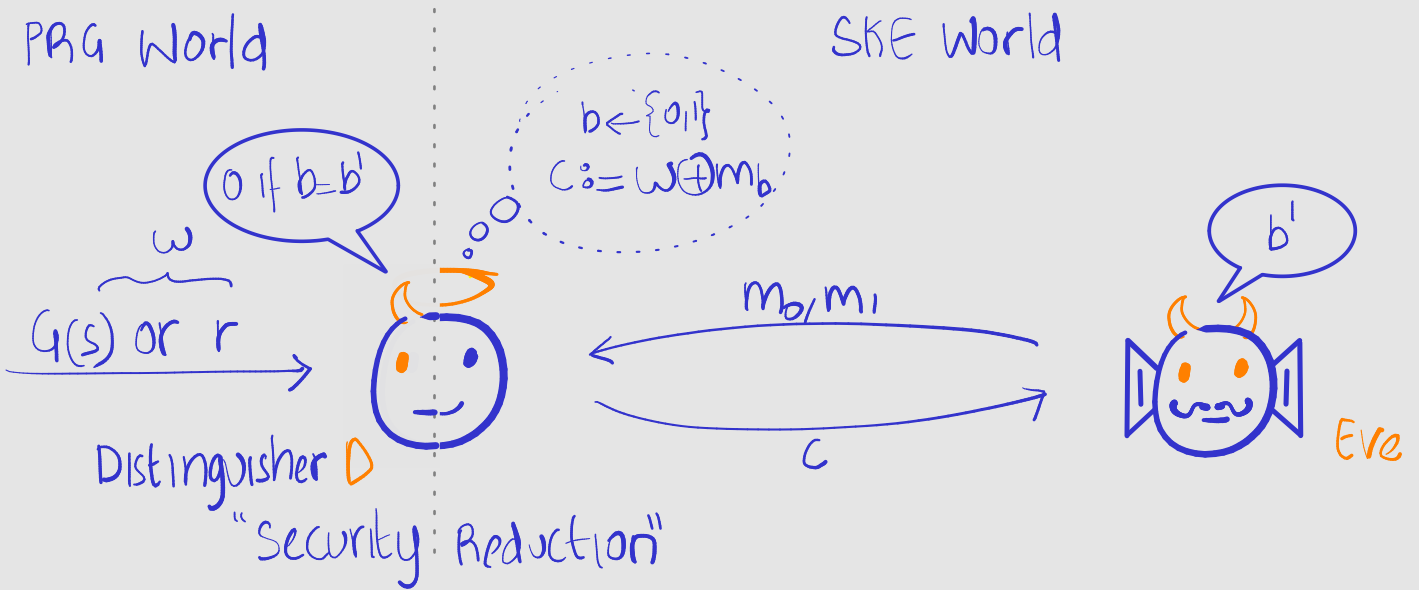
- $\mathsf{Gen}(1^n)$: *output* $k \leftarrow \{0,1\}^n$
- $\mathsf{Enc}^G(k, m)$: *query* G *on* k *to obtain* y *and output* $c := y \oplus m$
- $\mathsf{Dec}^G(k, c)$: *query* G *on* k *to obtain* y *and output* $m := y \oplus c$

Recall Our First Cryptographic Reduction...

Theorem 1

Assuming G *is a PRG, Construction 1 is computationally secret.*

Proof by *security reduction*: ∃Eve breaking Π ⇒ ∃D for G.



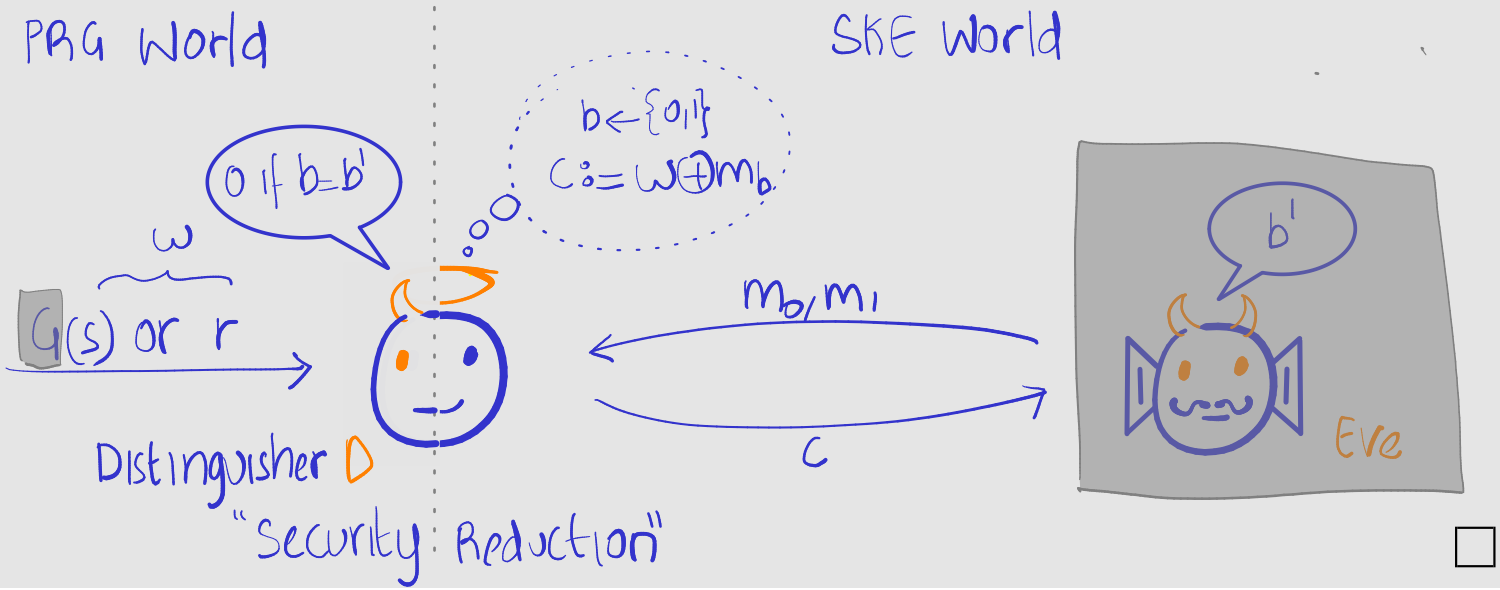
□

Recall Our First Cryptographic Reduction...

Theorem 1

Assuming G *is a PRG, Construction 1 is computationally secret.*

Proof by *security reduction*: ∃Eve breaking Π ⇒ ∃D for G.



- Security reduction D (+analysis) uses G and Eve as black box
  - Only needs query access to G and Eve: $D^{G,Eve}$
  - Does not depend on exact implementation of G and Eve

Our Reductions So Far Have All Been "Black Box"...

1. PRG $\rightarrow$ PRF (GGM construction)
2. PRF $\rightarrow$ CPA-SKE
3. OWP $\rightarrow$ hardcore predicate (Goldreich-Levin construction)
4. PRF $\rightarrow$ MAC
5. TDP $\rightarrow$ PKE
6. Commitment $\rightarrow$ Computational ZKP for **NP** (Blum's protocol)
7. OT $\rightarrow$ 2PC (GMW protocol)

...

Let's Formally Define (Fully) Black-Box Reduction

- *Both* construction and security reduction are black-box

Let's Formally Define (Fully) Black-Box Reduction

- *Both* construction and security reduction are black-box

Defintion 1 ((Fully) black-box (BB) reduction of OWP to OWF)

A pair of efficient oracle-algorithms $(\Pi^{\cdot}, \mathsf{Finv}^{\cdot,\cdot})$ *such that*

denotes oracle arguments

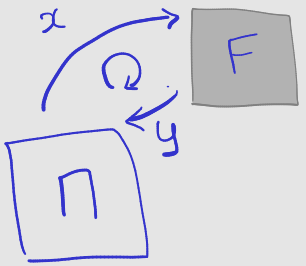
Let's Formally Define (Fully) Black-Box Reduction

- *Both* construction and security reduction are black-box

Defintion 1 ((Fully) black-box (BB) reduction of OWP to OWF)

A pair of efficient oracle-algorithms $(\Pi^{\cdot}, \mathbf{Finv}^{\cdot,\cdot})$ *such that* (denotes oracle arguments)

1. *Correctness: for every function* F*, construction* Π^F *is permutation*



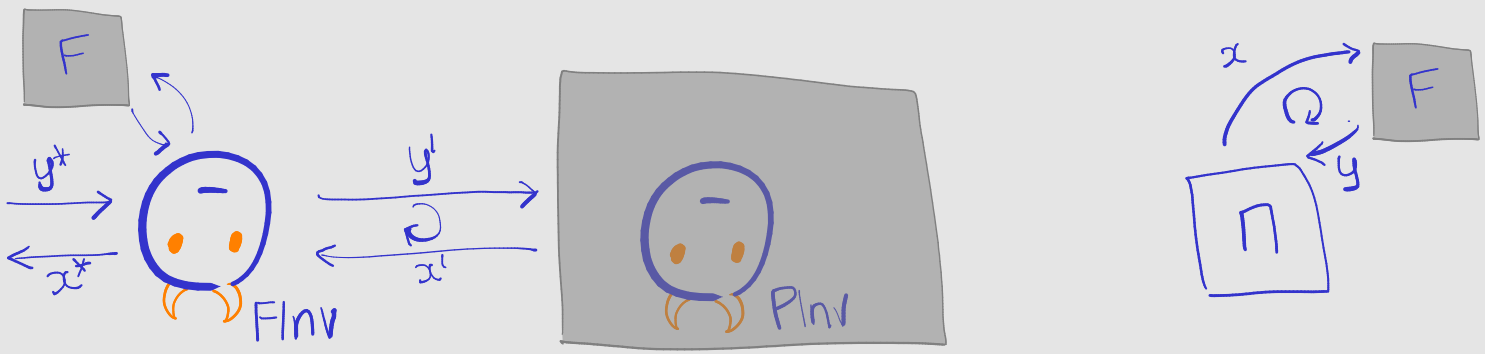
Let's Formally Define (Fully) Black-Box Reduction

- *Both* construction and security reduction are black-box

Defintion 1 ((Fully) black-box (BB) reduction of OWP to OWF)

A pair of efficient oracle-algorithms $(\Pi^{\cdot}, \mathbf{Finv}^{\cdot,\cdot})$ *such that* (denotes oracle arguments)

1. *Correctness: for every function* F*, construction* Π^F *is permutation*



2. *Security: for every one-way* F *and for every OWP-inverter* $\mathbf{Pinv}^{\cdot}$ *that inverts* Π^F*, the security reduction* $\mathbf{Finv}^{\mathbf{Pinv},F}$ *inverts* F

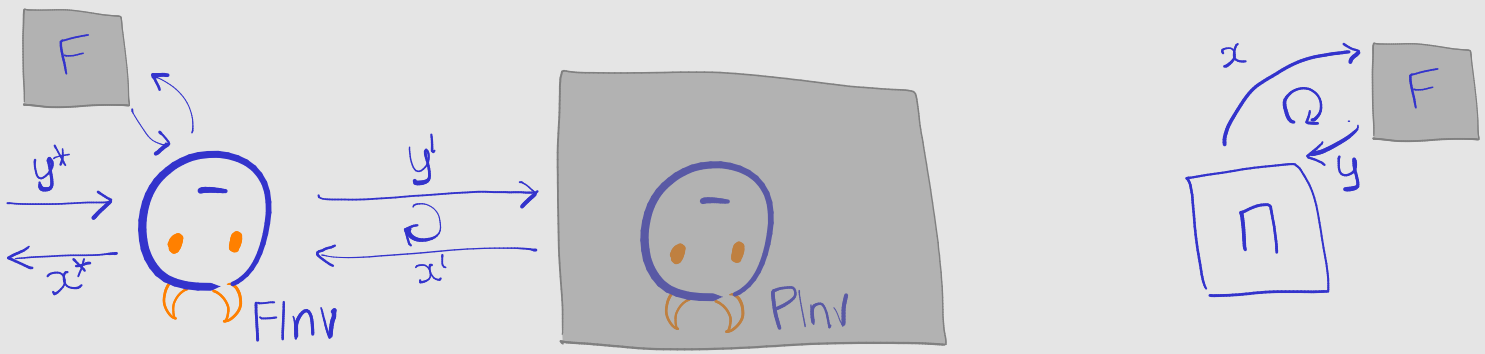
Let's Formally Define (Fully) Black-Box Reduction

- *Both* construction and security reduction are black-box

Defintion 1 ((Fully) black-box (BB) reduction of OWP to OWF)

A pair of efficient oracle-algorithms $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot})$ *such that* ($\cdot$ denotes oracle arguments)

1. *Correctness: for every function* F*, construction* Π^{F} *is permutation*



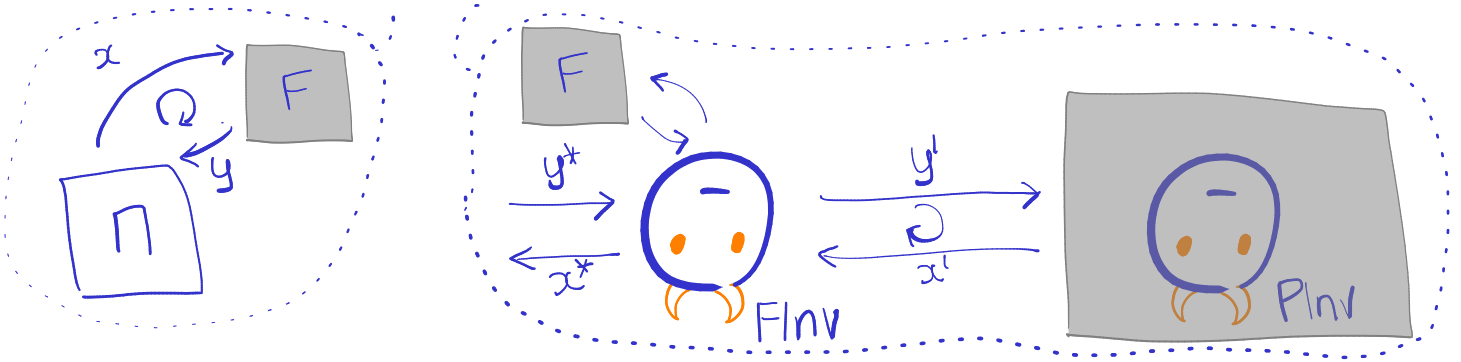
2. *Security: for every one-way* F *and for every OWP-inverter* $\mathbf{PInv}^{\cdot}$ *that inverts* Π^{F}*, the security reduction* $\mathbf{FInv}^{\mathbf{PInv},\mathsf{F}}$ *inverts* F

