

Q: Must you know the **code** of  $f$   
to securely compute  $f$ ?

Mike Rosulek |  The University of  
**Montana** | CRYPTO 2012

# black-box reductions

## Reduction

$X$  has an algorithm  $\Rightarrow Y$  has an algorithm

# black-box reductions

## Reduction

$X$  has an algorithm  $\Rightarrow Y$  has an algorithm

**Black-box:**  $\exists B: B^X$  is an algorithm for  $Y$

**Non-black-box:** Algorithm for  $Y$  depends on **code** of algorithm for  $X$

# black-box reductions

## Reduction

$X$  has an algorithm  $\Rightarrow Y$  has an algorithm

**Black-box:**  $\exists B: B^X$  is an algorithm for  $Y$

**Non-black-box:** Algorithm for  $Y$  depends on **code** of algorithm for  $X$

**Pervasive question since [ImpagliazzoRudich89]:**

*When do black-box constructions exist?*

# black-box reductions

## Reduction

$X$  has an algorithm  $\Rightarrow Y$  has an algorithm

**Black-box:**  $\exists B: B^X$  is an algorithm for  $Y$

**Non-black-box:** Algorithm for  $Y$  depends on **code** of algorithm for  $X$

**Pervasive question since [ImpagliazzoRudich89]:**

*When do black-box constructions exist?*

*Black-box constructions tend to be more practical  
(efficient & modular).*

# secure computation...

Several parties wish to carry out an agreed-upon computation.

- ▶ Parties have individual inputs / output
- ▶ Security guarantees:
  - ▶ Privacy (learn no more than your prescribed output)
  - ▶ Input independence
  - ▶ Output consistency, etc..
- ▶ Parties are mutually distrusting, some possibly malicious

# black-box secure computation

## Typical theorem statement:

If trapdoor functions exist, then for **every**  $f$ , there is a secure (in some model) protocol for evaluating  $f$ .

# black-box secure computation

## Typical theorem statement:

If trapdoor functions exist, then for **every**  $f$ , there is a secure (in some model) protocol for evaluating  $f$ .

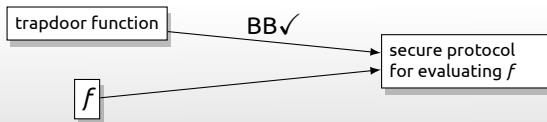




# black-box secure computation

## Typical theorem statement:

If trapdoor functions exist, then for **every**  $f$ , there is a secure (in some model) protocol for evaluating  $f$ .



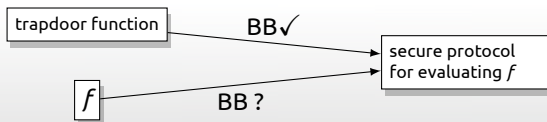
Protocol can be black-box in its **usage of underlying primitives!**

- ▶ [Ishai+06, LindellPinkas07, Haitner08, IshaiPrabhakaranSahai08, Choi+09, PassWee09, ..]

# black-box secure computation

## Typical theorem statement:

If trapdoor functions exist, then for **every**  $f$ , there is a secure (in some model) protocol for evaluating  $f$ .



Protocol can be black-box in its usage of underlying primitives!

- ▶ [Ishai+06, LindellPinkas07, Haitner08, IshaiPrabhakaranSahai08, Choi+09, PassWee09, ..]

What about *usage of f*? Typical approach (since [Yao86,GMW87]):

- ▶ Express  $f$  as a circuit, and evaluate it gate-by-gate — **non-black-box!**

# the model

# the model (2-party SFE)

Let  $\mathcal{C}$  be a class of 2-input functions.

## Definition

**Functionality-black-box (FBB)** secure evaluation of  $\mathcal{C}$  means:

- ▶  $\exists$  oracle machines  $\pi_A, \pi_B$ :
- ▶  $\forall f \in \mathcal{C}$ :
- ▶  $\pi_A^f(x) \stackrel{f}{\rightleftharpoons} \pi_B^f(y)$  is a secure protocol for evaluating  $f(x, y)$

# the model (2-party SFE)

Let  $\mathcal{C}$  be a class of 2-input functions.

