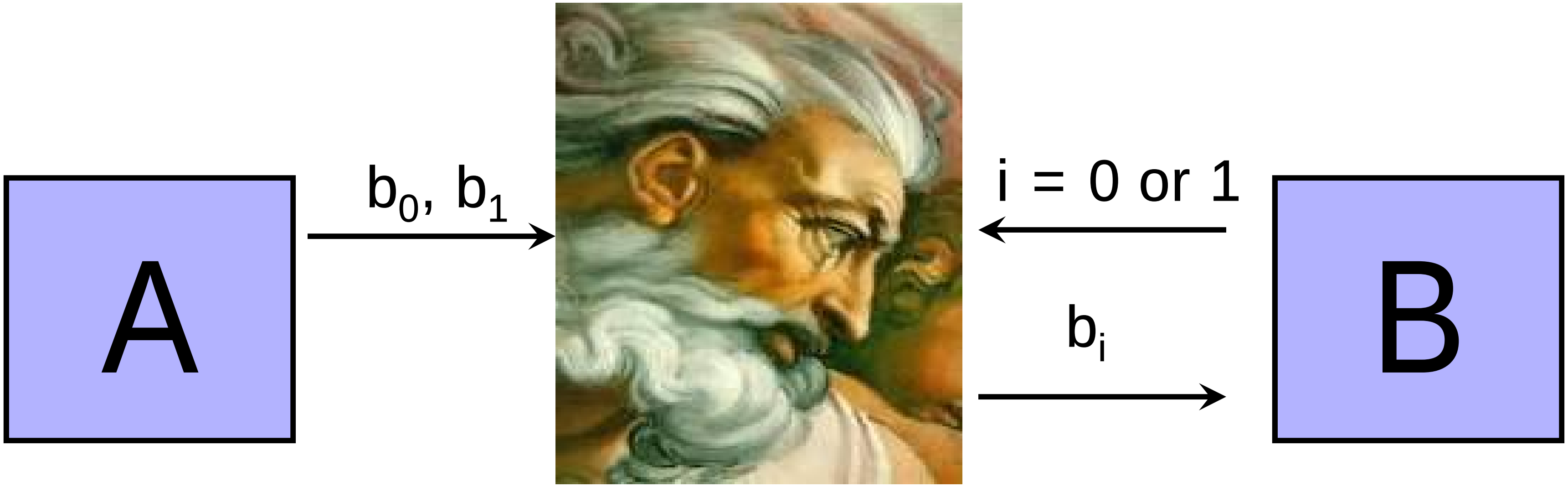
CS 380S

Oblivious Transfer and Secure Multi-Party Computation With Malicious Parties

Vitaly Shmatikov

Reminder: Oblivious Transfer



A

b_0, b_1

$i = 0$ or 1

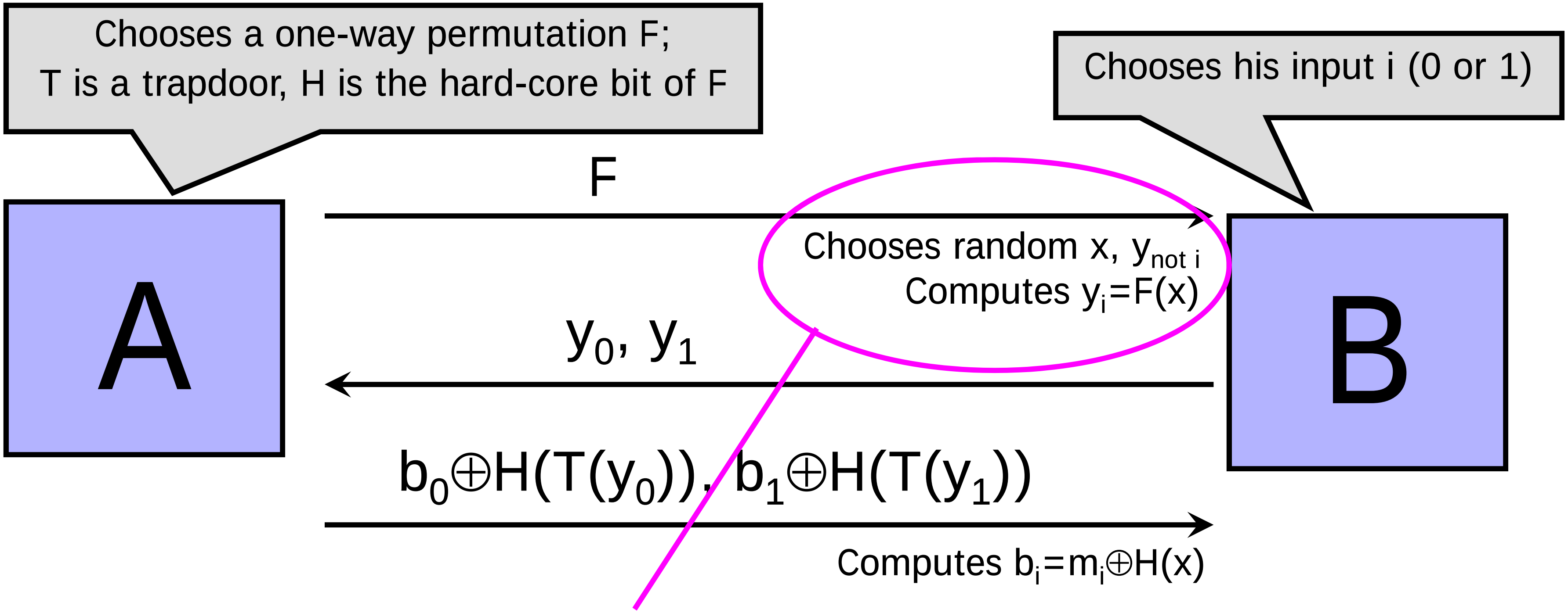
b_i

B

- A inputs two bits, B inputs the index of one of A's bits
- B learns his chosen bit, A learns nothing
  - A does not learn which bit B has chosen
  - B does not learn the value of the bit that he did not choose
- Generalizes to bitstrings, M instead of 2, etc.

Reminder: Semi-Honest OT Protocol

- Assume the existence of some family of one-way trapdoor permutations



Chooses a one-way permutation F;
T is a trapdoor, H is the hard-core bit of F

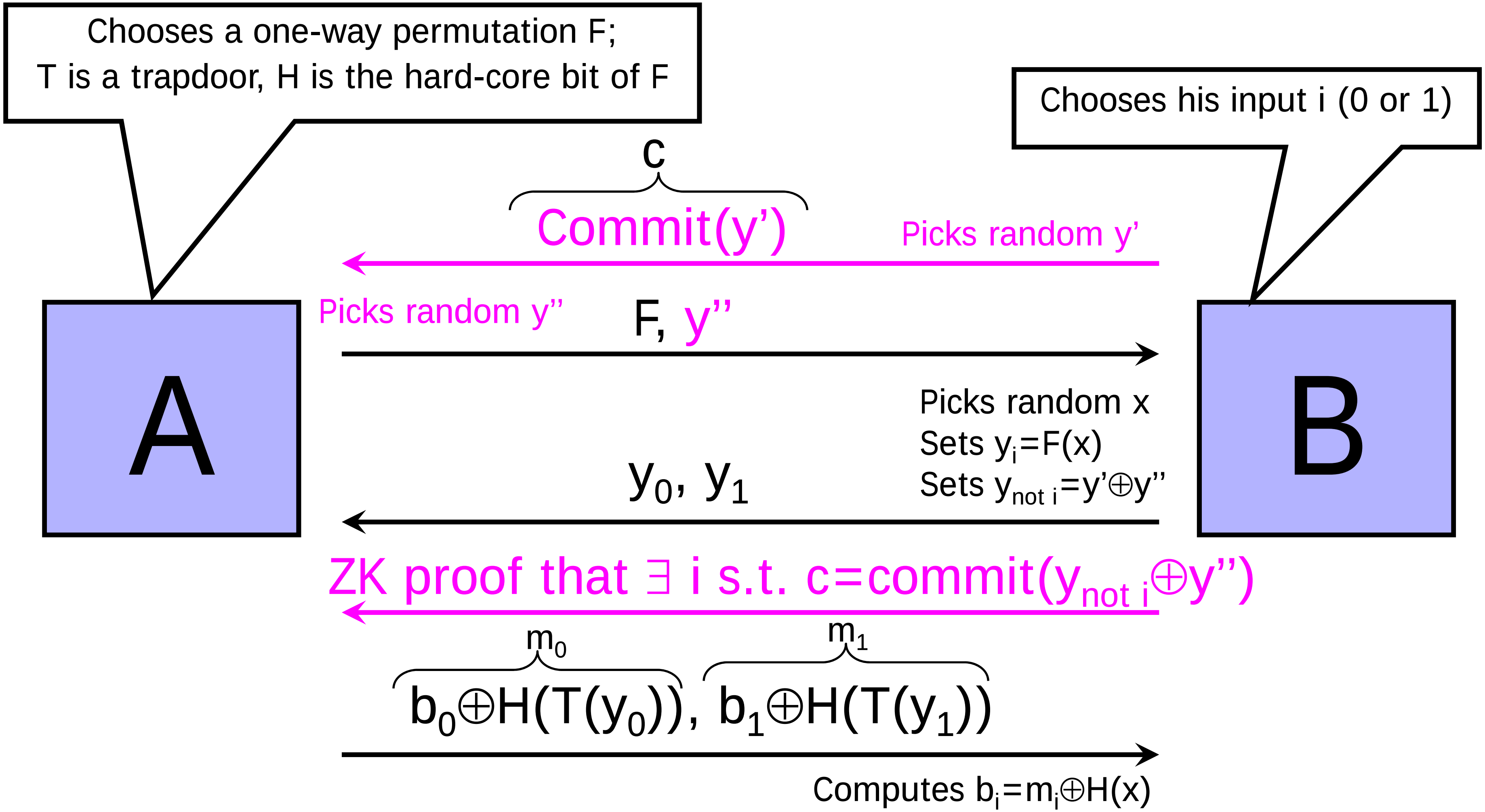
Chooses his input i (0 or 1)