Exercise 1

Formulate the general definition for any two primitives $\mathcal{P}$ *and* $\mathcal{Q}$

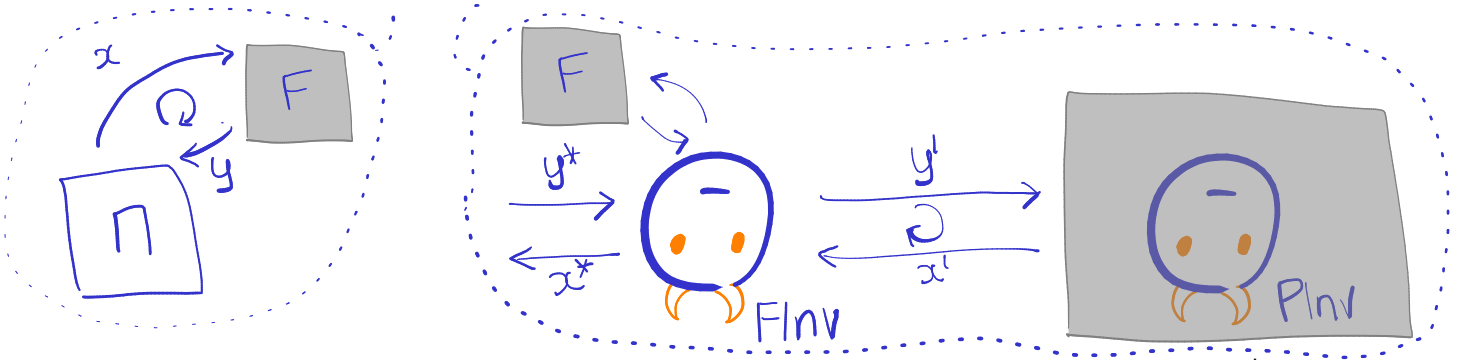
BB Reduction "Relativises"...

- BB reduction $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ of OWP to OWF:



BB Reduction "Relativises"...

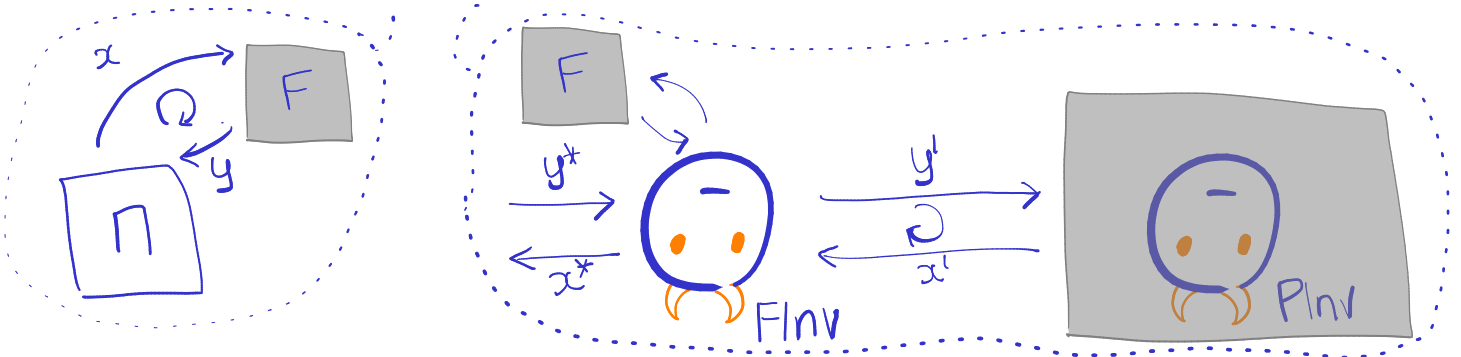
- BB reduction $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ of OWP to OWF:



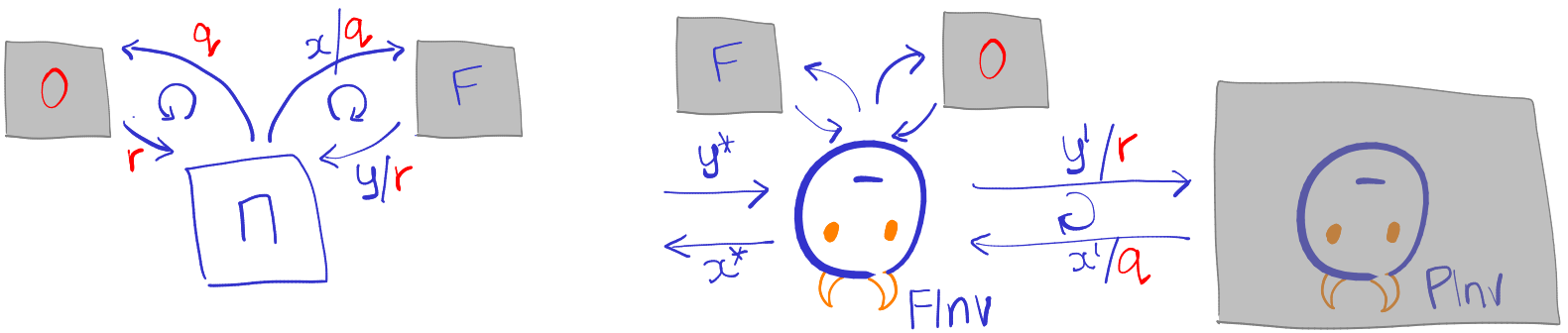
- $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ works *relative* to any oracle $\mathrm{O} : \{0,1\}^* \to \{0,1\}^*$
  - World where all parties have access to O

BB Reduction "Relativises"...

- BB reduction $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ of OWP to OWF:

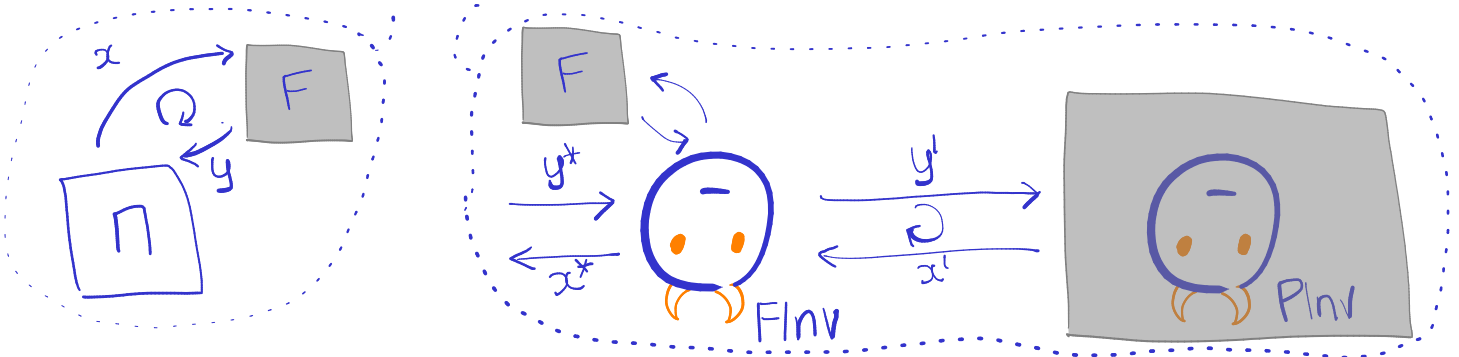


- $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ works *relative* to any oracle $\mathrm{O} : \{0,1\}^* \to \{0,1\}^*$
  - World where all parties have access to O
- BB reduction $(\Pi^{\cdot,\cdot}, \mathbf{FInv}^{\cdot,\cdot,\cdot})$ of OWP to OWF, relative to O:

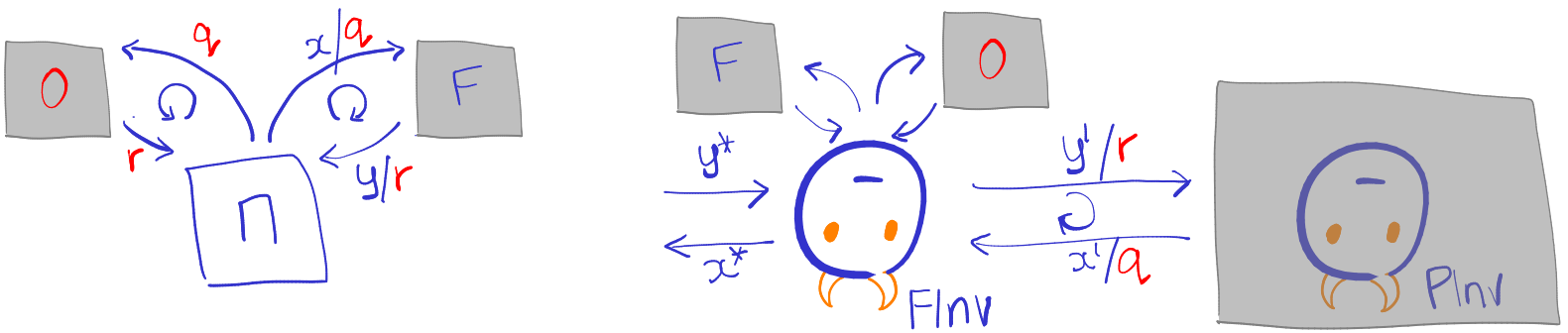


BB Reduction "Relativises"...

- BB reduction $(\Pi^{\cdot}, \mathsf{FInv}^{\cdot,\cdot})$ of OWP to OWF:



- $(\Pi^{\cdot}, \mathsf{FInv}^{\cdot,\cdot})$ works *relative* to any oracle $\mathrm{O} : \{0,1\}^* \to \{0,1\}^*$
  - World where all parties have access to O
- BB reduction $(\Pi^{\cdot,\cdot}, \mathsf{FInv}^{\cdot,\cdot,\cdot})$ of OWP to OWF, relative to O:



Claim 1 (Contrapositive)

$(\Pi^{\cdot,\cdot}, \mathsf{FInv}^{\cdot,\cdot,\cdot})$ *does not exist relative to some oracle* $\mathrm{O} \Rightarrow (\Pi^{\cdot}, \mathsf{FInv}^{\cdot,\cdot})$ *does not exist*

BB Reduction "Relativises"...

- PRG $\mathsf{G}^{\mathrm{O}} \rightarrow$ computational OTP $\Pi^{\mathsf{G}^{\cdot},\mathrm{O}}$

BB Reduction "Relativises"...

- PRG $\mathsf{G}^{\mathsf{O}} \rightarrow$ computational OTP $\Pi^{\mathsf{G},\mathsf{O}}$

Pseudocode 2 (Computational OTP $\Pi^{\mathsf{G},\mathsf{O}}$ from PRG G^{O})

- $\mathsf{Gen}(1^n)$: *output* $k \leftarrow \{0,1\}^n$ Π passes G's O-queries to O
- $\mathsf{Enc}^{\mathsf{G},\mathsf{O}}(k, m)$: *query* G^{O} *on* k *to obtain* y*; output* $c := y \oplus m$
- $\mathsf{Dec}^{\mathsf{G},\mathsf{O}}(k, c)$: *query* G^{O} *on* k *to obtain* y*; output* $m := y \oplus c$

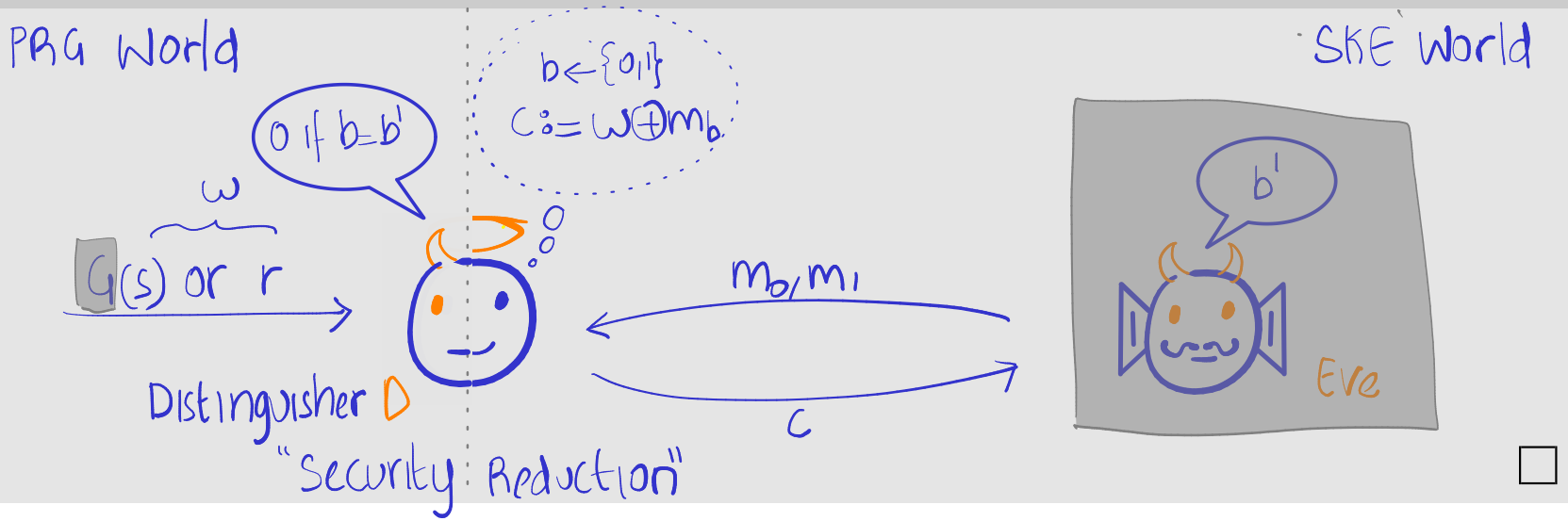
BB Reduction "Relativises"...