## Definition

**Functionality-black-box (FBB)** secure evaluation of  $\mathcal{C}$  means:

- ▶  $\exists$  oracle machines  $\pi_A, \pi_B$ :
- ▶  $\forall f \in \mathcal{C}$ :
- ▶  $\pi_A^f(x) \stackrel{f}{\leftrightarrow} \pi_B^f(y)$  is a secure protocol for evaluating  $f(x, y)$

If protocol uses trusted setup, then **same** setup for all  $f \in \mathcal{C}$ !

# the model (2-party SFE)

Let  $\mathcal{C}$  be a class of 2-input functions.

## Definition

**Functionality-black-box (FBB)** secure evaluation of  $\mathcal{C}$  means:

- ▶  $\exists$  oracle machines  $\pi_A, \pi_B$ :
- ▶  $\forall f \in \mathcal{C}$ :
- ▶  $\pi_A^f(x) \rightleftharpoons \pi_B^f(y)$  is a secure protocol for evaluating  $f(x, y)$

If protocol uses trusted setup, then **same** setup for all  $f \in \mathcal{C}$ !

FBB secure evaluation of  $\mathcal{C}$  is *trivial* if:

- ▶  $|\mathcal{C}| = 1$  (protocol could “know” code of  $f$ )
- ▶  $\mathcal{C}$  is exactly learnable via oracle queries (learn code of  $f$ , then proceed in non-black-box way)

# autoreducibility

# autoreducibility

*How much "structure" does a set/function  $L$  have?*



# autoreducibility

*How much “structure” does a set/function  $L$  have?*

## Basic Definition

$L$  is **autoreducible** if there exists efficient  $M$ :

1.  $M^L(x) = L(x)$
2.  $M$  doesn't simply query its oracle on  $x$

# autoreducibility examples

Discrete log problem in  $\langle g \rangle$  is autoreducible:

# autoreducibility examples

Discrete log problem in  $\langle g \rangle$  is autoreducible:

$d\log_g(x)$ : // find  $d$  such that  $g^d = x$

1. Choose  $a \leftarrow \mathbb{Z}_n$ , where  $n = \text{ord}(g)$ .
2. Output:  $d\log_g(x \cdot g^a) - a \pmod n$

# autoreducibility examples

Discrete log problem in  $\langle g \rangle$  is autoreducible:

$d\log_g(x)$ : // find  $d$  such that  $g^d = x$

1. Choose  $a \leftarrow \mathbb{Z}_n$ , where  $n = \text{ord}(g)$ .
2. Output:  $d\log_g(x \cdot g^a) - a \pmod n$

# autoreducibility examples

Discrete log problem in  $\langle g \rangle$  is **instance-hiding** autoreducible:

$d\log_g(x)$ : // find  $d$  such that  $g^d = x$

1. Choose  $a \leftarrow \mathbb{Z}_n$ , where  $n = \text{ord}(g)$ .
2. Output:  $d\log_g(x \cdot g^a) - a \pmod n$

“Instance-hiding” autoreducible [BeaverFeigenbaum90]

Oracle queries of  $M^l(x)$  distributed independent of  $x$ .

# semi-honest adversaries

# characterization

## Definition

A class  $\mathcal{C}$  is **2-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^{f,f}(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$

# characterization

## Definition

A class  $\mathcal{C}$  is **2-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^{f,f}(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$
2.  $M$ 's queries to left oracle "don't depend on"  $y$
3.  $M$ 's queries to right oracle "don't depend on"  $x$



# characterization

## Definition

A class  $\mathcal{C}$  is **2-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^{f,f}(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$
2.  $M$ 's queries to left oracle "don't depend on"  $y$
3.  $M$ 's queries to right oracle "don't depend on"  $x$

Discussion:

- ▶ Same  $M$  must work for every  $f \in \mathcal{C}$ .
- ▶ Distinction between  $x$  and  $y$ .

# characterization

## Definition

A class  $\mathcal{C}$  is **2-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^{f,f}(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$
2.  $M$ 's queries to left oracle "don't depend on"  $y$
3.  $M$ 's queries to right oracle "don't depend on"  $x$

Discussion:

- ▶ Same  $M$  must work for every  $f \in \mathcal{C}$ .
- ▶ Distinction between  $x$  and  $y$ .