Chooses random x, $y_{\text{not } i}$
Computes $y_i=F(x)$

Computes $b_i=m_i \oplus H(x)$

How do we force malicious B to follow the protocol and pick $y_{\text{not } i}$ randomly?

OT With Malicious Parties (Attempt)



Chooses a one-way permutation F;
T is a trapdoor, H is the hard-core bit of F

Chooses his input i (0 or 1)

$\underbrace{\text{Commit}(y')}_{}$ with label c

Picks random y'

Picks random y''

F, y''

Picks random x
Sets $y_i=F(x)$
Sets $y_{\text{not } i}=y'\oplus y''$

y_0, y_1

ZK proof that $\exists$ i s.t. $c=\text{commit}(y_{\text{not } i}\oplus y'')$

$\overbrace{b_0\oplus H(T(y_0))}^{m_0}, \overbrace{b_1\oplus H(T(y_1))}^{m_1}$

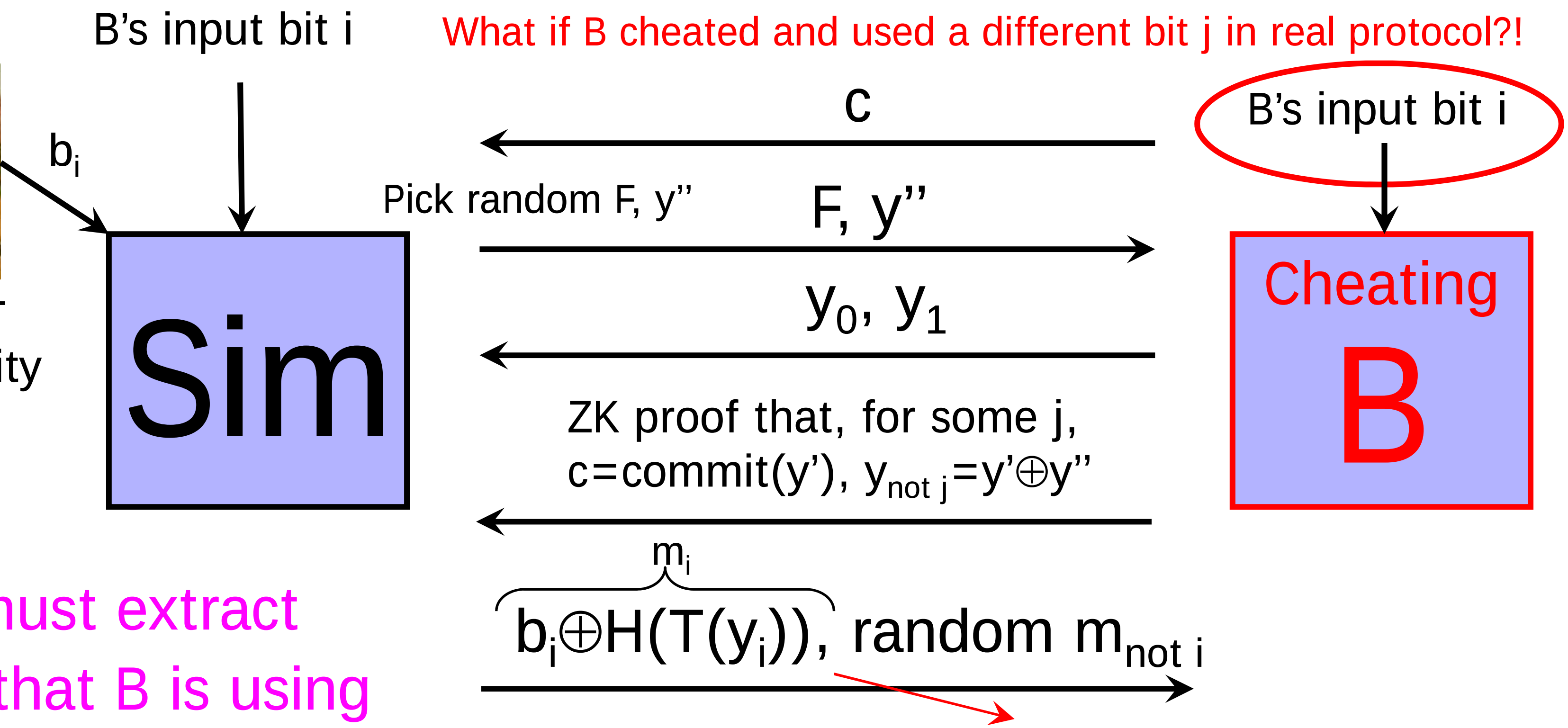
Computes $b_i=m_i\oplus H(x)$

Proving Security for Chooser

- Chooser in this protocol gives out much more information than in the original protocol
  - Commitment to a random value
  - ZK proof that he used this value in computing $y_{\text{not } i}$
- To prove security for Chooser, construct a simulator for Sender's view (details omitted)
  - Main idea: distribution of $\{c,y_0,y_1\}$ is independent of the bit *i* indicating Chooser's choice
  - Intuition: commitment c hides all information about y'
  - Intuition: Chooser's proof is zero-knowledge, thus there exists a simulator for Sender's view of the proof

Proof of Sender Security (Attempt)

- Simulating malicious Chooser's view of protocol



Ideal OT functionality

b_i

B's input bit i

Sim

What if B cheated and used a different bit j in real protocol?!

c

B's input bit i

Pick random F, y''

F, y''