- PRG $\mathsf{G}^{\mathsf{O}} \rightarrow$ computational OTP $\Pi^{\mathsf{G}^{\cdot},\mathsf{O}}$

Pseudocode 2 (Computational OTP $\Pi^{\mathsf{G}^{\cdot},\mathsf{O}}$ from PRG G^{O})

- $\mathsf{Gen}(1^n)$: *output* $k \leftarrow \{0,1\}^n$ — Π passes G's O-queries to O
- $\mathsf{Enc}^{\mathsf{G},\mathsf{O}}(k,m)$: *query* G^{O} *on* k *to obtain* y; *output* $c := y \oplus m$
- $\mathsf{Dec}^{\mathsf{G},\mathsf{O}}(k,c)$: *query* G^{O} *on* k *to obtain* y; *output* $m := y \oplus c$

Proof by *security reduction*: $\exists \mathsf{Eve}^{\Pi^{\cdot},\mathsf{O}}$ breaking $\Pi^{\mathsf{G}^{\cdot},\mathsf{O}} \Rightarrow \exists \mathsf{D}^{\mathsf{Eve}^{\cdot},\mathsf{G}^{\cdot},\mathsf{O}}$.



□

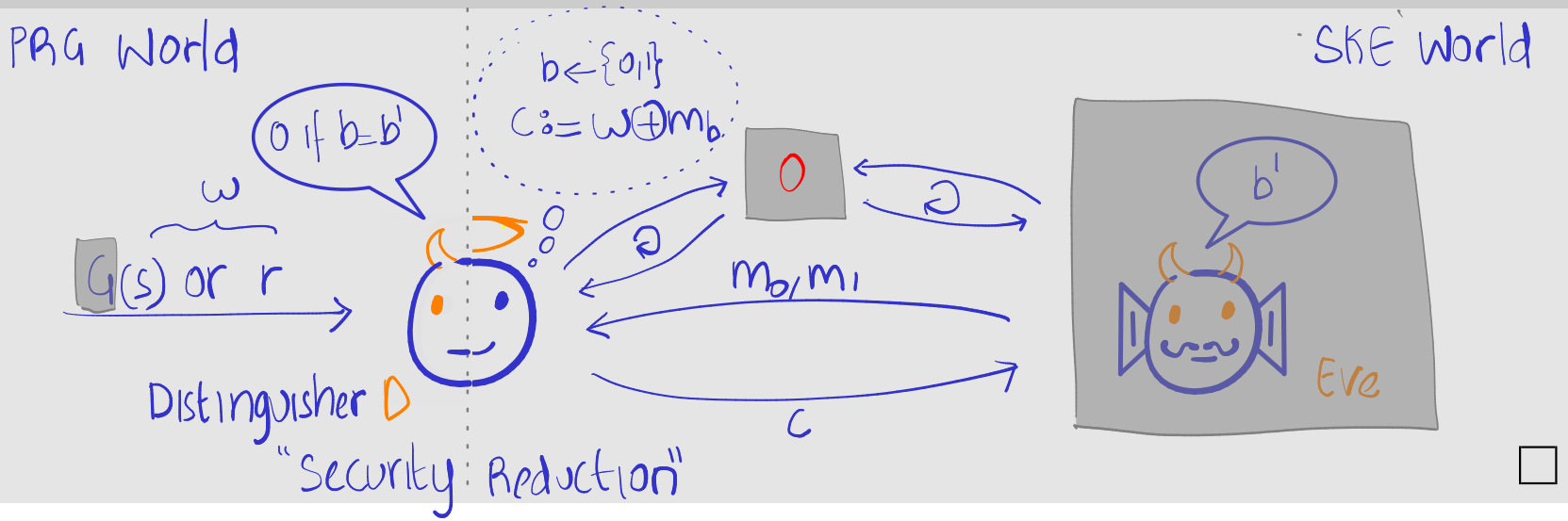
BB Reduction "Relativises"...

- PRG $\mathsf{G}^{\mathsf{O}} \rightarrow$ computational OTP $\Pi^{\mathsf{G},\mathsf{O}}$

Pseudocode 2 (Computational OTP $\Pi^{\mathsf{G},\mathsf{O}}$ from PRG G^{O})

- $\mathsf{Gen}(1^n)$: *output* $k \leftarrow \{0,1\}^n$ — Π passes G's O-queries to O
- $\mathsf{Enc}^{\mathsf{G},\mathsf{O}}(k,m)$: *query* G^{O} *on* k *to obtain* y; *output* $c := y \oplus m$
- $\mathsf{Dec}^{\mathsf{G},\mathsf{O}}(k,c)$: *query* G^{O} *on* k *to obtain* y; *output* $m := y \oplus c$

Proof by *security reduction*: $\exists \mathsf{Eve}^{\Pi,\mathsf{O}}$ breaking $\Pi^{\mathsf{G},\mathsf{O}} \Rightarrow \exists \mathsf{D}^{\mathsf{Eve},\mathsf{G},\mathsf{O}}$.

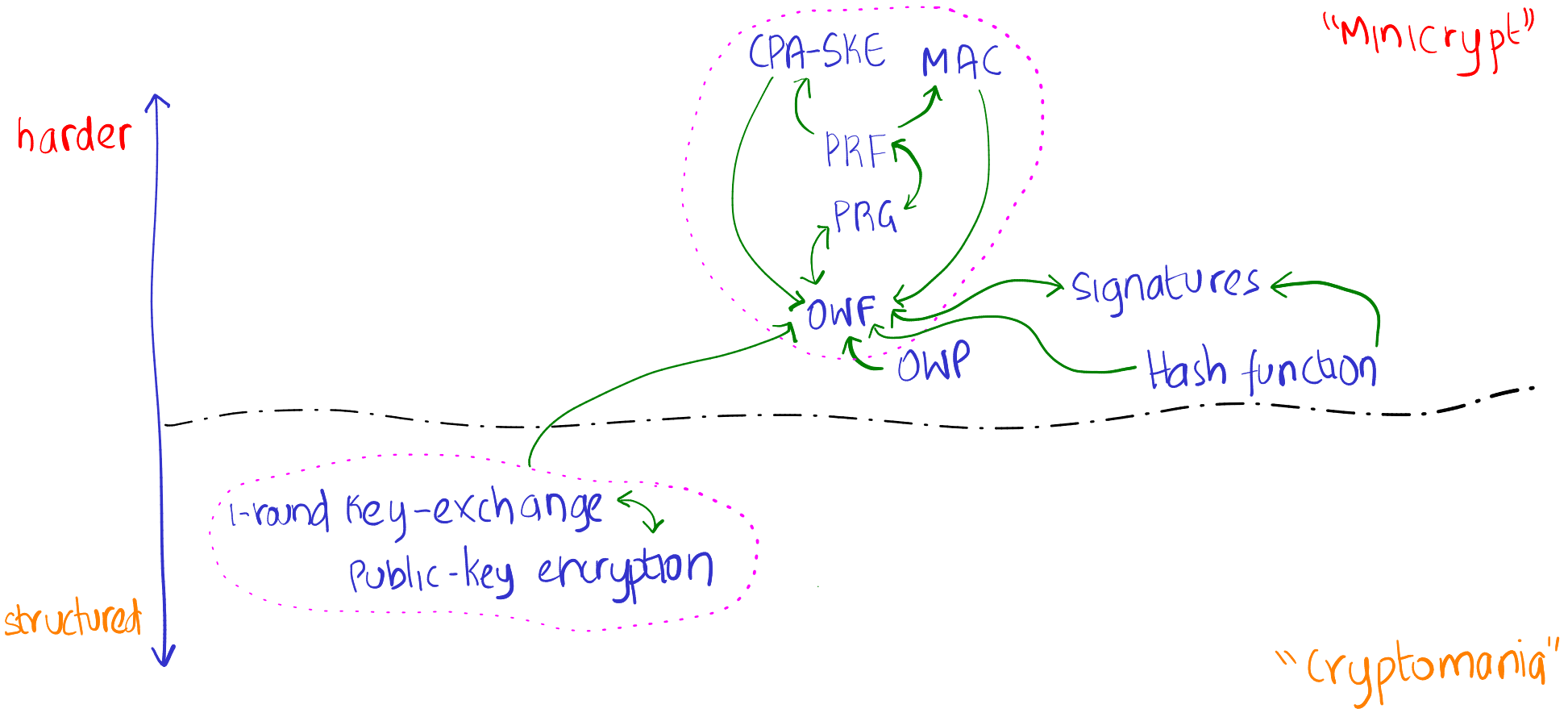


□

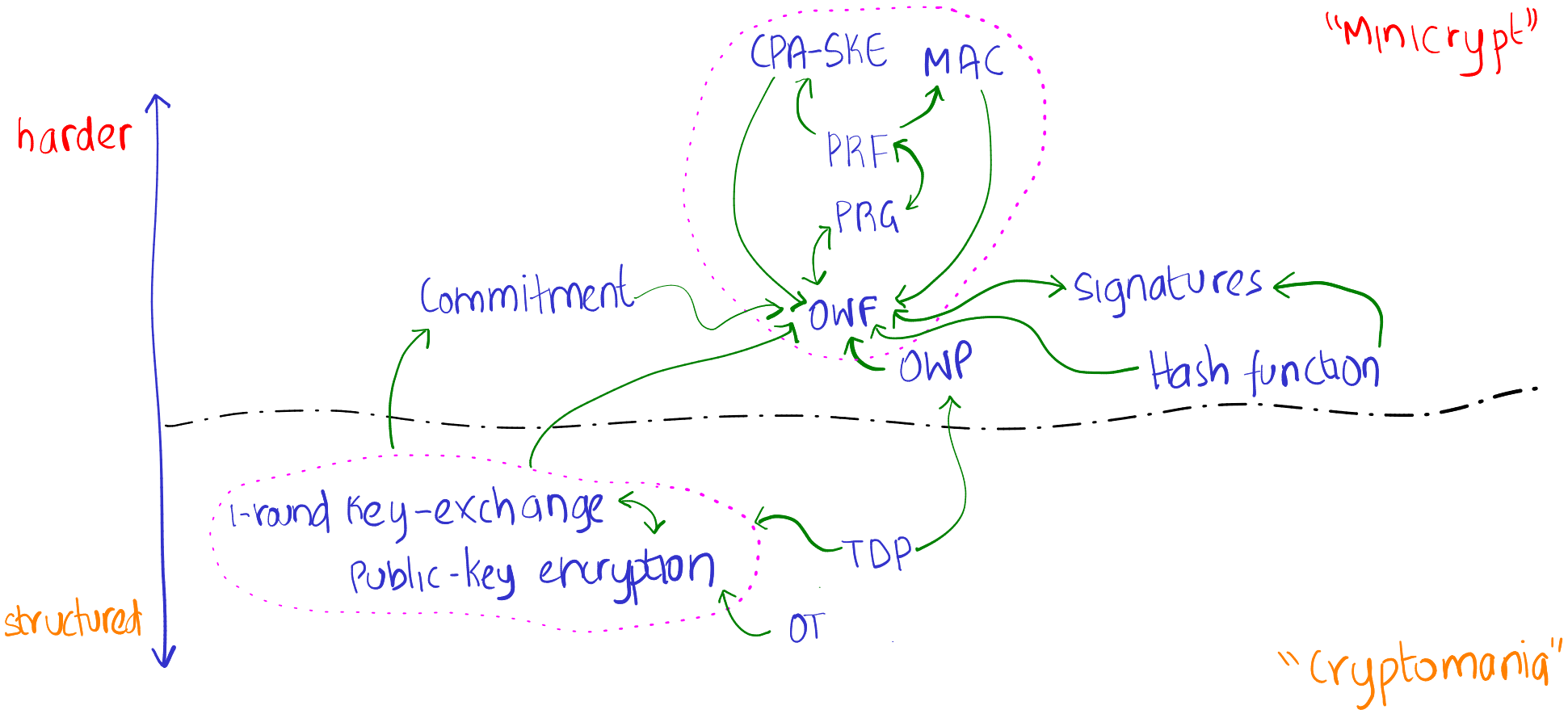
Plan for this Session

1. Black-Box Reduction $\rightarrow$
2. Black-Box Separation $\not\rightarrow$
3. Black-Box Separating OWF from OWP $\mathrm{OWF} \not\rightarrow \mathrm{OWP}$