## Theorem

FBB secure computation of  $\mathcal{C}$  is possible in  $\mathcal{F}_{\text{ot}}$ -hybrid (against semi-honest adversaries) *if and only if*  $\mathcal{C}$  is 2-hiding autoreducible

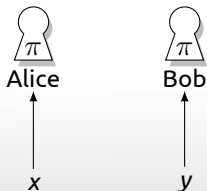
# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:



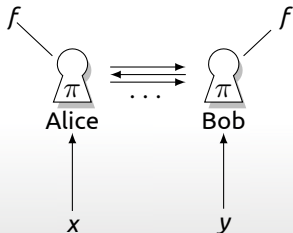
# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:



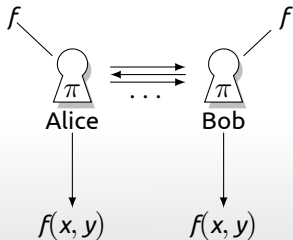
# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:



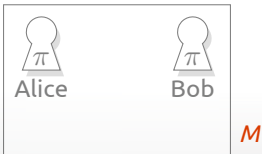
# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:



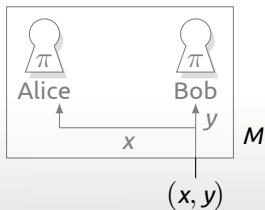
# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, **construct**  $M$  for autoreducibility:



# proof: fbb $\Rightarrow$ autoreducible

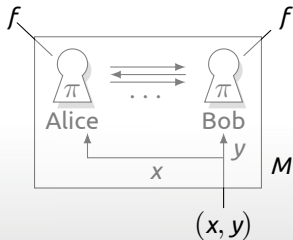
Given FBB protocol, construct  $M$  for autoreducibility:





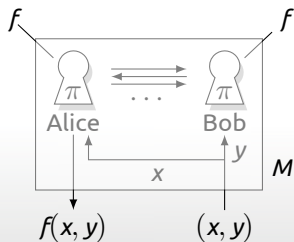
# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:



# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:

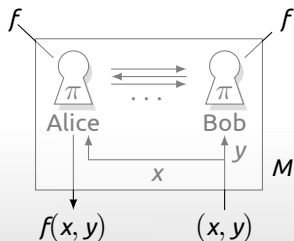


Correctness of protocol:

$\Rightarrow$  Output is  $f(x, y)$

# proof: fbb $\Rightarrow$ autoreducible

Given FBB protocol, construct  $M$  for autoreducibility:



Correctness of protocol:

$\Rightarrow$  Output is  $f(x, y)$

Security of protocol:

$\Rightarrow$  Alice's view (incl. oracle queries) "doesn't depend on"  $y$ .

$\Rightarrow$  Bob's view (incl. oracle queries) "doesn't depend on"  $x$ .

# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:

$M$

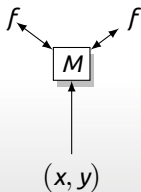
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



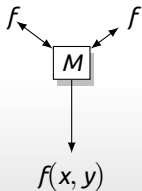
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



# proof: autoreducible $\Rightarrow$ fbb

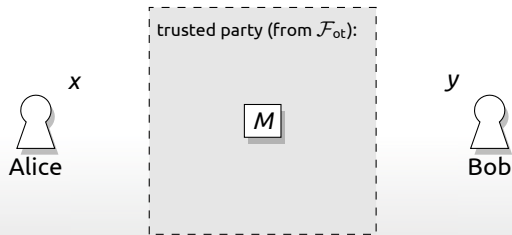
Given  $M$  from autoreducibility, **construct FBB protocol:**





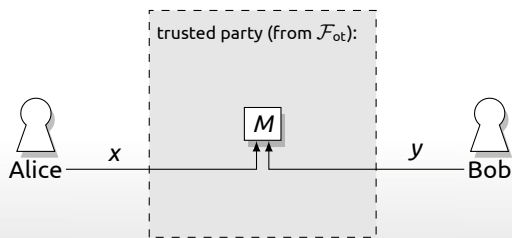
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



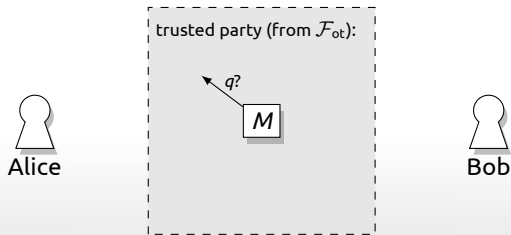
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