y_0, y_1

Cheating B

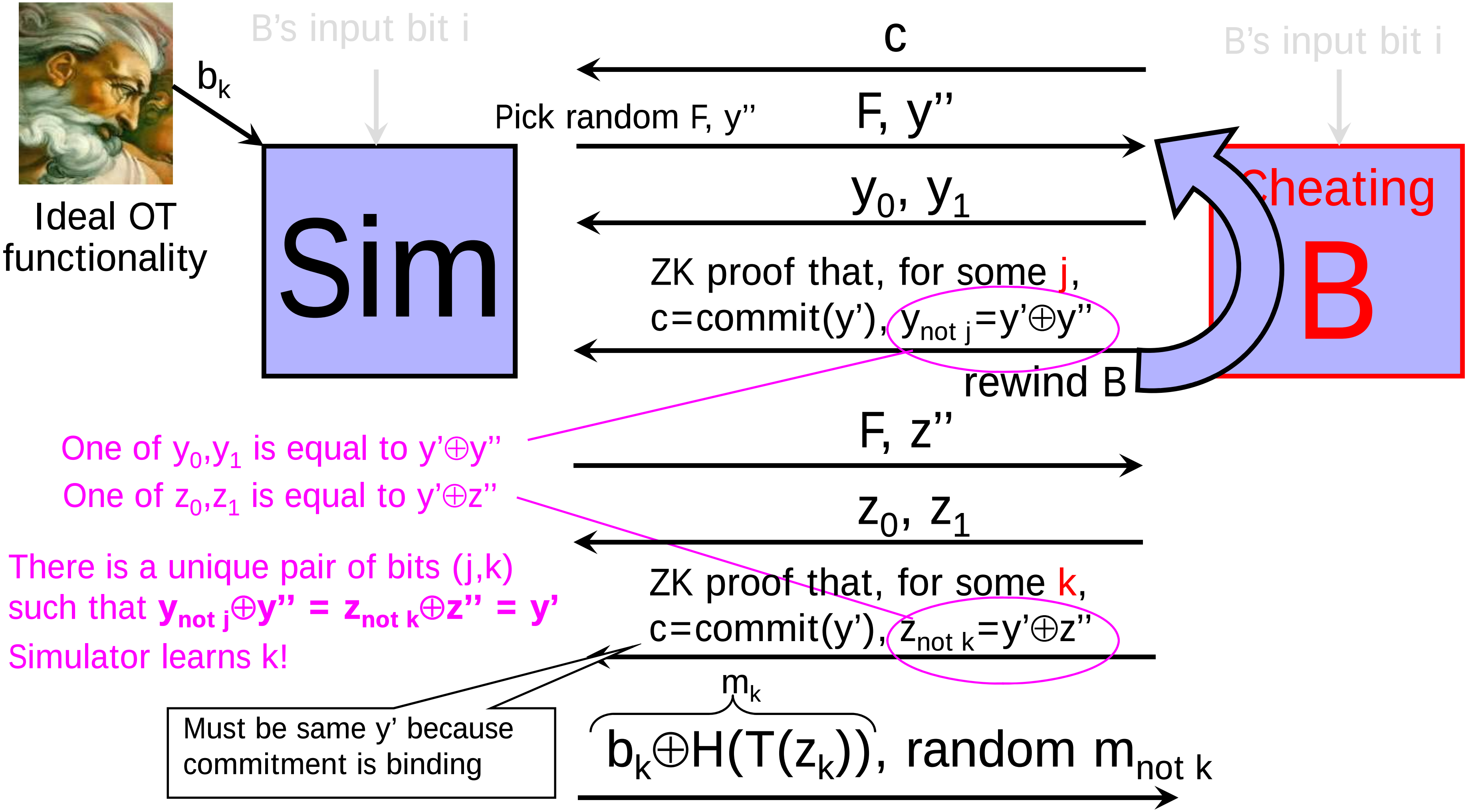
ZK proof that, for some j,
$c=\mathrm{commit}(y'), y_{\text{not } j}=y' \oplus y''$

$\overbrace{b_i \oplus H(T(y_i))}^{m_i}$, random $m_{\text{not } i}$

B will detect the difference between real protocol and simulation if $i \neq j$

Sim must extract bit / that B is using as his real choice

Extracting Chooser's Bit (Sketch!)



Ideal OT functionality

b_k

B's input bit i

Sim

Pick random F, y''

c

F, y''

y_0, y_1

ZK proof that, for some j, c=commit(y'), $y_{\text{not } j} = y' \oplus y''$

rewind B

F, z''

z_0, z_1

ZK proof that, for some k, c=commit(y'), $z_{\text{not } k} = y' \oplus z''$

$\overset{m_k}{\overbrace{b_k \oplus H(T(z_k))}}$, random $m_{\text{not } k}$

B's input bit i

Cheating B

One of y_0,y_1 is equal to $y' \oplus y''$

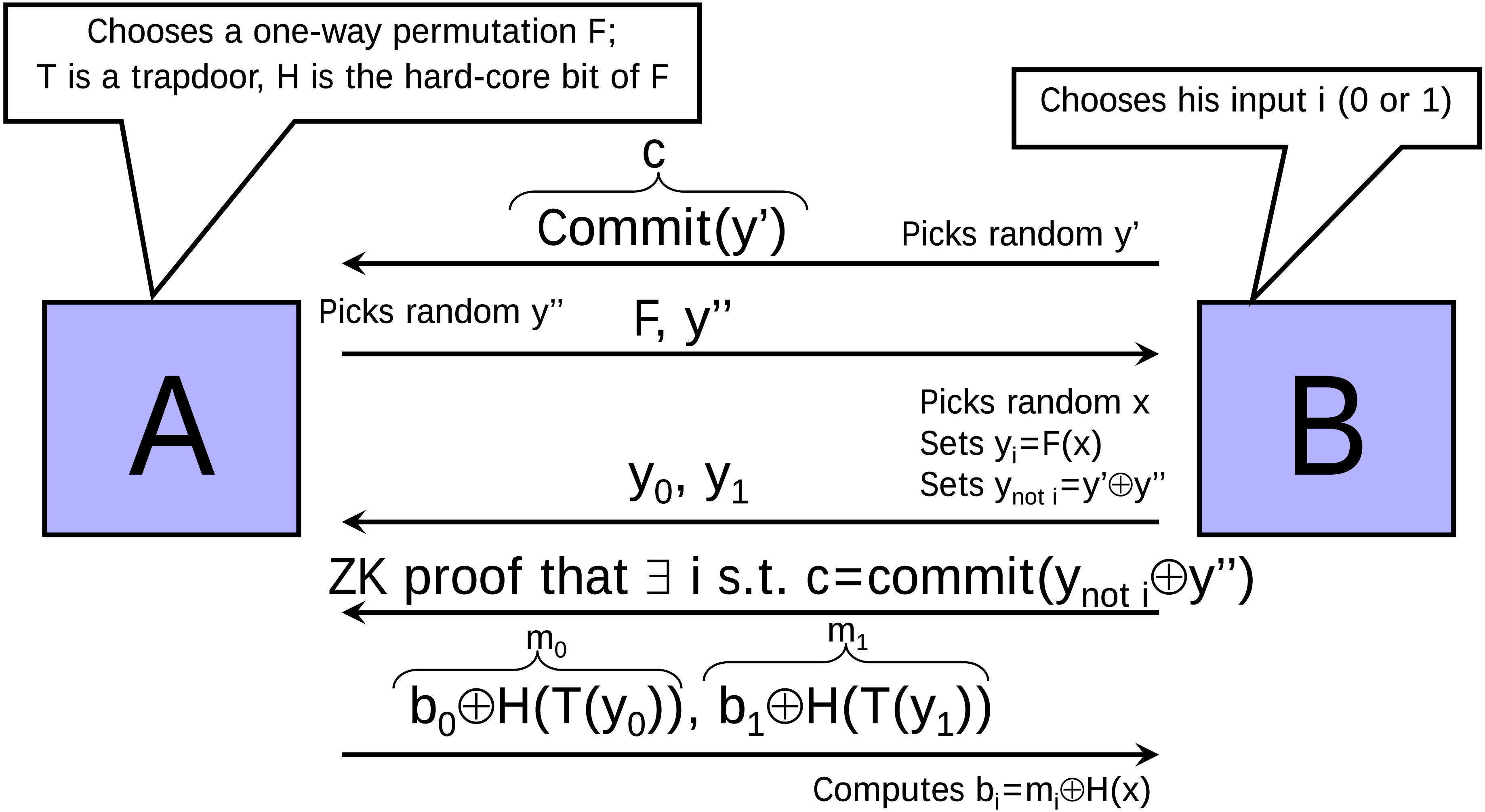
One of z_0,z_1 is equal to $y' \oplus z''$

There is a unique pair of bits (j,k) such that $\mathbf{y_{not\ j} \oplus y'' = z_{not\ k} \oplus z'' = y'}$

Simulator learns k!

Must be same y' because commitment is binding

So, Is This Protocol Secure?



Chooses a one-way permutation F;
T is a trapdoor, H is the hard-core bit of F

Chooses his input i (0 or 1)

A

B

Picks random y'

$\overbrace{\text{Commit}(y')}^{c}$

Picks random y''

F, y''