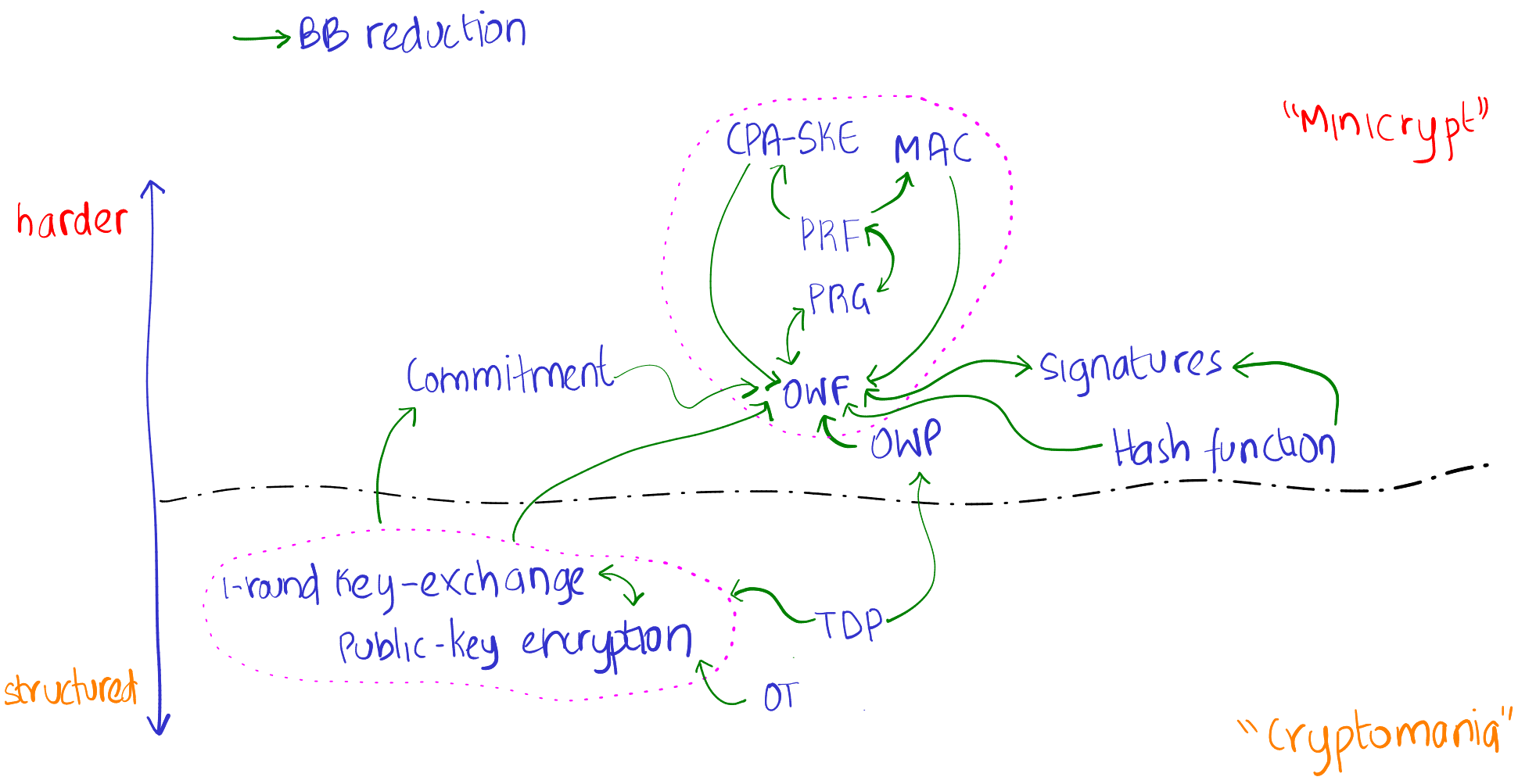
Recall Our Landscape



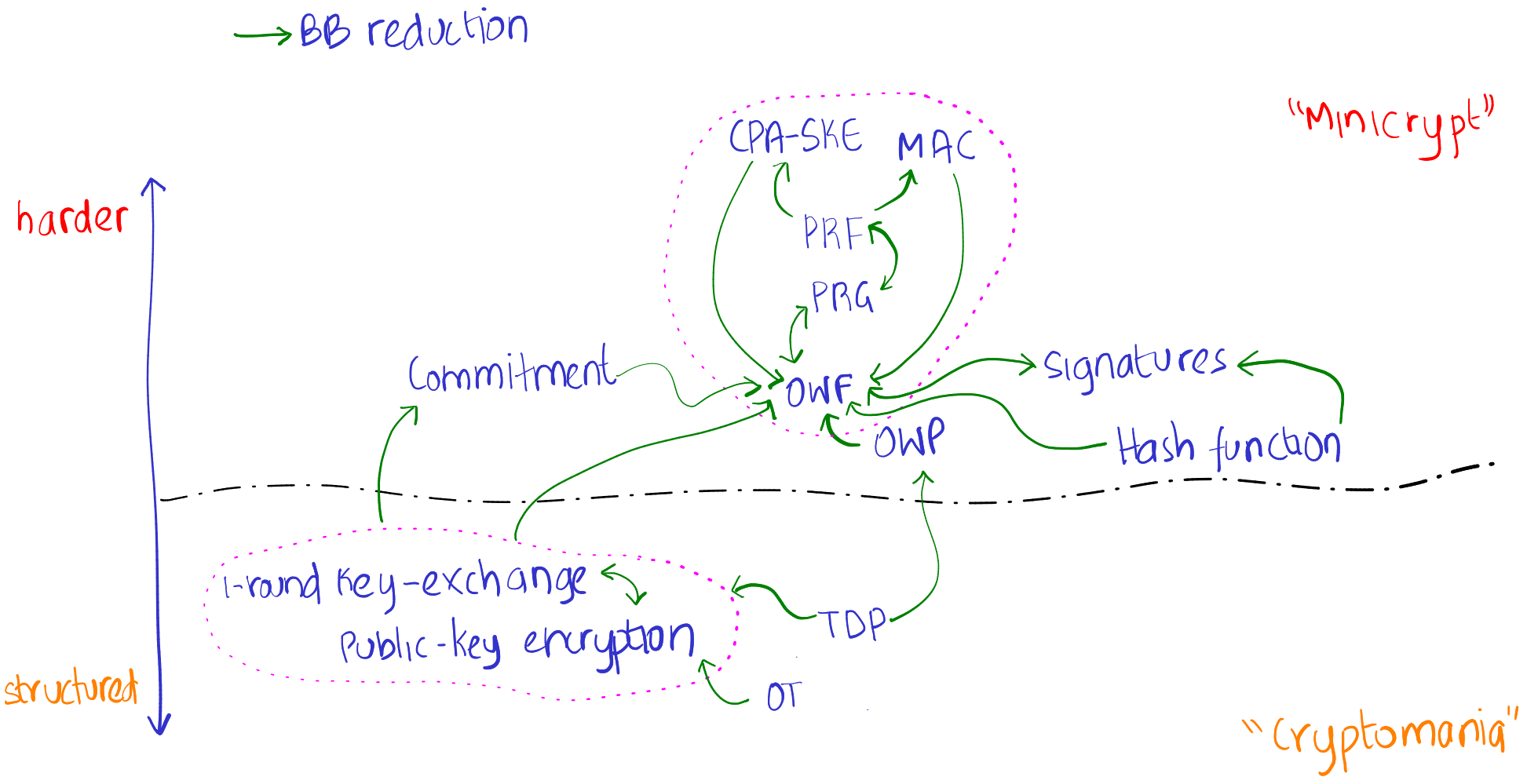
Recall Our Landscape



Recall Our Landscape

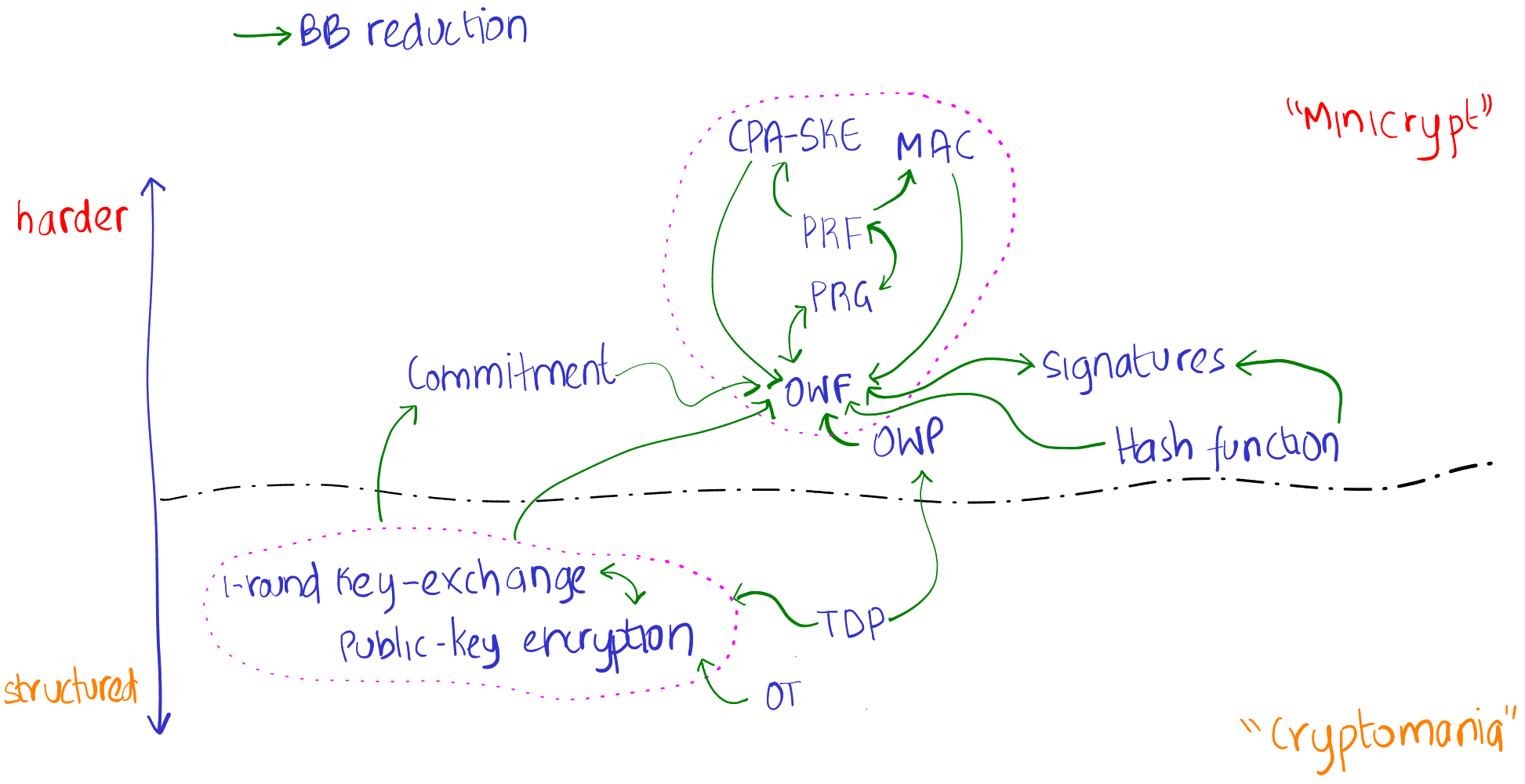


Recall Our Landscape



- What about OWF → OWP or OWF → PKE?

Recall Our Landscape



- What about OWF → OWP or OWF → PKE? We don't know

“Separating” OWF from OWP

Show that OWF exists but OWP doesn't?

“Separating” OWF from OWP

- Show that OWF exists but OWP doesn't?
  - Both OWF and OWP are believed to exist
  - Implies $\mathsf{P} \neq \mathsf{NP}$

“Separating” OWF from OWP

- Show that OWF exists but OWP doesn't?
  - ⚠ Both OWF and OWP are believed to exist
  - ⚠ Implies $\mathsf{P} \neq \mathsf{NP}$

OWF $\not\to$ OWP

- Instead show that there is no BB reduction of OWP to OWF

“Separating” OWF from OWP

- Show that OWF exists but OWP doesn't?
  - ⚠ Both OWF and OWP are believed to exist
  - ⚠ Implies $\mathbf{P} \neq \mathbf{NP}$

OWF ↛ OWP

- Instead show that there is no BB reduction of OWP to OWF
- We must show (by negating Defintion 1) that:
  - ∀ 1 for every BB reduction $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ of OWP to OWF
  - ∃ 2 there exists a OWF F and a OWP-inverter $\mathbf{PInv}^{\cdot}$ such that
  - — 3 $\mathbf{PInv}$ inverts Π^{F} but $\mathbf{FInv}^{\mathbf{PInv},F}$ does not invert F

"Separating" OWF from OWP

Show that OWF exists but OWP doesn't?

⚠ Both OWF and OWP are believed to exist

⚠ Implies $\mathbf{P} \neq \mathbf{NP}$

OWF ↛ OWP

- Instead show that there is no BB reduction of OWP to OWF
- We must show (by negating Defintion 1) that:

∀ 1 for every BB reduction $(\Pi^{\cdot}, \mathbf{FInv}^{\cdot,\cdot})$ of OWP to OWF

∃ 2 there exists a OWF F and a OWP-inverter $\mathbf{PInv}^{\cdot}$ such that

— 3 $\mathbf{PInv}$ inverts Π^{F} but $\mathbf{FInv}^{\mathbf{PInv},F}$ does not invert F

∃

- By Claim 1, suffices to show there exists oracle O such that:

∀ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF

∃ 2 there exists a OWF F^{O} and a OWP-inverter $\mathbf{PInv}^{\cdot,O}$ such that

— 3 $\mathbf{PInv}^{\cdot,O}$ inverts $\Pi^{F,O}$ but $\mathbf{FInv}^{\mathbf{PInv},F,O}$ does not invert F^{O}

Plan for this Lecture

1. Black-Box Reduction $\rightarrow$
2. Black-Box Separation $\not\rightarrow$
3. Black-Box Separating OWF from OWP $\text{OWF} \not\rightarrow \text{OWP}$

High-Level Idea of the Separation

- We will come up with a "helper" oracle O such that

think of $O = \mathrm{PSPACE}$

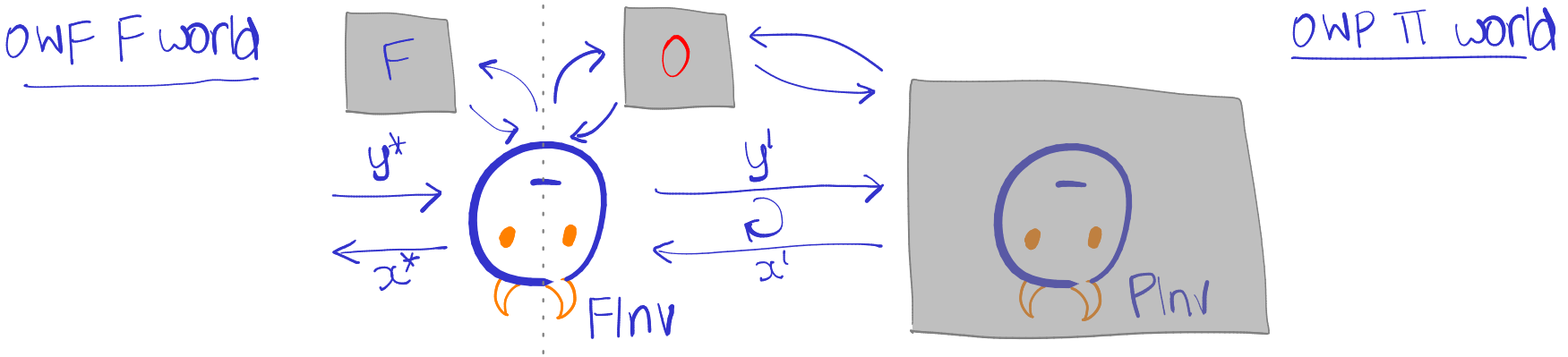
High-Level Idea of the Separation

- We will come up with a "helper" oracle O such that
  - $\forall$ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - $\exists$ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$