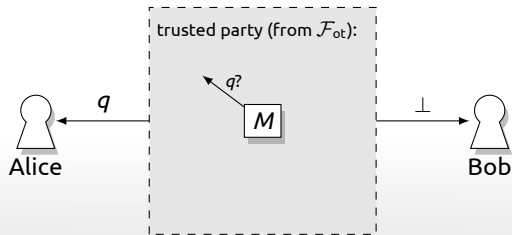
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



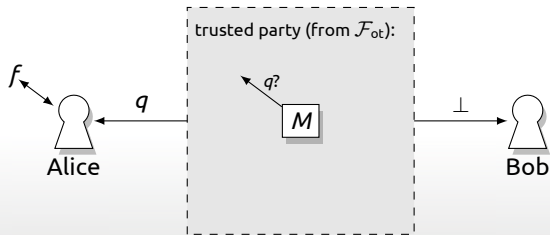
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



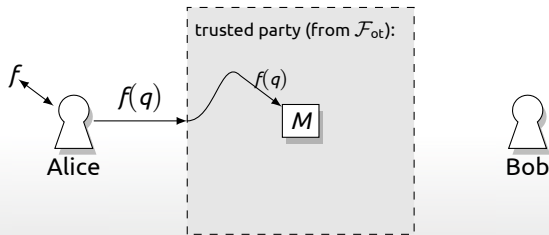
# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



# proof: autoreducible $\Rightarrow$ fbb

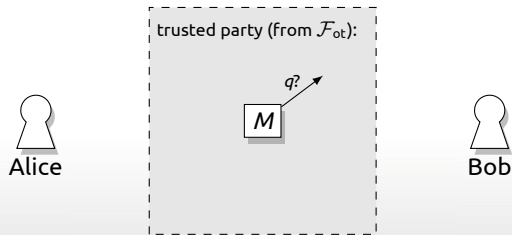
Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.

# proof: autoreducible $\Rightarrow$ fbb

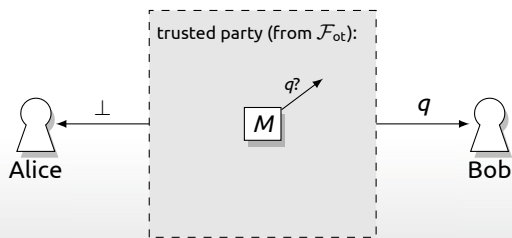
Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.

# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:

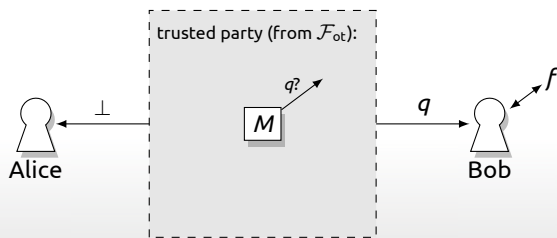


- ▶ Entire protocol treats  $f$  as black-box.



# proof: autoreducible $\Rightarrow$ fbb

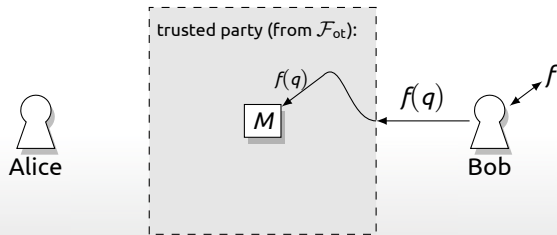
Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.

# proof: autoreducible $\Rightarrow$ fbb

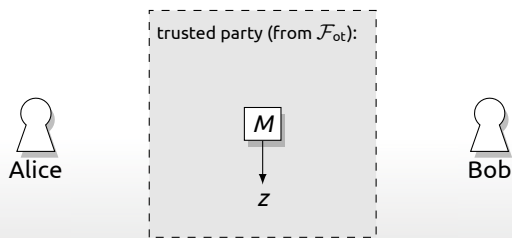
Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.

# proof: autoreducible $\Rightarrow$ fbb

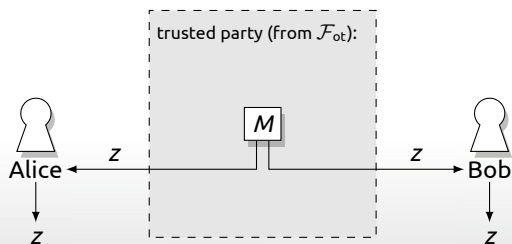
Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.