Picks random x
Sets $y_i=F(x)$
Sets $y_{\text{not } i}=y'\oplus y''$

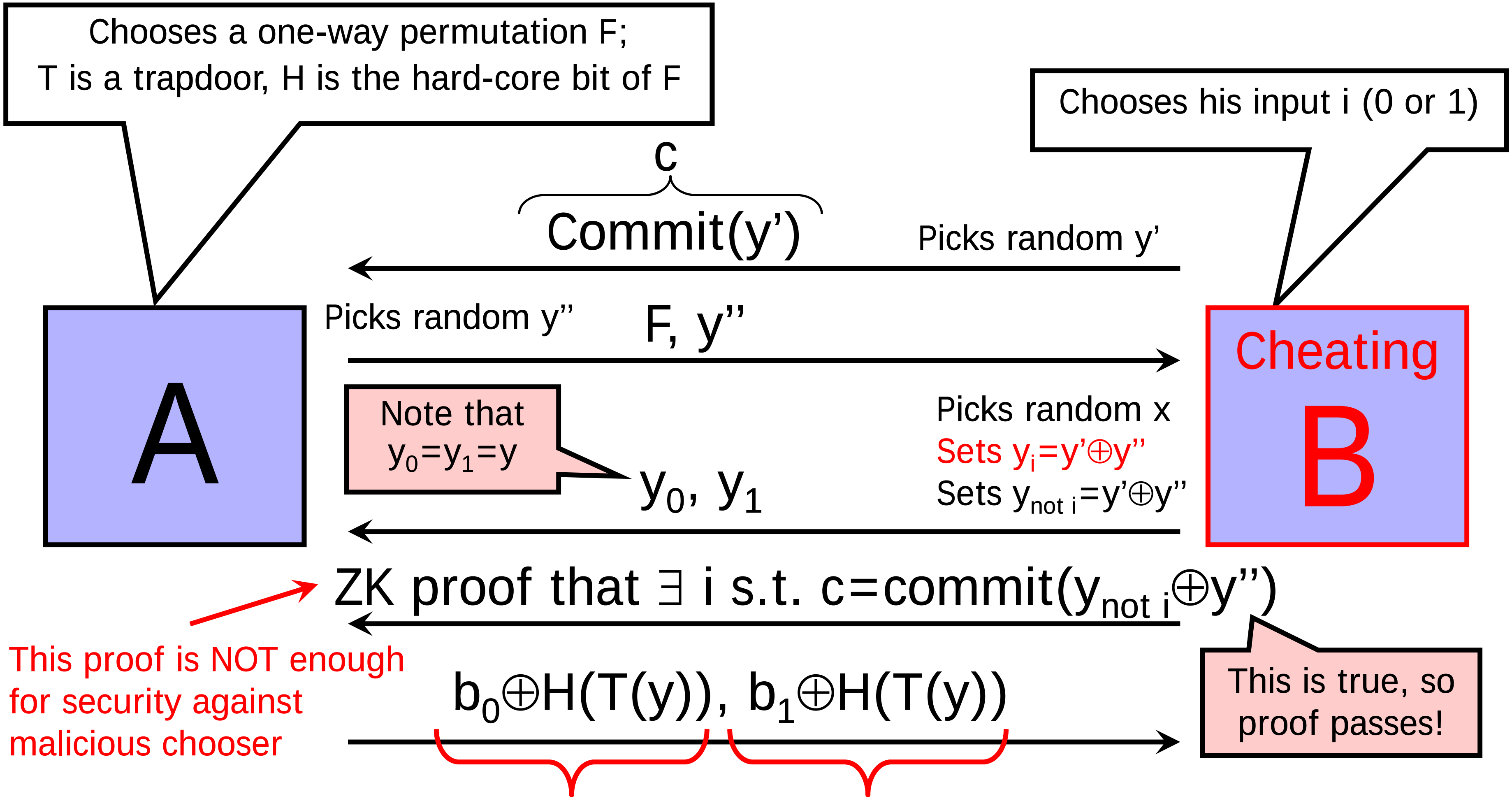
y_0, y_1

ZK proof that $\exists$ i s.t. $c=\text{commit}(y_{\text{not } i}\oplus y'')$

$\overbrace{b_0\oplus H(T(y_0))}^{m_0}$, $\overbrace{b_1\oplus H(T(y_1))}^{m_1}$

Computes $b_i=m_i\oplus H(x)$

Oops!



Chooses a one-way permutation F;
T is a trapdoor, H is the hard-core bit of F

Chooses his input i (0 or 1)

A

Cheating B

$\overbrace{\text{Commit}(y')}^{c}$

Picks random y'

Picks random y''

F, y''

Note that $y_0=y_1=y$

y_0, y_1

Picks random x

Sets $y_i=y'\oplus y''$

Sets $y_{not\ i}=y'\oplus y''$

ZK proof that ∃ i s.t. $c=commit(y_{not\ i}\oplus y'')$

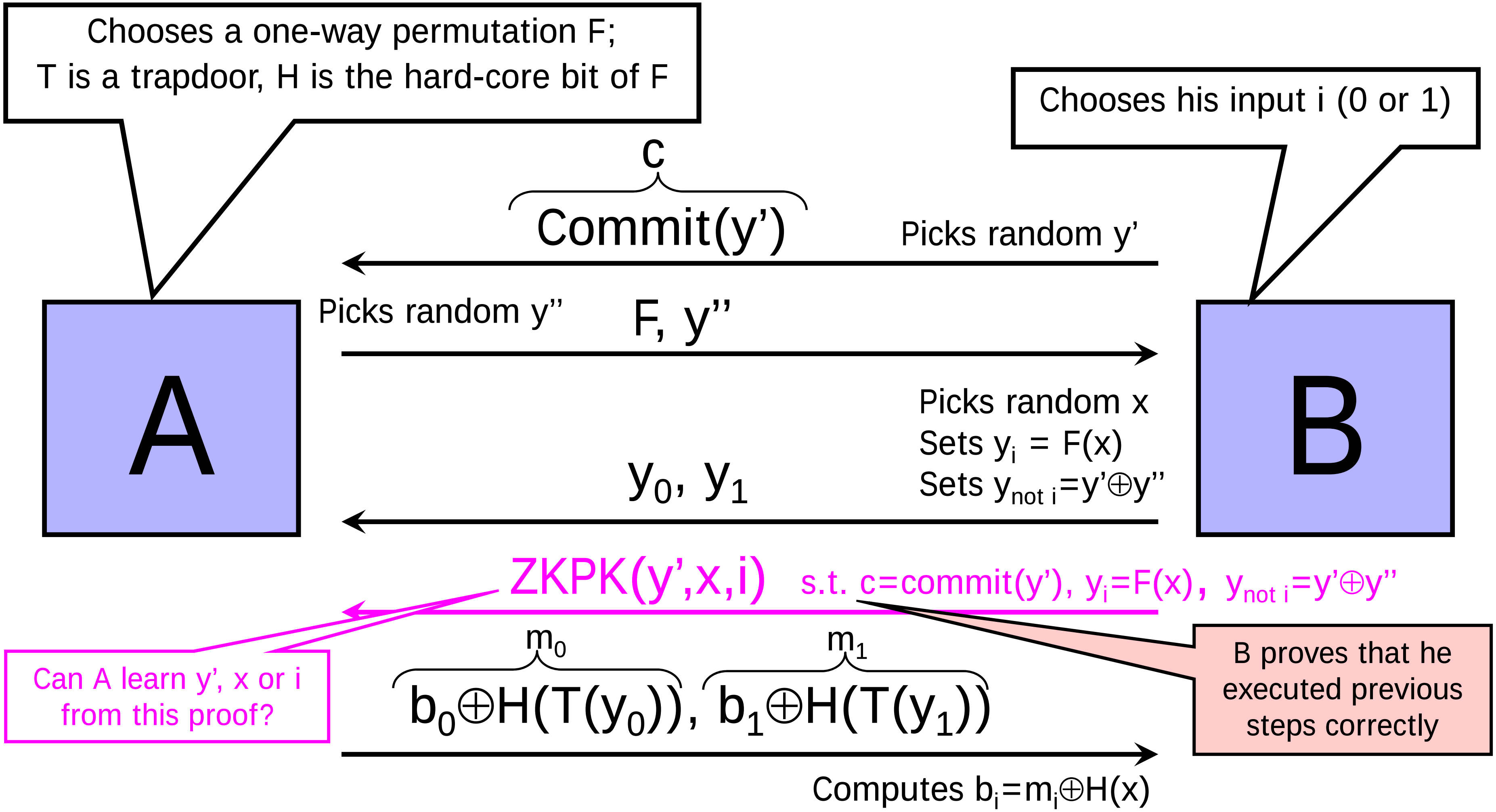
This is true, so proof passes!

This proof is NOT enough for security against malicious chooser

$\underbrace{b_0\oplus H(T(y))}, \underbrace{b_1\oplus H(T(y))}$

If these values are the same, B learns that $b_0 = b_1$
Would he be able to learn this in the ideal world?

OT Protocol with Malicious Parties



Chooses a one-way permutation F;
T is a trapdoor, H is the hard-core bit of F

Chooses his input i (0 or 1)

A

B

$\overbrace{\text{Commit}(y')}^{c}$

Picks random y'

Picks random y''

F, y''

Picks random x
Sets $y_i = F(x)$
Sets $y_{\text{not } i}=y'\oplus y''$

y_0, y_1

ZKPK(y',x,i) s.t. c=commit(y'), $y_i=F(x)$, $y_{\text{not } i}=y'\oplus y''$

Can A learn y', x or i from this proof?