High-Level Idea of the Separation

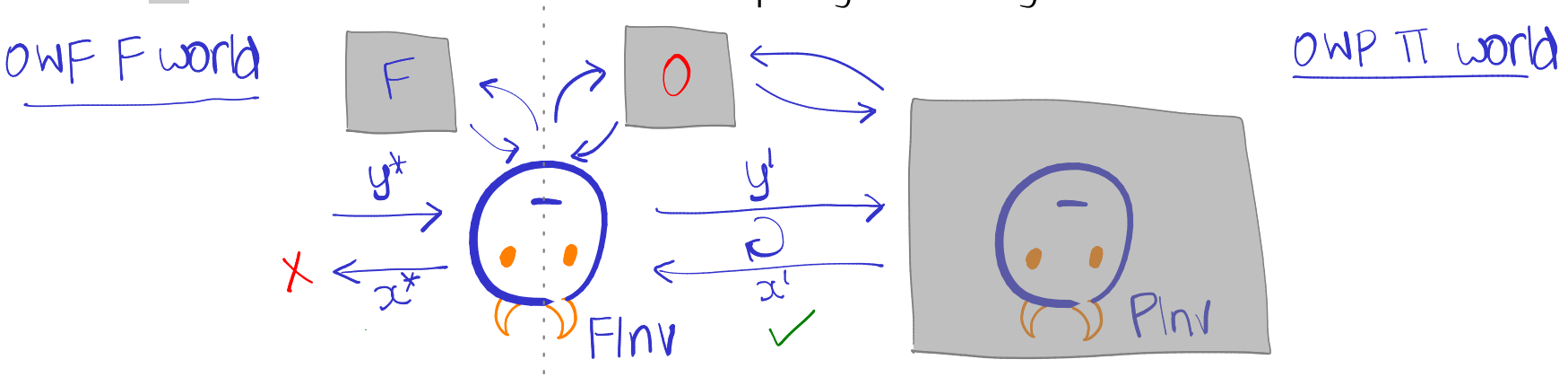
- We will come up with a "helper" oracle O such that
  - $\forall$ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - $\exists$ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$

think of O = PSPACE



High-Level Idea of the Separation

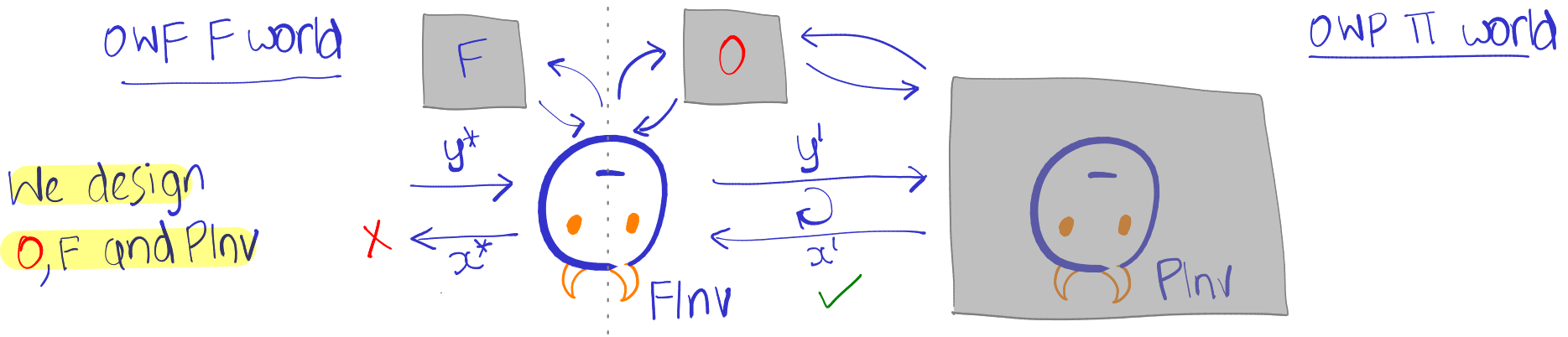
- We will come up with a "helper" oracle O such that
  - ∀ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - ∃ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$



  - — 3 $\mathbf{PInv}^{\cdot,O}$ inverts $\Pi^{F,O}$ but $\mathbf{FInv}^{\mathbf{PInv},F,O}$ does not invert F^O

High-Level Idea of the Separation

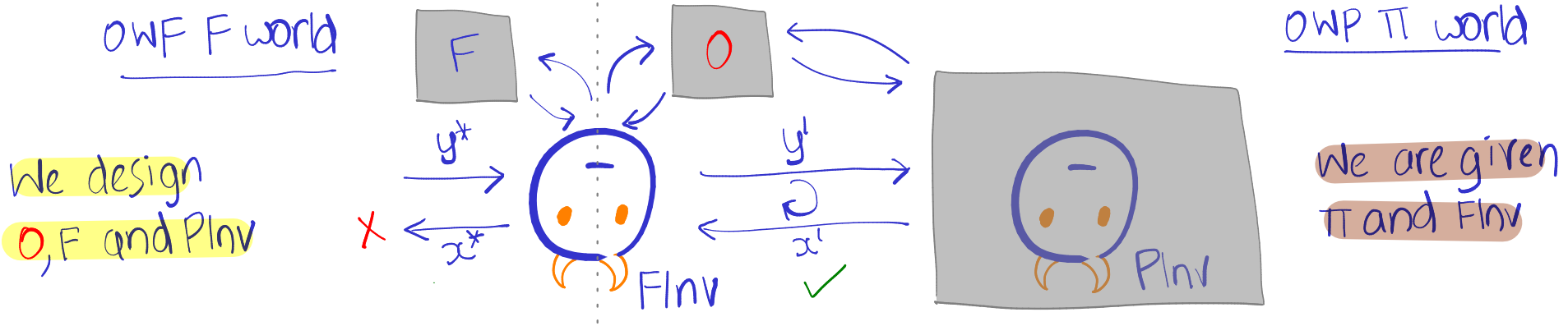
- We will come up with a "helper" oracle O such that (think of O = PSPACE)
  - ∀ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - ∃ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$



- 3 $\mathbf{PInv}^{\cdot,O}$ inverts $\Pi^{F,O}$ but $\mathbf{FInv}^{\mathbf{PInv},F,O}$ does not invert F^O

High-Level Idea of the Separation

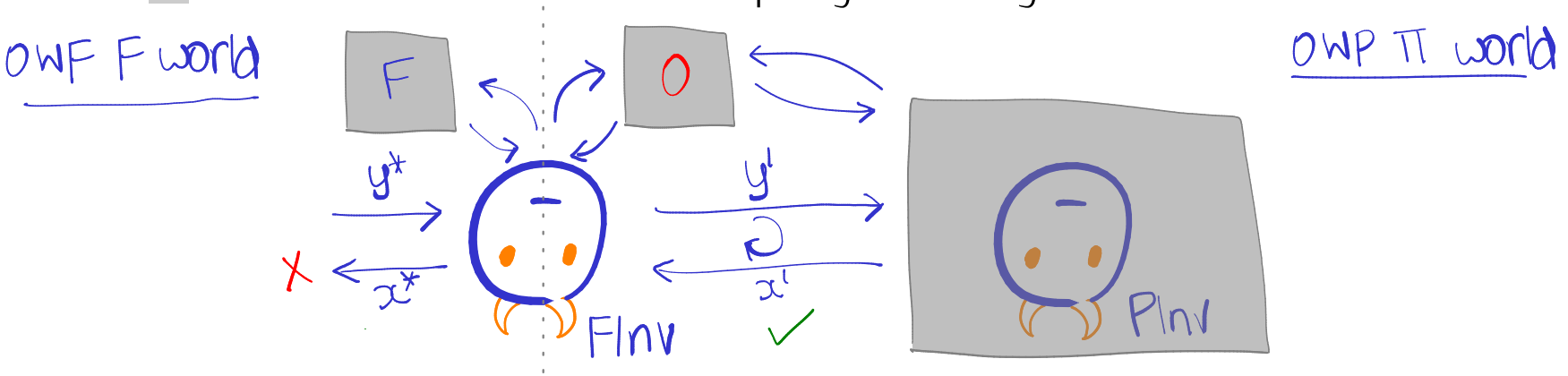
- We will come up with a "helper" oracle O such that (think of O = PSPACE)
  - ∀ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - ∃ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$



- 3 $\mathbf{PInv}^{\cdot,O}$ inverts $\Pi^{F,O}$ but $\mathbf{FInv}^{\mathbf{PInv},F,O}$ does not invert F^O

High-Level Idea of the Separation

- We will come up with a "helper" oracle O such that
  - ∀ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - ∃ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$

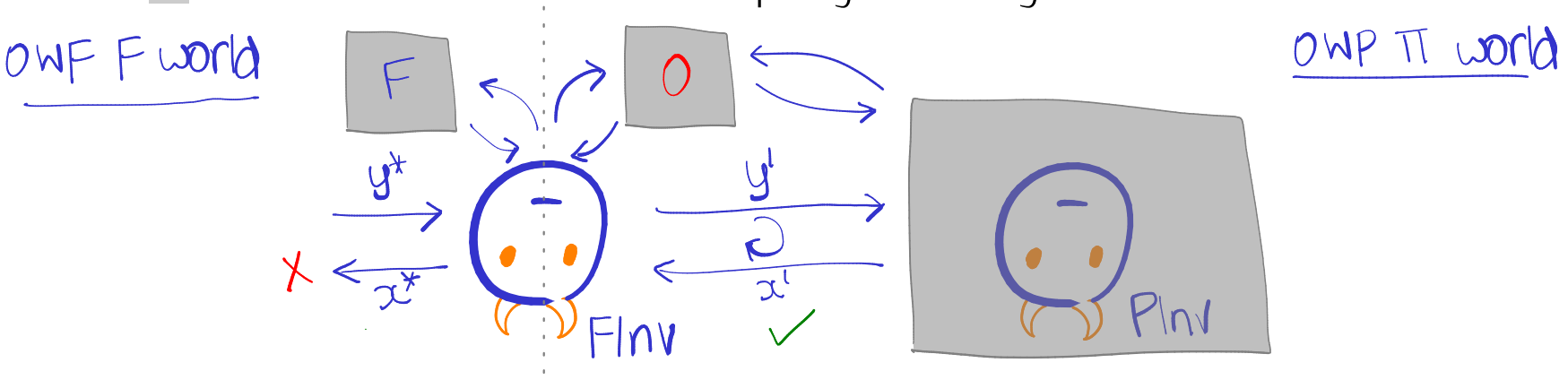


  - — 3 $\mathbf{PInv}^{\cdot,O}$ inverts $\Pi^{F,O}$ but $\mathbf{FInv}^{\mathbf{PInv},F,O}$ does not invert F^O

- Step I: design query-learning $\mathbf{PInv}$ that *efficiently* breaks OWP Π^F given access to O
  - Idea: exploit the fact that Π^F is a permutation for *any* F

High-Level Idea of the Separation

- We will come up with a "helper" oracle O such that
  - ∀ 1 for every BB reduction $(\Pi^{\cdot,O}, \mathbf{FInv}^{\cdot,\cdot,O})$ of OWP to OWF
  - ∃ 2 for *random oracle* F and "query-learning" $\mathbf{PInv}^{\cdot,O}$

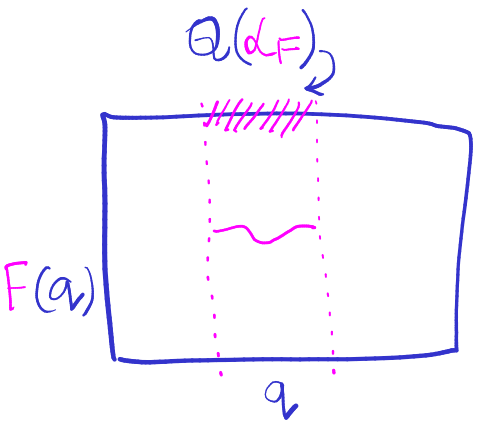


  - — 3 $\mathbf{PInv}^{\cdot,O}$ inverts $\Pi^{F,O}$ but $\mathbf{FInv}^{\mathbf{PInv},F,O}$ does not invert F^O

- Step I: design query-learning **PInv** that *efficiently* breaks OWP Π^F given access to O
  - Idea: exploit the fact that Π^F is a permutation for *any* F
- Step II: show **FInv** can't break random oracle F even given O
  - Idea: random oracle output is unpredictable ⇒ one-wayness

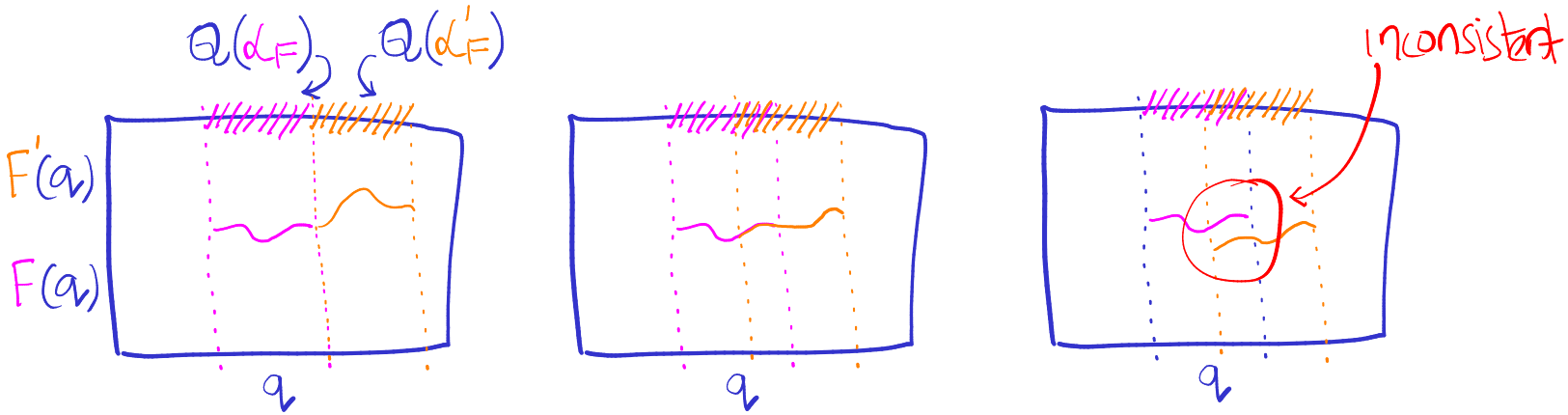
Step I: Design Query-Learning PInv...