# proof: autoreducible $\Rightarrow$ fbb

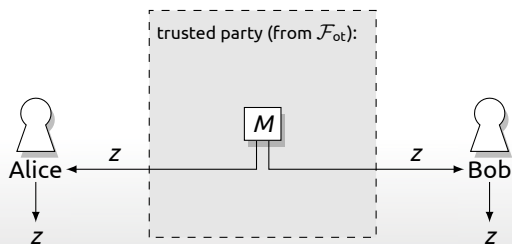
Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.
- ▶ Protocol output is correct (when protocol is followed!)

# proof: autoreducible $\Rightarrow$ fbb

Given  $M$  from autoreducibility, construct FBB protocol:



- ▶ Entire protocol treats  $f$  as black-box.
- ▶ Protocol output is correct (when protocol is followed!)
- ▶ Alice sees only output &  $M$ 's left oracle queries.
  - ▶ These "don't depend on" Bob's input  $y$ .
- ▶ Bob's sees only output &  $M$ 's right oracle queries.
  - ▶ These "don't depend on" Alice's input  $x$ .

# using the characterization:

## Positive example

There is a class  $\mathcal{C}$  that is 2-hiding autoreducible, but *not learnable* via oracle queries.

⇒ *Non-trivial* FBB secure computation!

☹ Class  $\mathcal{C}$  is not especially interesting.

# using the characterization:

## Positive example

There is a class  $\mathcal{C}$  that is 2-hiding autoreducible, but *not learnable* via oracle queries.

⇒ *Non-trivial* FBB secure computation!

☹ Class  $\mathcal{C}$  is not especially interesting.

## Negative example

Class of all PRFs is **not** 2-hiding autoreducible.

# using the characterization:

## Positive example

There is a class  $\mathcal{C}$  that is 2-hiding autoreducible, but *not learnable* via oracle queries.

⇒ *Non-trivial* FBB secure computation!

☹ Class  $\mathcal{C}$  is not especially interesting.

## Negative example

Class of all PRFs is **not** 2-hiding autoreducible.

⇒ Can't securely evaluate PRFs in FBB way (Alice holds seed, Bob holds input)

... even against semi-honest adversaries.

... even with arbitrarily powerful trusted setup



# malicious adversaries

# malicious adversaries

## Definition

A class  $\mathcal{C}$  is **1-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^f(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$
2.  $M$ 's queries to oracle "don't depend on"  $(x, y)$

Compare to "instance hiding" [BeaverFeigenbaum90]

# malicious adversaries

## Definition

A class  $\mathcal{C}$  is **1-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^f(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$
2.  $M$ 's queries to oracle "don't depend on"  $(x, y)$

Compare to "instance hiding" [BeaverFeigenbaum90]

## Theorem

If  $\mathcal{C}$  is 1-hiding autoreducible, then FBB secure computation of  $\mathcal{C}$  is possible against **malicious** adversaries.

# malicious adversaries

## Definition

A class  $\mathcal{C}$  is **1-hiding autoreducible** if there exists efficient  $M$ :

1.  $M^f(x, y) = f(x, y)$ , for all  $f \in \mathcal{C}$
2.  $M$ 's queries to oracle "don't depend on"  $(x, y)$

Compare to "instance hiding" [BeaverFeigenbaum90]

## Theorem

If  $\mathcal{C}$  is 1-hiding autoreducible, then FBB secure computation of  $\mathcal{C}$  is possible against **malicious** adversaries.

Proof sketch:

- ▶ Securely simulate  $M$
- ▶ Send its oracle queries to *both* parties
- ▶ Securely check for agreement of oracle responses

# wrap-up...

Also in the paper:

- ▶ Definition of FBB for more than just function evaluation
- ▶ Impossibility of ZK for membership in  $\text{im}(f)$ , for  $f \in \text{OWF}$

# wrap-up...

Also in the paper:

- ▶ Definition of FBB for more than just function evaluation
- ▶ Impossibility of ZK for membership in  $\text{im}(f)$ , for  $f$  OWF

Summary:

- ▶ Definitions for MPC protocol that has “black-box usage of functionality”
- ▶ Characterizations based on autoreducibility
- ▶ It is possible to “evaluate  $f$  without knowing the code of  $f$ ”
- ▶ ... but definitely not in general.

*The End*