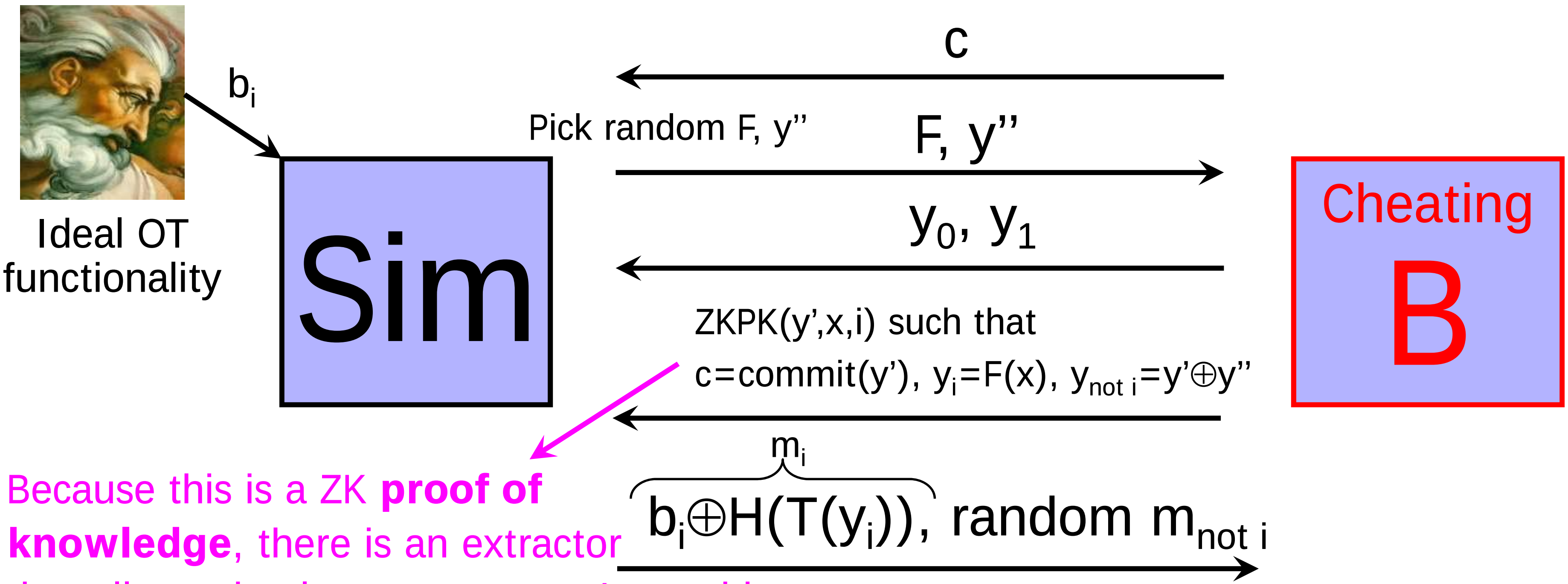
B proves that he executed previous steps correctly

$\overbrace{b_0\oplus H(T(y_0))}^{m_0}, \overbrace{b_1\oplus H(T(y_1))}^{m_1}$

Computes $b_i=m_i\oplus H(x)$

Proving Sender Security

- Simulating malicious Chooser’s view of protocol



Ideal OT functionality

b_i

Sim

Cheating B

c

Pick random F, y’’

F, y''

y_0, y_1

ZKPK(y’,x,i) such that

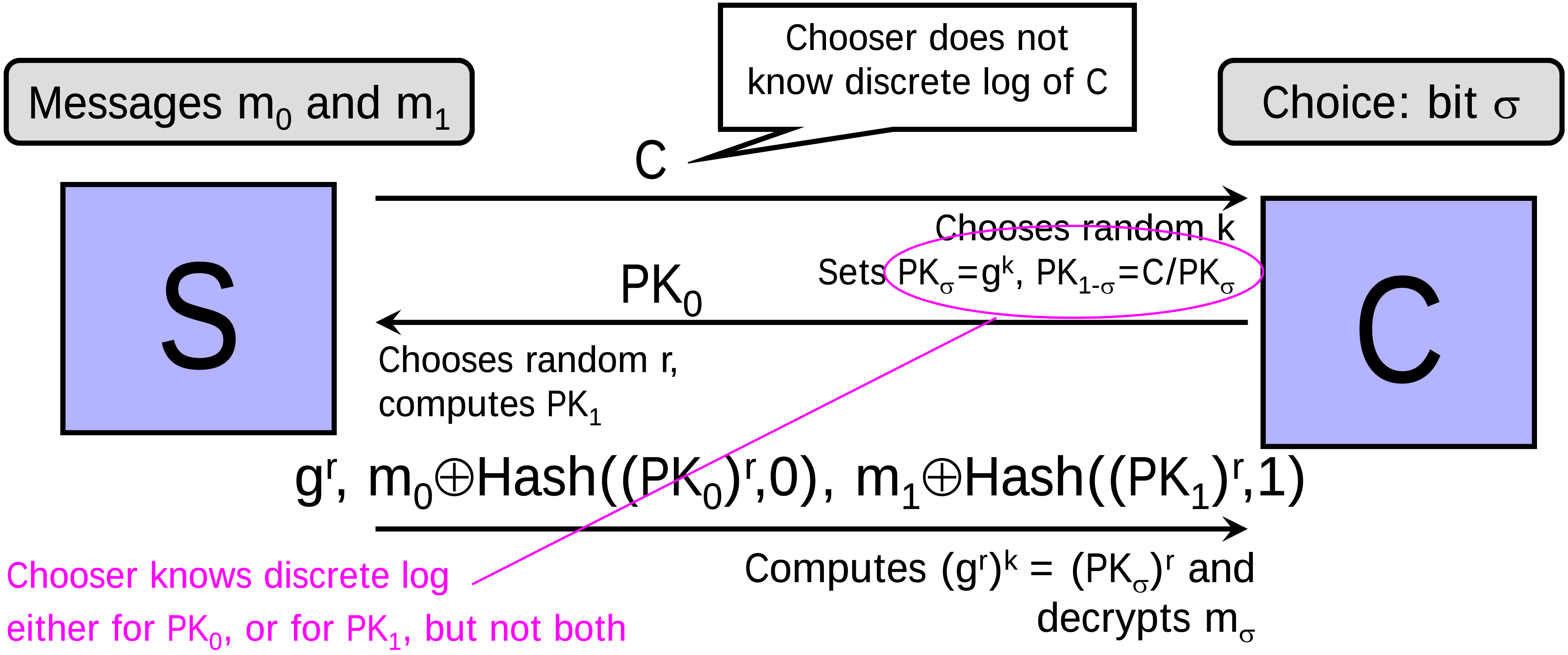
c=commit(y’), $y_i=F(x)$, $y_{\text{not } i}=y'\oplus y''$

$\overbrace{b_i \oplus H(T(y_i))}^{m_i}$, random $m_{\text{not } i}$

Because this is a ZK **proof of knowledge**, there is an extractor that allows simulator to extract y’, x and i (but ZKPK proof system must be secure against a malicious verifier – why?)

Naor-Pinkas Oblivious Transfer

Setting: order-q subgroup of Z*p, p is prime, q divides p-1
g is a generator group for which CDH assumption holds



Messages m_0 and m_1

Chooser does not know discrete log of C

Choice: bit σ

S

C

C

Chooses random k

Sets $PK_\sigma = g^k$, $PK_{1-\sigma} = C/PK_\sigma$

PK_0

Chooses random r, computes PK_1

$g^r, m_0 \oplus Hash((PK_0)^r,0), m_1 \oplus Hash((PK_1)^r,1)$

Computes $(g^r)^k = (PK_\sigma)^r$ and decrypts m_σ

Chooser knows discrete log either for PK_0, or for PK_1, but not both

Chooser does not know the discrete log of $PK_{1-\sigma}$, thus cannot distinguish between a random value g_z and $(PK_{1-\sigma})^r$ - **why?**

1-out-of-4 Oblivious Transfer

A

b_1,b_2,b_3,b_4

$i=1,2,3$ or 4

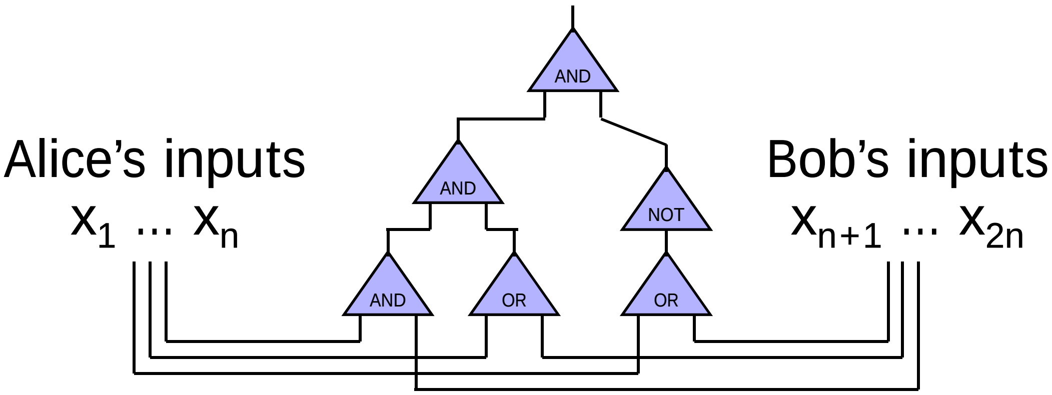
b_i

B

- Very similar to 1-out-of-2 oblivious transfer
- How to construct a 1-out-of-4 OT protocol given an 1-out-of-2 protocol?