- Notation/observations:
  - $\mathcal{L}_F$: a list of query-response pairs of some function F
    - $(q, r) \in \mathcal{L}_F \Rightarrow r = F(q)$
    - $Q(\mathcal{L}_F)$: set of *queries* in $\mathcal{L}_F$, i.e., $\{q : (q, r) \in \mathcal{L}_F\}$



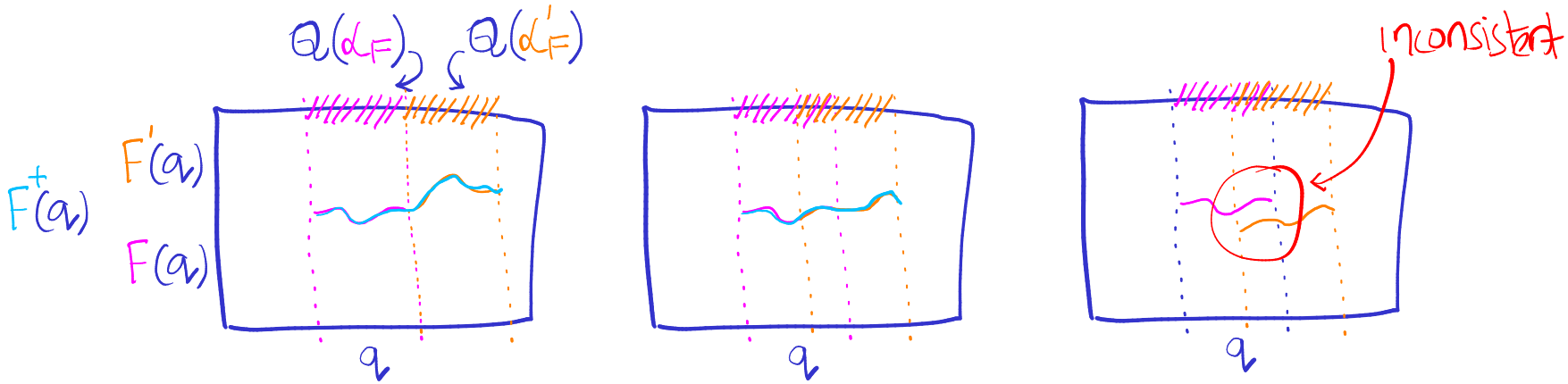
Step I: Design Query-Learning PInv...

- Notation/observations:
  - $\mathcal{L}_F$: a list of query-response pairs of some function F
    - $(q, r) \in \mathcal{L}_F \Rightarrow r = F(q)$
    - $Q(\mathcal{L}_F)$: set of *queries* in $\mathcal{L}_F$, i.e., $\{q : (q, r) \in \mathcal{L}_F\}$
    - $\mathcal{L}_{F'}$ consistent with $\mathcal{L}_F$ if $\forall q \in Q(\mathcal{L}_F) \cap Q(\mathcal{L}_{F'})$: $F(q) = F'(q)$



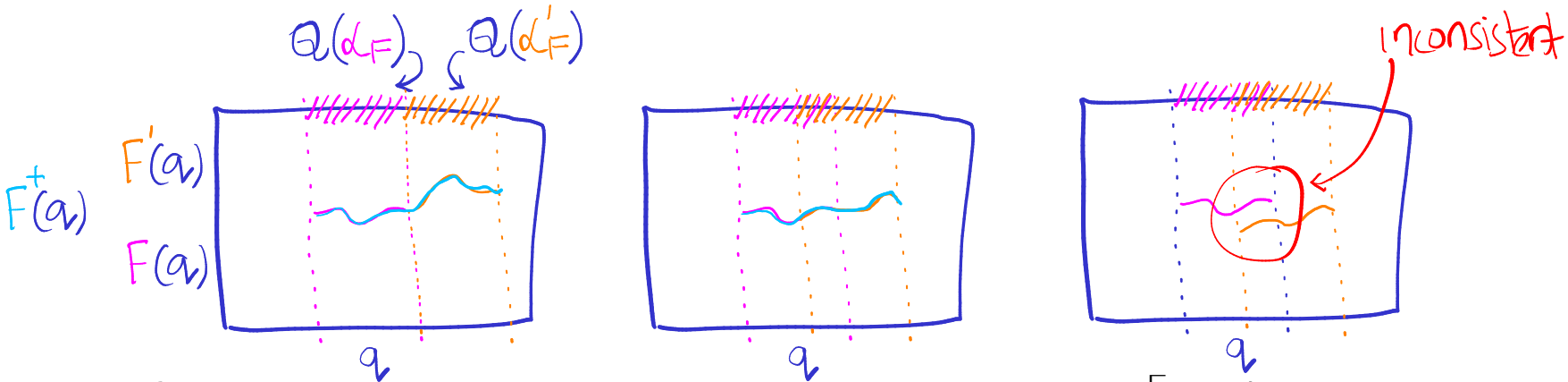
Step I: Design Query-Learning PInv...

- Notation/observations:
  - $\mathcal{L}_F$: a list of query-response pairs of some function F
    - $(q, r) \in \mathcal{L}_F \Rightarrow r = F(q)$
    - $Q(\mathcal{L}_F)$: set of *queries* in $\mathcal{L}_F$, i.e., $\{q : (q, r) \in \mathcal{L}_F\}$
  - $\mathcal{L}_{F'}$ consistent with $\mathcal{L}_F$ if $\forall q \in Q(\mathcal{L}_F) \cap Q(\mathcal{L}_{F'})$: $F(q) = F'(q)$
  - Consistent $\mathcal{L}_F$ and $\mathcal{L}_{F'}$ can be "merged" into $\mathcal{L}_{F^+} := \mathcal{L}_F \cup \mathcal{L}_{F'}$



Step I: Design Query-Learning PInv...

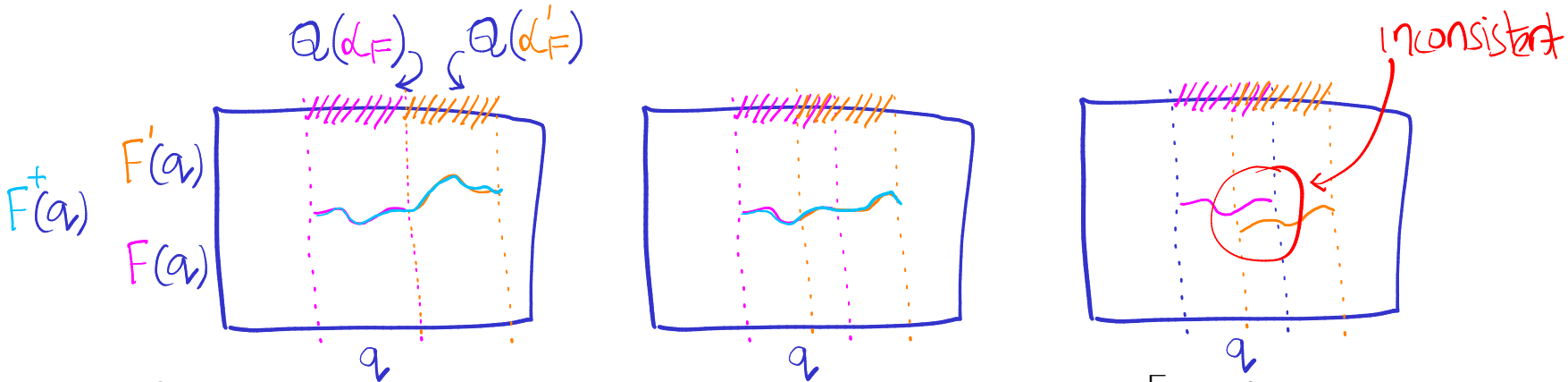
- Notation/observations:
  - $\mathcal{L}_F$: a list of query-response pairs of some function F
    - $(q, r) \in \mathcal{L}_F \Rightarrow r = F(q)$
    - $Q(\mathcal{L}_F)$: set of *queries* in $\mathcal{L}_F$, i.e., $\{q : (q, r) \in \mathcal{L}_F\}$
    - $\mathcal{L}_{F'}$ consistent with $\mathcal{L}_F$ if $\forall q \in Q(\mathcal{L}_F) \cap Q(\mathcal{L}_{F'})$: $F(q) = F'(q)$
    - Consistent $\mathcal{L}_F$ and $\mathcal{L}_{F'}$ can be "merged" into $\mathcal{L}_{F^+} := \mathcal{L}_F \cup \mathcal{L}_{F'}$



  - $\mathcal{C}_{F,x}$: query-answer pairs involved in *computing* $\Pi^F(x)$ for some function F and input x (in F's domain)
    - $Q(\mathcal{C}_{F,x})$: set of *queries* in $\mathcal{C}_{F,x}$, i.e., $\{q : (q, r) \in \mathcal{C}_{F,x}\}$

Step I: Design Query-Learning PInv...

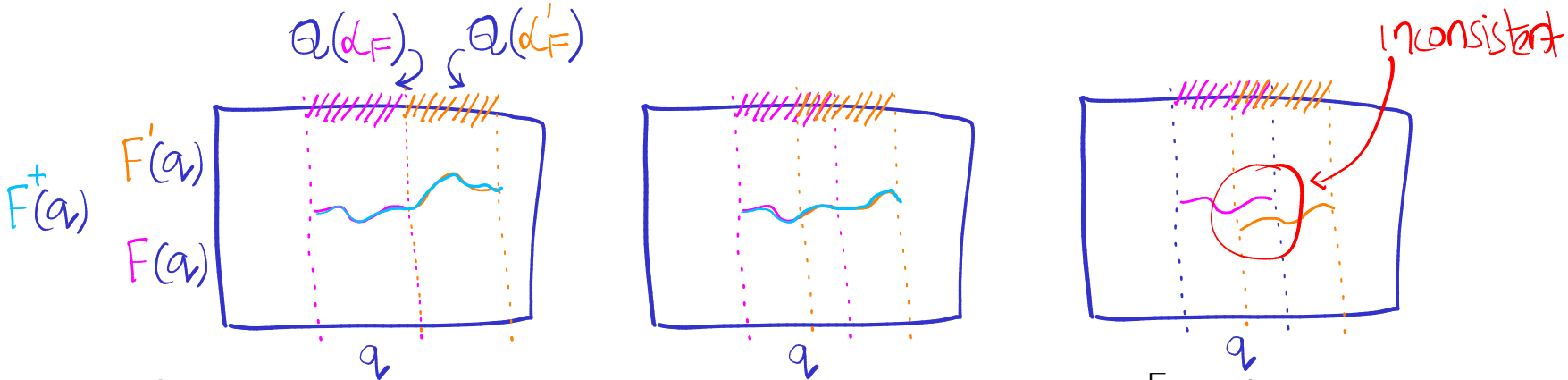
- Notation/observations:
  - $\mathcal{L}_F$: a list of query-response pairs of some function F
    - $(q, r) \in \mathcal{L}_F \Rightarrow r = F(q)$
    - $Q(\mathcal{L}_F)$: set of *queries* in $\mathcal{L}_F$, i.e., $\{q : (q, r) \in \mathcal{L}_F\}$
    - $\mathcal{L}_{F'}$ consistent with $\mathcal{L}_F$ if $\forall q \in Q(\mathcal{L}_F) \cap Q(\mathcal{L}_{F'})$: $F(q) = F'(q)$
    - Consistent $\mathcal{L}_F$ and $\mathcal{L}_{F'}$ can be "merged" into $\mathcal{L}_{F^+} := \mathcal{L}_F \cup \mathcal{L}_{F'}$



  - $\mathcal{C}_{F,x}$: query-answer pairs involved in *computing* $\Pi^F(x)$ for some function F and input x (in F's domain)
    - $Q(\mathcal{C}_{F,x})$: set of *queries* in $\mathcal{C}_{F,x}$, i.e., $\{q : (q, r) \in \mathcal{C}_{F,x}\}$
    - $\Pi^F(x)$ is determined once $\mathcal{C}_{F,x}$ fixed
    - Π is efficient $\Rightarrow$ for every x and F, $|\mathcal{C}_{F,x}|$ is polynomial

Step I: Design Query-Learning PInv...

- Notation/observations:
  - $\mathcal{L}_F$: a list of query-response pairs of some function F
    - $(q, r) \in \mathcal{L}_F \Rightarrow r = F(q)$
    - $Q(\mathcal{L}_F)$: set of *queries* in $\mathcal{L}_F$, i.e., $\{q : (q, r) \in \mathcal{L}_F\}$
    - $\mathcal{L}_{F'}$ consistent with $\mathcal{L}_F$ if $\forall q \in Q(\mathcal{L}_F) \cap Q(\mathcal{L}_{F'})$: $F(q) = F'(q)$
    - Consistent $\mathcal{L}_F$ and $\mathcal{L}_{F'}$ can be "merged" into $\mathcal{L}_{F^+} := \mathcal{L}_F \cup \mathcal{L}_{F'}$



  - $\mathcal{C}_{F,x}$: query-answer pairs involved in *computing* $\Pi^F(x)$ for some function F and input x (in F's domain)
    - $Q(\mathcal{C}_{F,x})$: set of *queries* in $\mathcal{C}_{F,x}$, i.e., $\{q : (q, r) \in \mathcal{C}_{F,x}\}$
    - $\Pi^F(x)$ is determined once $\mathcal{C}_{F,x}$ fixed
    - Π is efficient $\Rightarrow$ for every x and F, $|\mathcal{C}_{F,x}|$ is polynomial
  - Main observation: (partial F and F' consistent) and $(\Pi^F(x) = \Pi^{F'}(x') = y) \Rightarrow Q(\mathcal{C}_{F,x}) \cap Q(\mathcal{C}_{F',x'}) \neq \emptyset$. Why?

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$) $y = \Pi^F(\alpha)$

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

RO ↖ ↗ PSPACE

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$) $y = \Pi^F(\alpha)$

Step I: Design Query-Learning PInv...

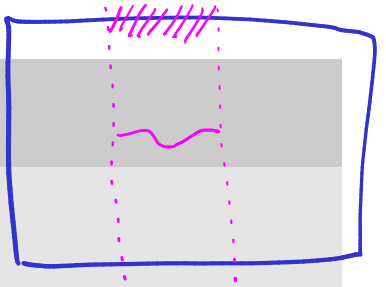
strategy: learn at least one new query from $C_{F,x}$ per round

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1 *Initiate list* $\mathcal{L}_F := \emptyset$ query-answer pairs of F we have learnt so far

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

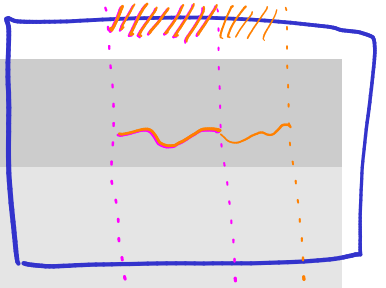


Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$*:* F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq C_{F',x'}$ *and* $\Pi^{F'}(x') = y$*:* x' *is a valid inverse w.r.to.* F'

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

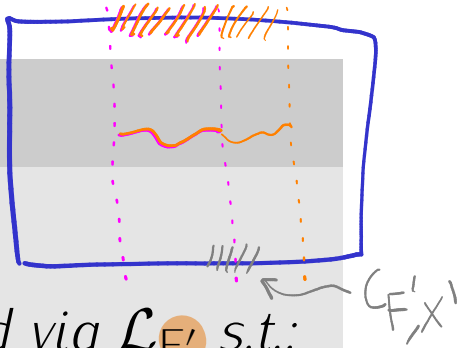


Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$*:* F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq C_{F',x'}$ *and* $\Pi^{F'}(x') = y$*:* x' *is a valid inverse w.r.to.* F'

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

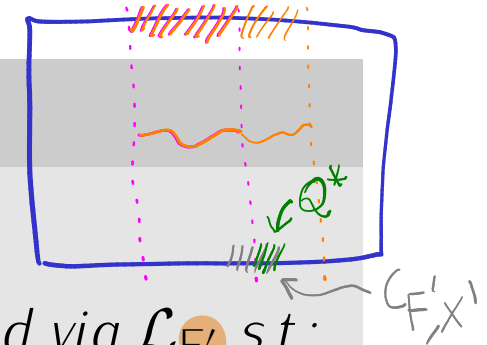


Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.to.* F'

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

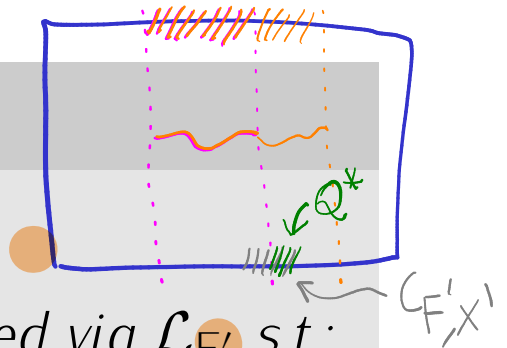


Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$*:* F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$*:* x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F*: if* $\Pi^F(x') = y$*, output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round

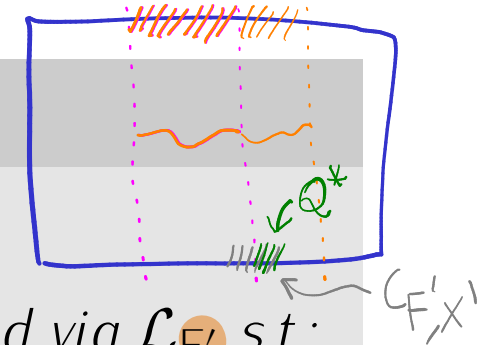


Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F*: if* $\Pi^{F}(x') = y$*, output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := \mathrm{Q}(\mathcal{C}_{F',x'}) \setminus \mathrm{Q}(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*

Step I: Design Query-Learning PInv...

strategy: learn at least one new query from $C_{F,x}$ per round



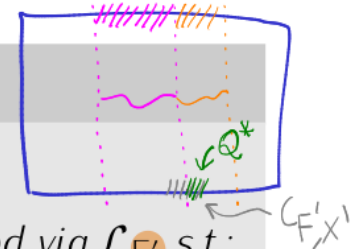
Construction 2 (Query-learning inverter $\mathsf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$*:* F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$*:* x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F*: if* $\Pi^F(x') = y$*, output* x' *and halt*
4. *Query* F *with "fresh" queries* $Q^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*

Lemma 2

If $O = \mathsf{PSPACE}$*,* PInv *outputs* $x : \Pi^F(x) = y$ *in polynomial time*

Why does PInv Work?



Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

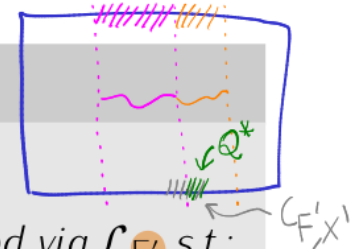
1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F: *if* $\Pi^F(x') = y$, *output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*

Proof of Lemma 2 (PInv outputs $x : \Pi^F(x) = y$ in polynomial time).

1. Claim 1: PInv outputs x such that $\Pi^F(x) = y$ and halts

Why does PInv Work?

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)



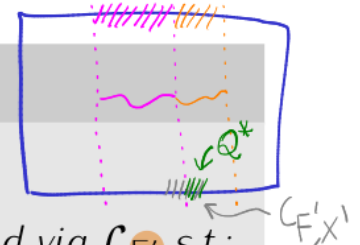
1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F: *if* $\Pi^F(x') = y$, *output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*

Proof of Lemma 2 (PInv outputs $x : \Pi^F(x) = y$ in polynomial time).

1. Claim 1: PInv outputs x such that $\Pi^F(x) = y$ and halts
   - Sub-claim: $\mathcal{Q}^*$ contains at least one query $q^* \in Q(\mathcal{C}_{F,x} \setminus \mathcal{L}_F)$.
     - (?) Why?

Why does PInv Work?

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,\mathsf{F},\mathsf{O}}(y)$)



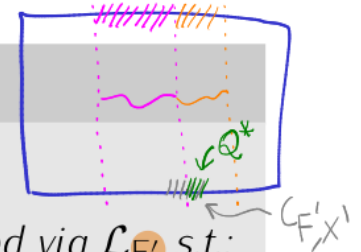
1. *Initiate list* $\mathcal{L}_{\mathsf{F}} := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{\mathsf{F}'}$ *s.t.:*
   1. $\mathcal{L}_{\mathsf{F}'} \supseteq \mathcal{L}_{\mathsf{F}}$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{\mathsf{F}'} \supseteq \mathcal{C}_{\mathsf{F}',x'}$ *and* $\Pi^{\mathsf{F}'}(x') = y$: x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F: *if* $\Pi^{\mathsf{F}}(x') = y$, *output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := \mathrm{Q}(\mathcal{C}_{\mathsf{F}',x'}) \setminus \mathrm{Q}(\mathcal{L}_{\mathsf{F}})$ *and add these query-response pairs to* $\mathcal{L}_{\mathsf{F}}$
5. *Repeat from Step 2*

Proof of Lemma 2 ($\mathbf{PInv}$ outputs $x : \Pi^{\mathsf{F}}(x) = y$ in polynomial time).

1. Claim 1: $\mathbf{PInv}$ outputs x such that $\Pi^{\mathsf{F}}(x) = y$ and halts
   - Sub-claim: $\mathcal{Q}^*$ contains at least one query $q^* \in \mathrm{Q}(\mathcal{C}_{\mathsf{F},x} \setminus \mathcal{L}_{\mathsf{F}})$.
     - (?) Why? Otherwise, possible to "merge" F and F' into F^+ such that collision! $\hookrightarrow$ $\Pi^{\mathsf{F}^+}(x') = \Pi^{\mathsf{F}^+}(x) = y$. How?
   - Sub-claim $\Rightarrow$ Claim 1 since $|\mathcal{C}_{\mathsf{F},x}|$ is polynomial $\square$

Why does PInv Work?

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,\mathsf{F},\mathsf{O}}(y)$)



1. *Initiate list* $\mathcal{L}_{\mathsf{F}} := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{\mathsf{F}'}$ *s.t.:*
   1. $\mathcal{L}_{\mathsf{F}'} \supseteq \mathcal{L}_{\mathsf{F}}$*:* F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{\mathsf{F}'} \supseteq \mathcal{C}_{\mathsf{F}',x'}$ *and* $\Pi^{\mathsf{F}'}(x') = y$*:* x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F*: if* $\Pi^{\mathsf{F}}(x') = y$*, output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := \mathsf{Q}(\mathcal{C}_{\mathsf{F}',x'}) \setminus \mathsf{Q}(\mathcal{L}_{\mathsf{F}})$ *and add these query-response pairs to* $\mathcal{L}_{\mathsf{F}}$
5. *Repeat from Step 2*

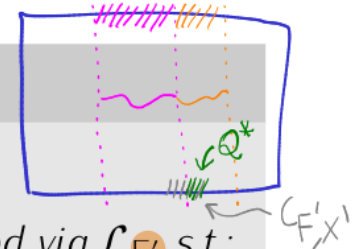
Proof of Lemma 2 (PInv outputs $x : \Pi^{\mathsf{F}}(x) = y$ in polynomial time).

2. Claim 2: PInv is efficient given access to (e.g.) PSPACE O.

Why does PInv Work?

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F: *if* $\Pi^F(x') = y$, *output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*



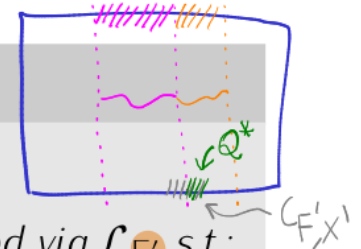
Proof of Lemma 2 (PInv outputs $x : \Pi^F(x) = y$ in polynomial time).

2. Claim 2: PInv is efficient given access to (e.g.) PSPACE O.
   - Sub-claim: in Step 2 there *must* exist (x', F') s.t. $\Pi^{F'}(x') = y$
     - (?) Why?

Why does PInv Work?

Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi^\cdot, F, O}(y)$)

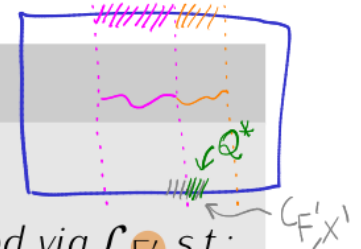
1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.t.* F'
3. *Test* x' *on* F: *if* $\Pi^F(x') = y$, *output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*



Proof of Lemma 2 (PInv outputs $x : \Pi^F(x) = y$ in polynomial time).

2. Claim 2: PInv is efficient given access to (e.g.) PSPACE O.
   - Sub-claim: in Step 2 there *must* exist (x', F') s.t. $\Pi^{F'}(x') = y$
     - Why? $\Pi^\cdot$ is always a permutation
     - Only need to fix $\mathcal{C}_{F',x'}$ to determine $\Pi^{F'}(x') \Rightarrow$ suffices to fix $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$

Why does PInv Work?



Construction 2 (Query-learning inverter $\mathbf{PInv}^{\Pi,F,O}(y)$)

1. *Initiate list* $\mathcal{L}_F := \emptyset$
2. *Use* O *to find input* x' *&partial function* F' *defined via* $\mathcal{L}_{F'}$ *s.t.:*
   1. $\mathcal{L}_{F'} \supseteq \mathcal{L}_F$: F' *is consistent with all we know so far about* F
   2. $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$ *and* $\Pi^{F'}(x') = y$: x' *is a valid inverse w.r.to.* F'
3. *Test* x' *on* F: *if* $\Pi^F(x') = y$, *output* x' *and halt*
4. *Query* F *with "fresh" queries* $\mathcal{Q}^* := Q(\mathcal{C}_{F',x'}) \setminus Q(\mathcal{L}_F)$ *and add these query-response pairs to* $\mathcal{L}_F$
5. *Repeat from Step 2*

Proof of Lemma 2 (PInv outputs $x : \Pi^F(x) = y$ in polynomial time).

2. Claim 2: PInv is efficient given access to (e.g.) PSPACE O.
   - Sub-claim: in Step 2 there *must* exist (x', F') s.t. $\Pi^{F'}(x') = y$
     - Why? Π is always a permutation
     - Only need to fix $\mathcal{C}_{F',x'}$ to determine $\Pi^{F'}(x') \Rightarrow$ suffices to fix $\mathcal{L}_{F'} \supseteq \mathcal{C}_{F',x'}$
   - Framed as NP language: $\{(y, \mathcal{L}_F), (\mathcal{L}'_F, x') : 2.1 \text{ and } 2.2 \text{ holds}\}$

What is PInv's run-time? □

Step II: Show F is One-Way Even Given O

Claim 2

For any fixed, efficient $\mathsf{FInv}^{\cdot}$*, the following is negligible*

$$\Pr_{F,x}[\mathsf{FInv}^{F}(F(x)) \in F^{-1}(F(x))]$$

Step II: Show F is One-Way Even Given O

Claim 2

For any fixed, efficient $\mathsf{FInv}^{\cdot}$*, the following is negligible*

$$\Pr_{\mathsf{F},x}[\mathsf{FInv}^{\mathsf{F}}(\mathsf{F}(x)) \in \mathsf{F}^{-1}(\mathsf{F}(x))]$$

Proof idea: random oracles are unpredictable $\Rightarrow$ one-wayness.