Boolean Circuits

- Alice and Bob want to compute function f(a,b)
  - Assume for now Alice and Bob are semi-honest
- First, convert the function into a boolean circuit

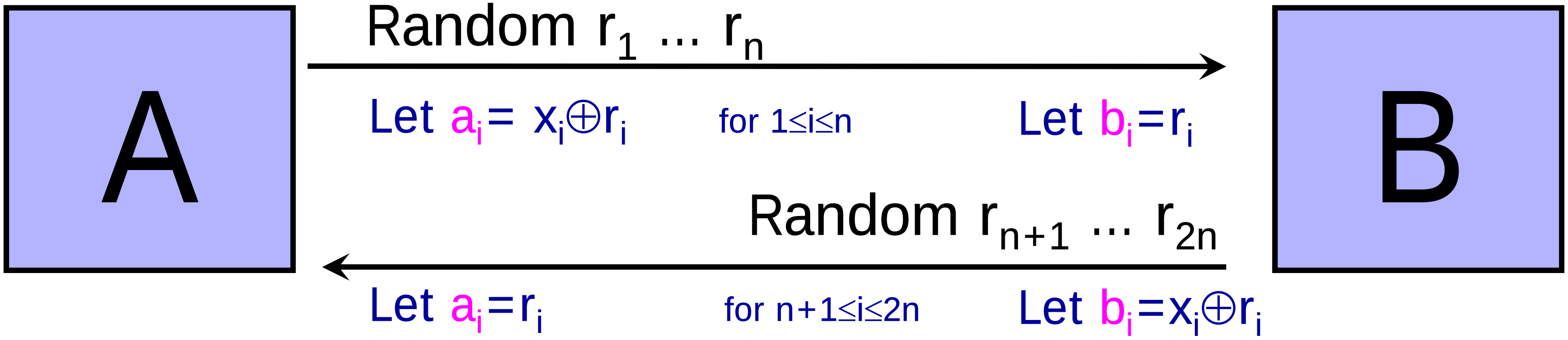


Alice's inputs
$x_1 \ldots x_n$

Bob's inputs
$x_{n+1} \ldots x_{2n}$

- Next, parties securely share their inputs

Input Sharing



Knows x_1 ... x_n

Knows x_{n+1} ... x_{2n}

Random r_1 ... r_n

Let $a_i = x_i \oplus r_i$ for $1 \le i \le n$ Let $b_i = r_i$

Random r_{n+1} ... r_{2n}

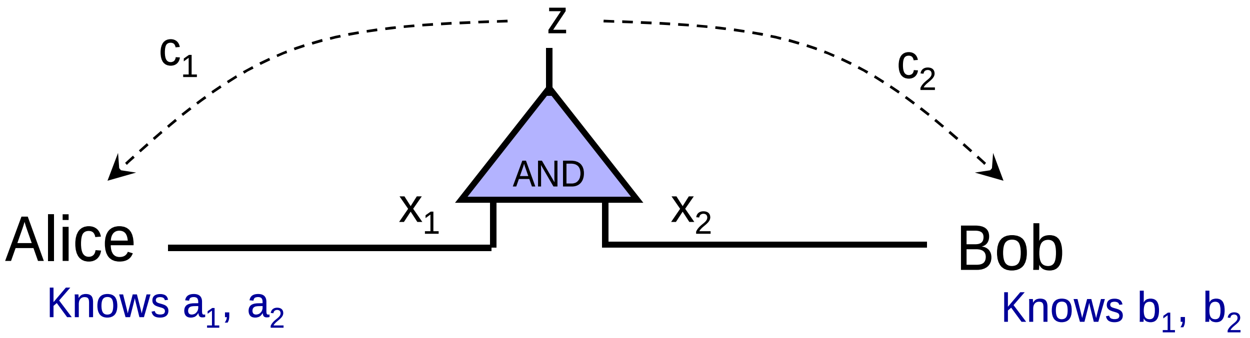
Let $a_i = r_i$ for $n+1 \le i \le 2n$ Let $b_i = x_i \oplus r_i$

◆ After this exchange, for all inputs x_i ...

$$x_i = a_i \oplus b_i$$

- Alice still doesn’t know Bob’s inputs, and vice versa
- This is information-theoretically secure

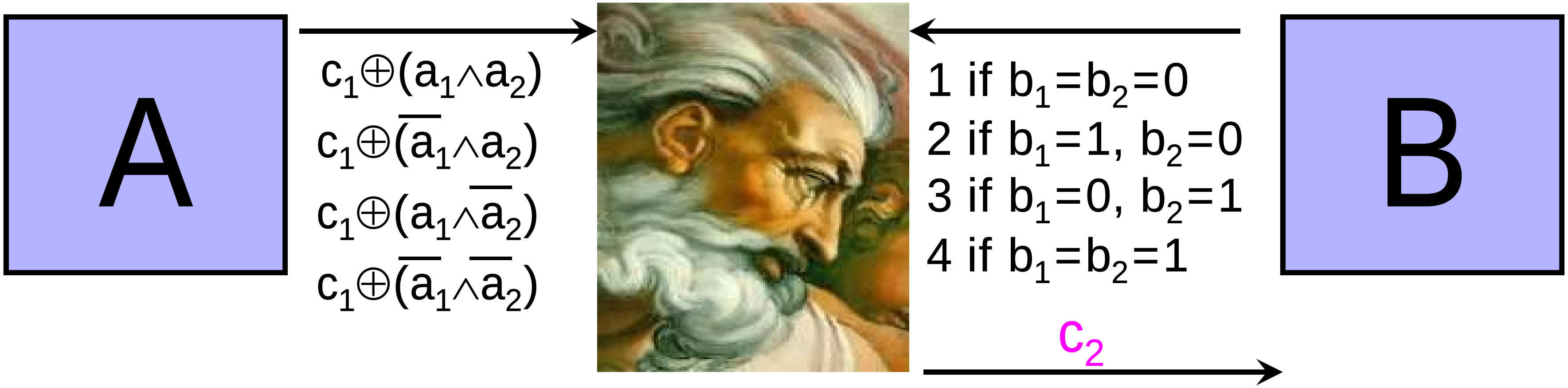
Evaluating an AND Gate



- Inputs x_1 and x_2 are shared between Alice and Bob
  - $x_1=a_1\oplus b_1$, $x_2=a_2\oplus b_2$
- At the end of the protocol, Alice learns bit c_1 and Bob learns bit c_2 such that…
  - Bits $c_{1,2}$ are "random"
  - $c_1\oplus c_2 = x_1\wedge x_2 = (a_1\oplus b_1)\wedge(a_2\oplus b_2)$
    - Output of the gate is shared between A and B just like the inputs