- $\mathsf{FInv}^{\cdot}$ efficient $\Rightarrow$ $\mathsf{FInv}^{\cdot}$ can make a fixed polynomial number of queries to F
- $\mathsf{FInv}^{\cdot}$ can only win if it queries an x' such that $\mathsf{F}(x') = \mathsf{F}(x)$
- Probability of this event for its each query is exactly $1/2^{|\mathsf{F}(x)|}$
- Claim follows by union bound over all its queries $\square$

Step II: Show F is One-Way Even Given $\mathcal{O}$

Claim 2

For any fixed, efficient $\mathsf{FInv}^{\cdot}$*, the following is negligible*

$$\Pr_{\mathsf{F},x}[\mathsf{FInv}^{\mathsf{F}}(\mathsf{F}(x)) \in \mathsf{F}^{-1}(\mathsf{F}(x))]$$

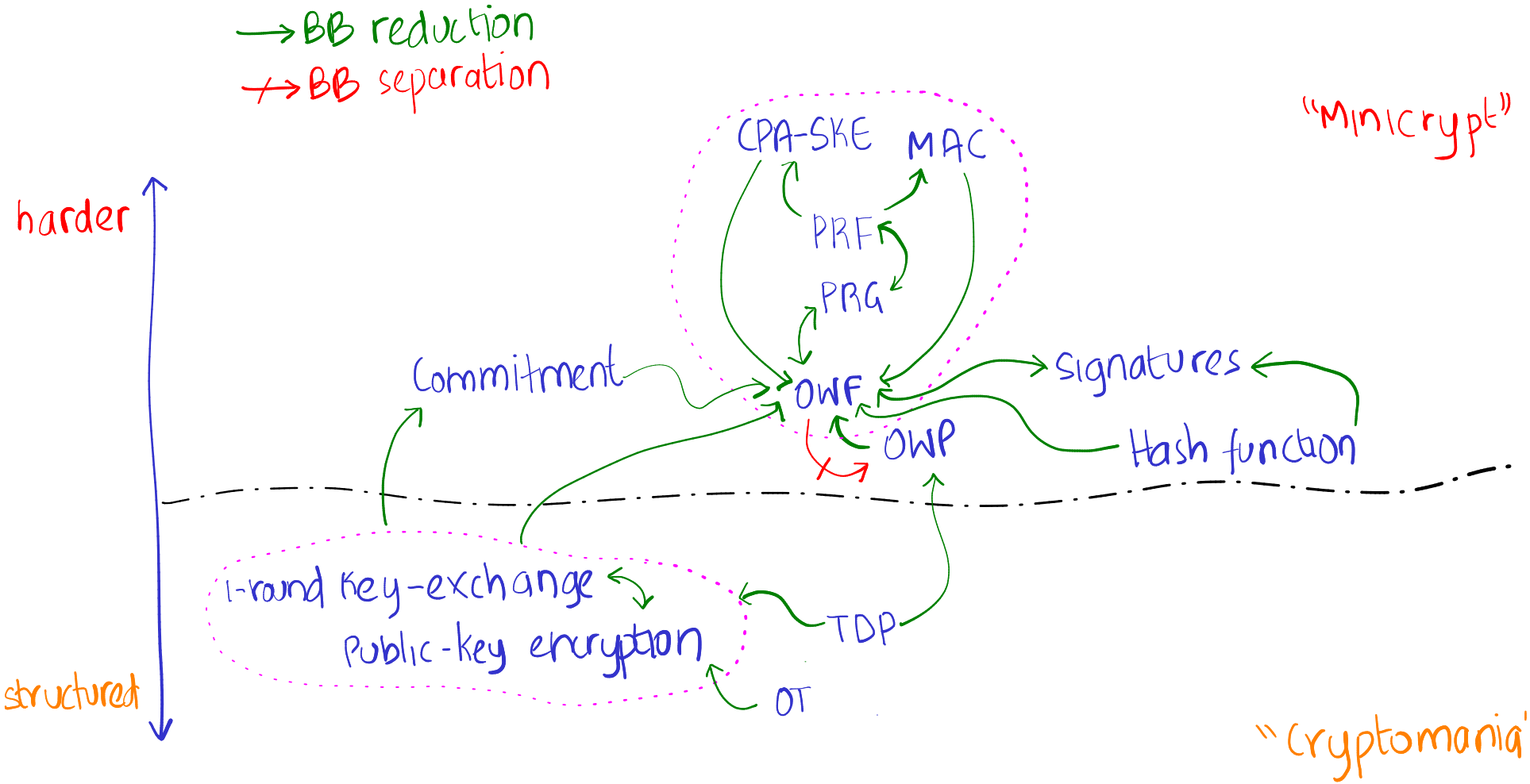
Proof idea: random oracles are unpredictable $\Rightarrow$ one-wayness.

- $\mathsf{FInv}^{\cdot}$ efficient $\Rightarrow$ $\mathsf{FInv}^{\cdot}$ can make a fixed polynomial number of queries to F
- $\mathsf{FInv}^{\cdot}$ can only win if it queries an x' such that $\mathsf{F}(x') = \mathsf{F}(x)$
- Probability of this event for its each query is exactly $1/2^{|\mathsf{F}(x)|}$
- Claim follows by union bound over all its queries $\square$

Exercise 2

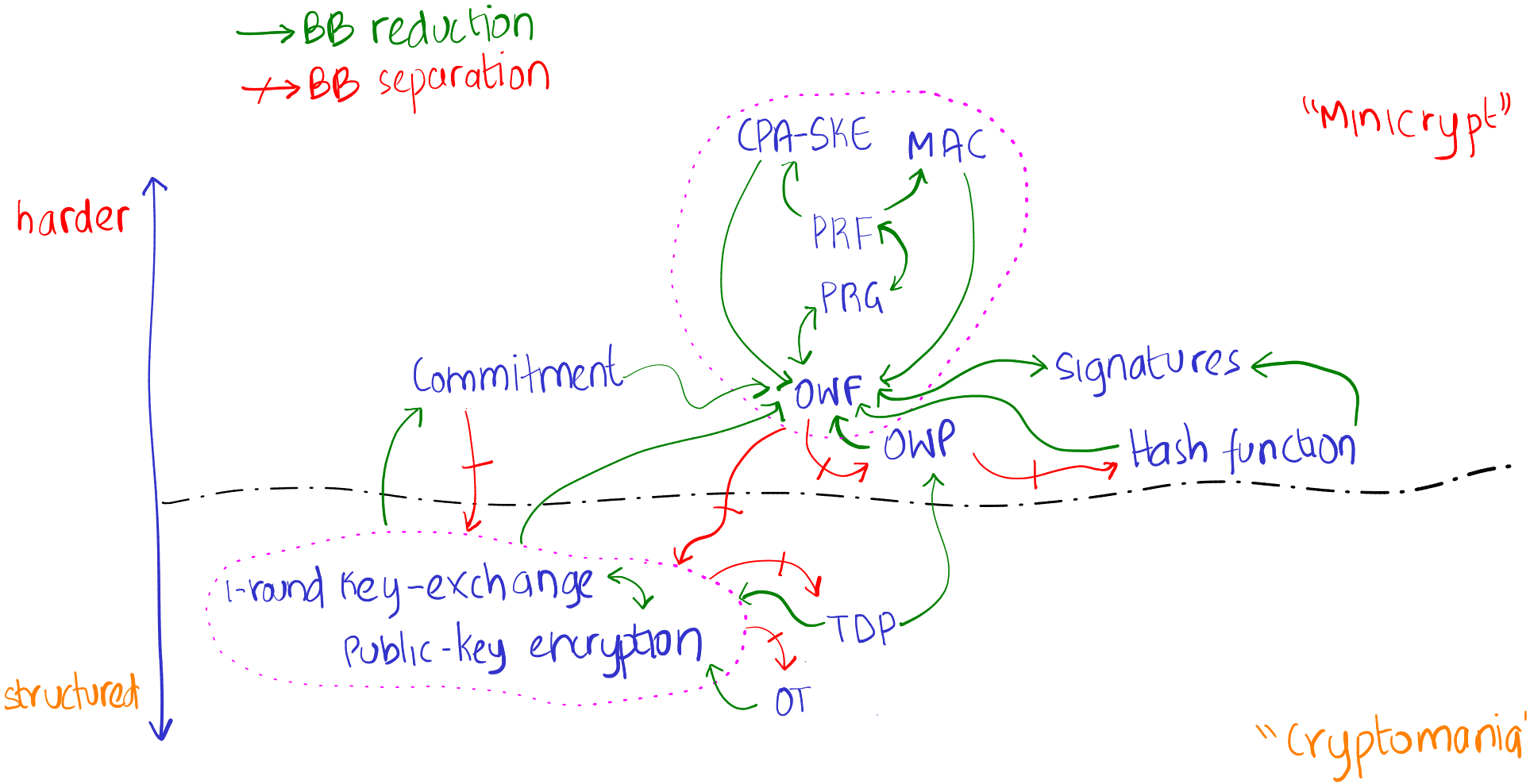
Show that Claim 2 holds also with respect to the PSPACE *oracle* $\mathcal{O}$*.*

What Else Has Been BB Separated?



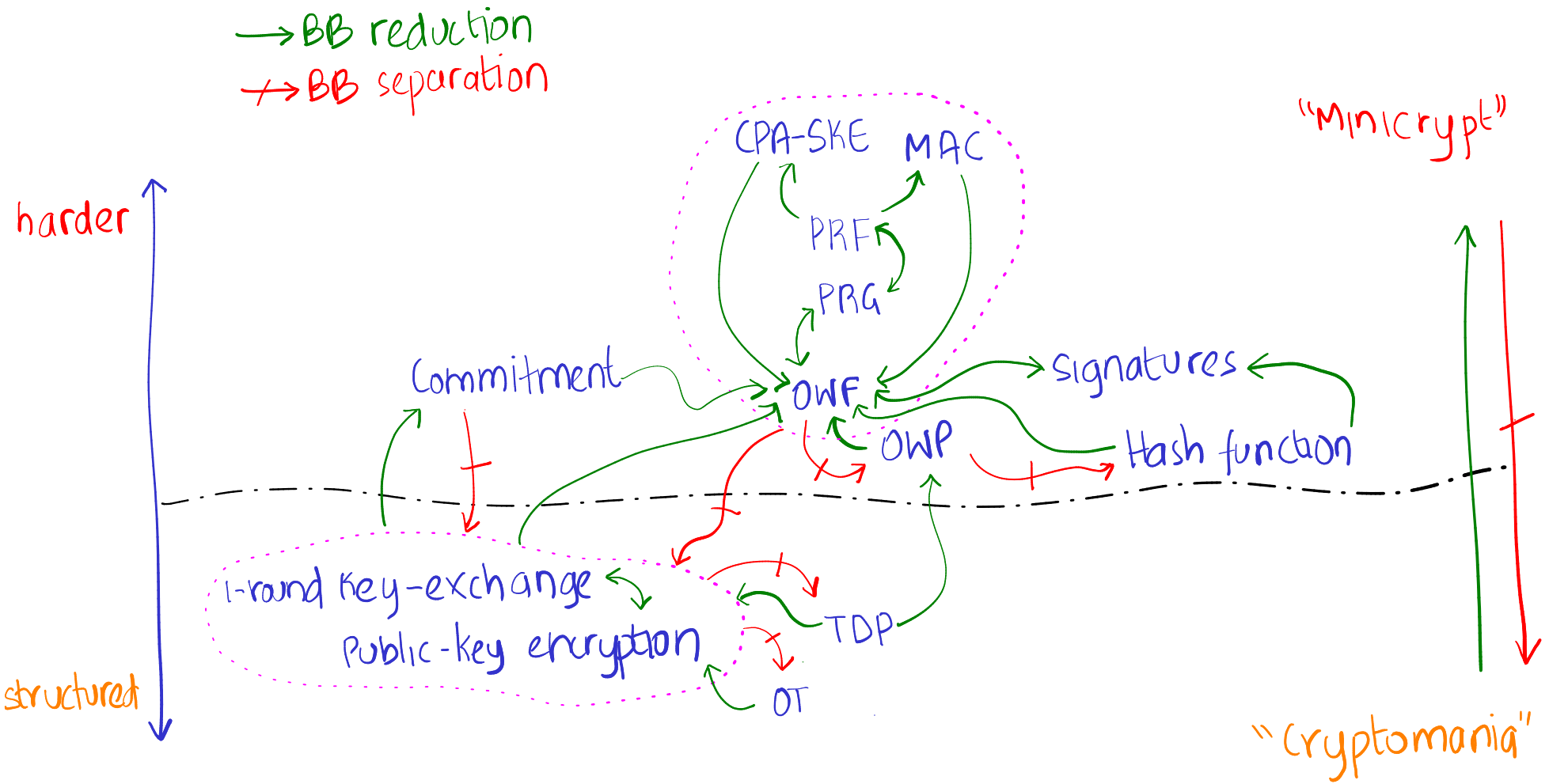
- Most primitives that don't BB-reduce to each other!

What Else Has Been BB Separated?



- Most primitives that don't BB-reduce to each other!

What Else Has Been BB Separated?



- Most primitives that don't BB-reduce to each other!

To Recap Today's Lecture

- Black-box reductions and its limitations

To Recap Today's Lecture

- Black-box reductions and its limitations
- Black-box *separations*
  - Formally defined what it means to separate one primitive from another

To Recap Today's Lecture

- Black-box reductions and its limitations
- Black-box *separations*
  - Formally defined what it means to separate one primitive from another

- Separated OWF from OWP
- Key ideas:
  - Black-box reduction relativises: suffices to come up with an "oracle world" O where OWF exists but OWP doesn't
  - Efficient query-set learning algorithm that exploits perfect correctness of the construction
  - Random oracles are unpredictable, and hence one-way

Next Lecture

- Friday (25/Oct): crib session for Quiz 2

Next Lecture

- Friday (25/Oct): crib session for Quiz 2
- Tuesday (29/Oct): Obfuscation I
  - Virtual black-box (VBB) obfuscation
  - Bypassing separation OWF and OWP using code of the OWF:
    - OWF $\xrightarrow{\text{VBB obfuscation}}$ OWP
  - Impossibility of VBB obfuscation for general programs
  - Way around: relax to indistinguishability obfuscation (IO)

References

1. This lecture is mostly based on [Rud84, RTV04, Yer11]
2. Formal definition of fully BB reduction can be found in [RTV04, Yer11]. You can also find formal definitions of other notions of reductions (semi-BB, relativising etc.) there
3. The black-box separation of OWF from OWP is from Rudich's thesis [Rud84]
4. For more discussion on relativising reductions, refer to [AB09, §4.3]


Sanjeev Arora and Boaz Barak.
Computational Complexity - A Modern Approach.
Cambridge University Press, 2009.


Omer Reingold, Luca Trevisan, and Salil P. Vadhan.
Notions of reducibility between cryptographic primitives.
In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2004.


Steven Rudich.
Limits on the Provable Consequences of One Way Functions.
PhD thesis, University of California at Berkeley, 1984.


Arkady Yerukhimovich.
A Study of Separations in Cryptography: New Results and Models.
PhD thesis, University of Maryland, College Park, 2011.