Use Oblivious Transfer

Pick random c_1



A

$c_1 \oplus (a_1 \wedge a_2)$

$c_1 \oplus (\overline{a_1} \wedge a_2)$

$c_1 \oplus (a_1 \wedge \overline{a_2})$

$c_1 \oplus (\overline{a_1} \wedge \overline{a_2})$

1 if $b_1 = b_2 = 0$

2 if $b_1 = 1$, $b_2 = 0$

3 if $b_1 = 0$, $b_2 = 1$

4 if $b_1 = b_2 = 1$

c_2

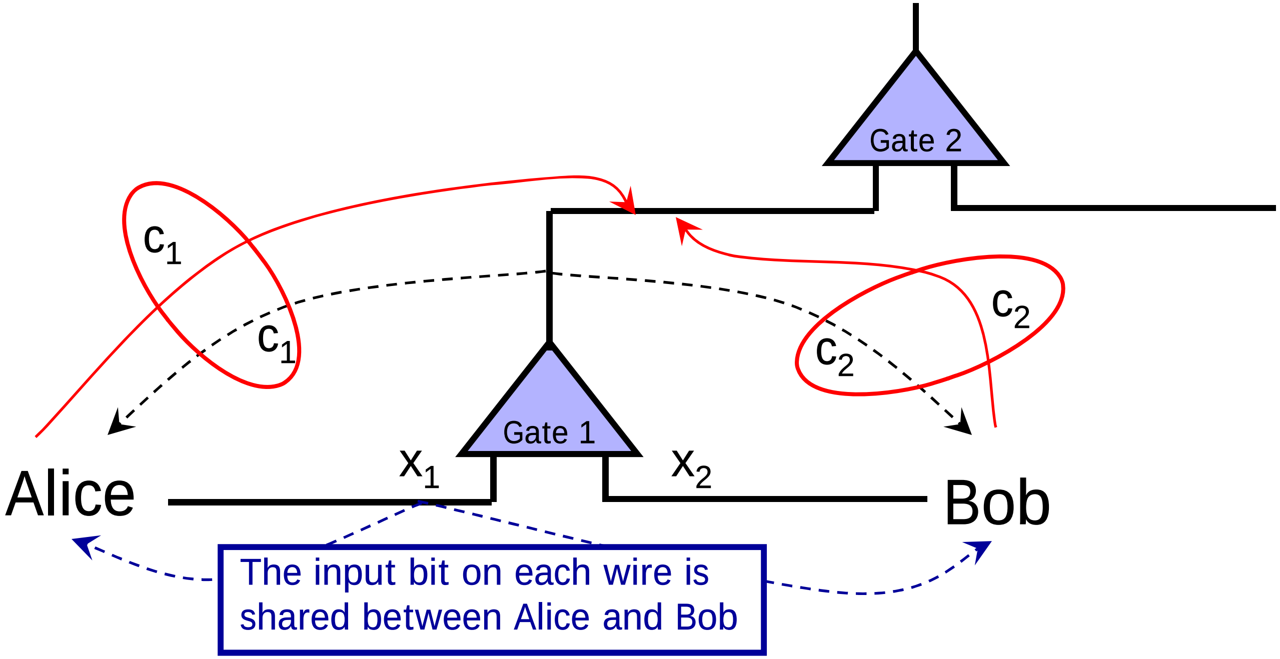
B

- In every case, $c_1 \oplus c_2 = (a_1 \oplus b_1) \wedge (a_2 \oplus b_2) = x_1 \wedge x_2$
- Can use similar tricks for other gate types
- Why do I give the ideal functionality only, not the actual protocol?

Is Secure OT Enough?

- We saw an oblivious transfer protocol which is secure even if parties are malicious
  - Chooser commits to his input and proves in zero knowledge that he performed his computations correctly
- Suppose each gate of the circuit is evaluated using an oblivious transfer protocol which is secure with malicious parties...
- Do we obtain a protocol for securely computing any function with malicious parties?

Oops!



The input bit on each wire is shared between Alice and Bob

- When output of Gate 1 is used as input into Gate 2, both parties must use the same pair of bits that was obtained from evaluating Gate 1 (why?)

Security with Malicious Parties

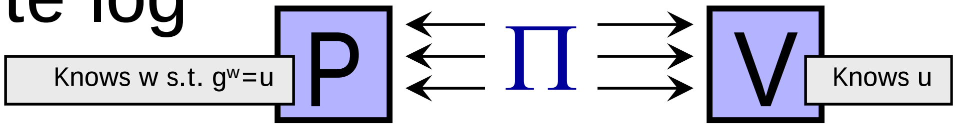
- Details omitted (this is less than a sketch!)
- Intuition: every party commits to its inputs and proves in ZK that it did its computation correctly
  - This includes proving that output of one OT protocol is consistent with input of another OT protocol
- Main advantage: secure building blocks compose into secure protocols
  - Each building block (commitment, OT, etc.) is indistinguishable from its ideal functionality (IF)
  - IFs can be composed without comprosiming security
  - Can build a secure protocol from secure primitives

Issues

- Parallel composition of zero-knowledge proofs does not work in general
  - One proof is zero-knowledge, but two proofs executed concurrently are no longer zero-knowledge
- How can cryptographic primitives be formulated as secure multi-party problems?
  - Many protocols use encryption, digital signatures, etc.
- Adaptive corruptions: adversary may corrupt an honest party in the middle of protocol execution
  - Proofs with “rewinding” don’t work any more (why?)

ZKPK of Discrete Log

- Suppose we have some ZK protocol Π for proving knowledge of discrete log



Knows w s.t. $g^w=u$

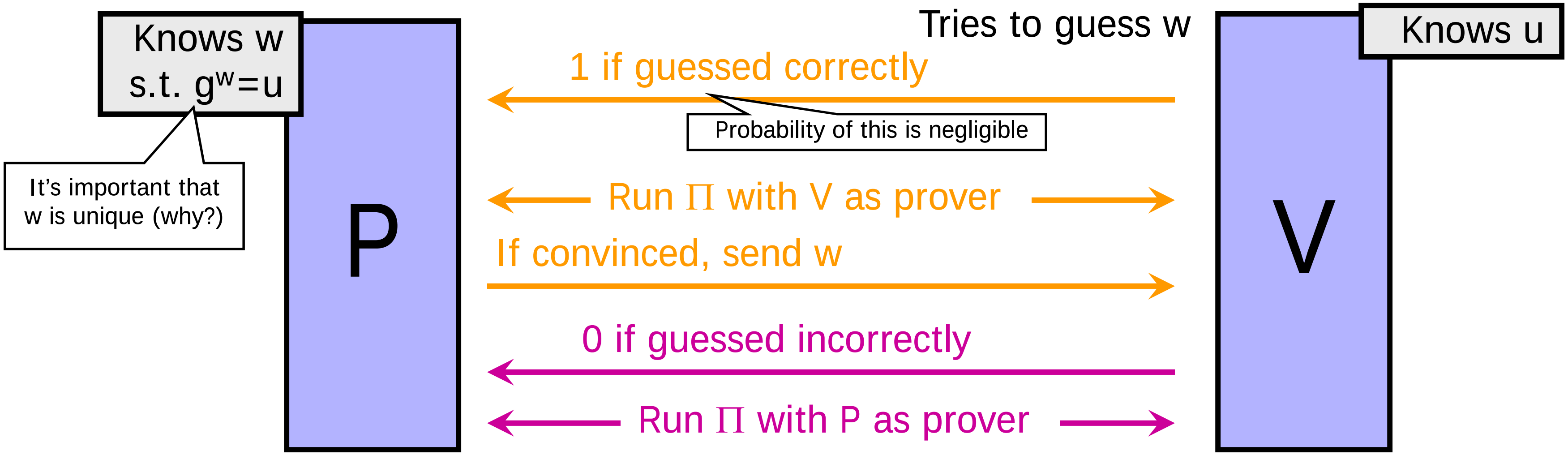
P

Π

V

Knows u

- Here is another protocol Π'



Knows w s.t. $g^w=u$

It's important that w is unique (why?)

P

Tries to guess w

1 if guessed correctly

Probability of this is negligible

Run Π with V as prover

If convinced, send w

0 if guessed incorrectly

Run Π with P as prover

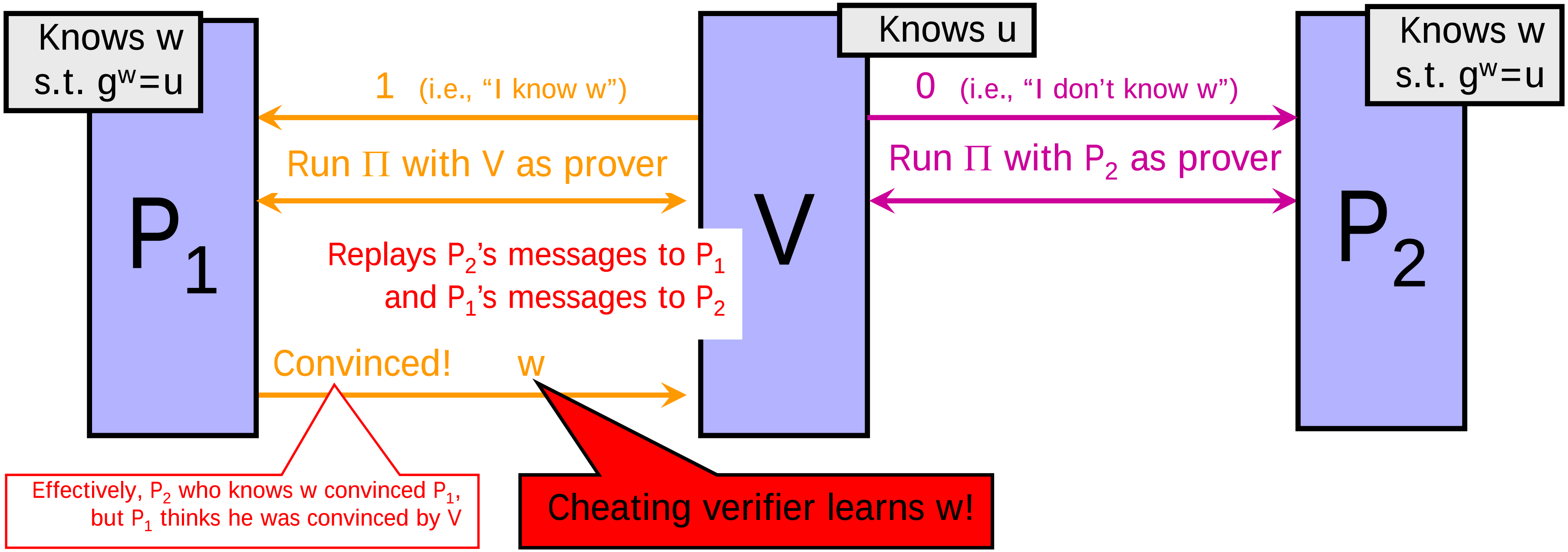
V

Knows u

Π' is a sound zero-knowledge protocol of discrete-log knowledge (why?)

Concurrent Composition

- V runs two instances of protocol Π’ in parallel



Knows w s.t. g^w=u

Knows u

Knows w s.t. g^w=u

P_1

V

P_2

1 (i.e., “I know w”)

Run Π with V as prover

0 (i.e., “I don’t know w”)

Run Π with P_2 as prover

Replays P_2’s messages to P_1 and P_1’s messages to P_2

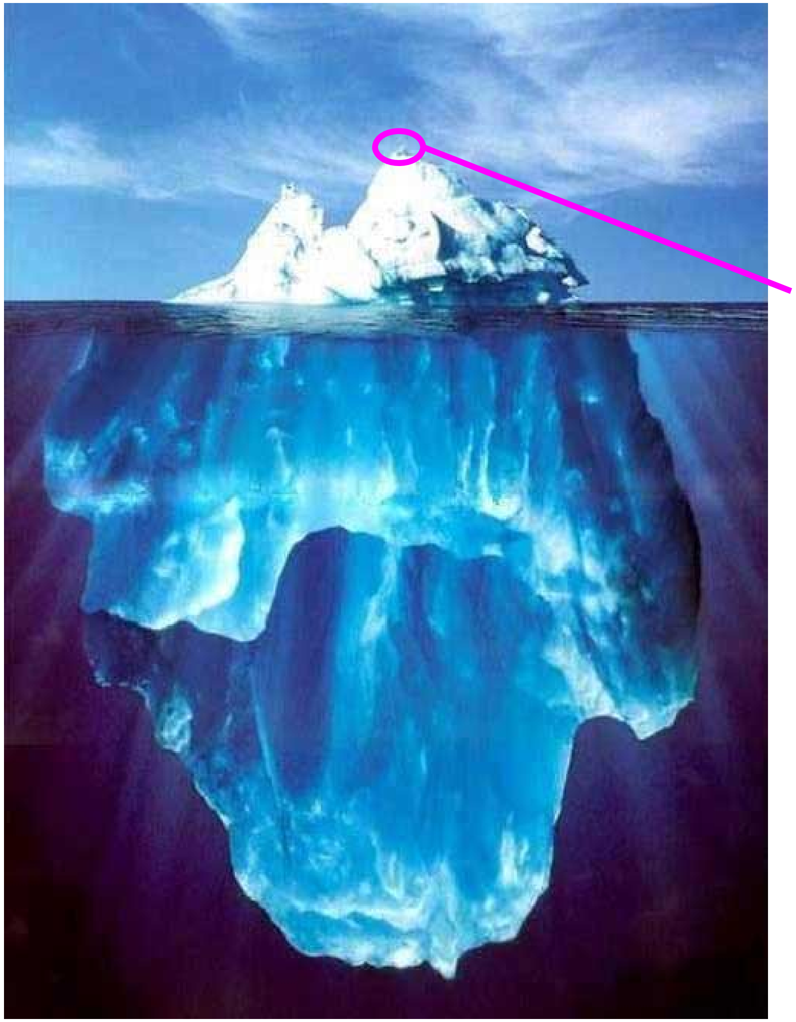
Convinced! w

Effectively, P_2 who knows w convinced P_1, but P_1 thinks he was convinced by V

Cheating verifier learns w!

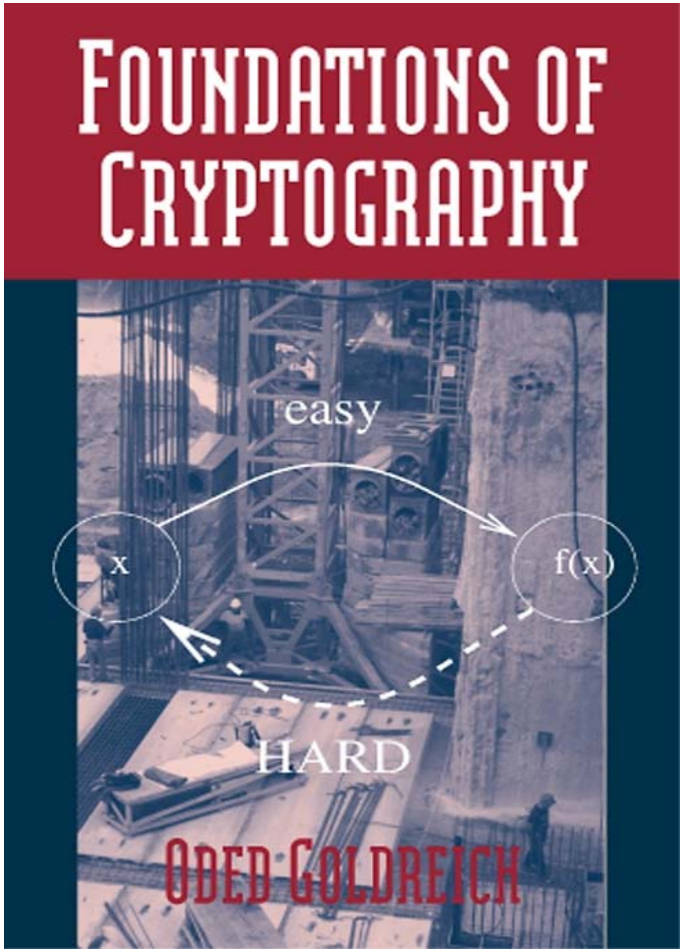
This protocol is clearly NOT zero-knowledge (why?)

SMC In This Course



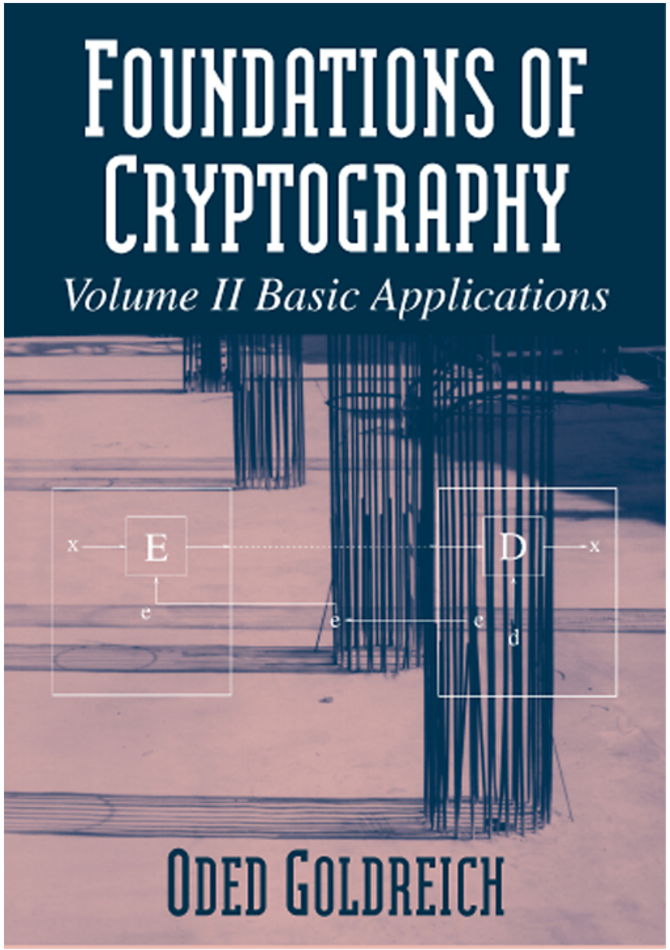
This is how much
we cover in this class

To Learn More



FOUNDATIONS OF CRYPTOGRAPHY

ODED GOLDREICH



FOUNDATIONS OF CRYPTOGRAPHY

Volume II Basic Applications

ODED GOLDREICH