Constant-Rate Oblivious Transfer from Noisy Channels

Yuval **I**shai
Eyal **K**ushilevitz
Rafail **O**strovsky
Manoj **P**rabhakaran
Amit **S**ahai
Jürg **W**ullschleger

Constant-Rate Oblivious Transfer from Noisy Channels

Yuval **I**shai
Eyal **K**ushilevitz
Rafail **O**strovsky
Manoj **P**rabhakaran
Amit **S**ahai
Jürg **W**ullschleger

Noisy Channel & Crypto

Noisy Channel & Crypto

From our point of view, an ideal communication line is a sterile, cryptographically uninteresting entity. Noise, on the other hand, breeds disorder, uncertainty, and confusion. Thus, it is the cryptographer's natural ally.

Claude Crépeau & Joe Kilian, 1988.

Noisy Channel & Crypto

Noisy Channel & Crypto

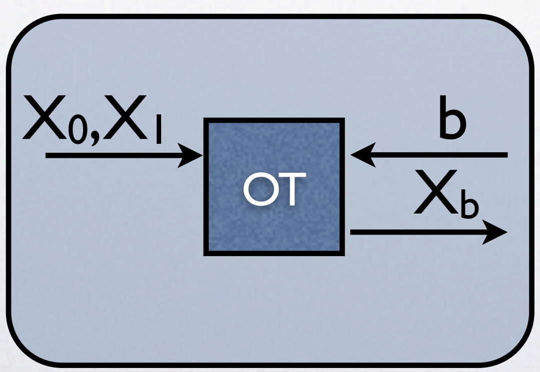
- Wyner's wire-tap channel: information-theoretically secret communication, without shared keys [W'75]

Noisy Channel & Crypto

- Wyner's wire-tap channel: information-theoretically secret communication, without shared keys [W'75]
- Oblivious Transfer from noisy channel [CK'88]

Noisy Channel & Crypto

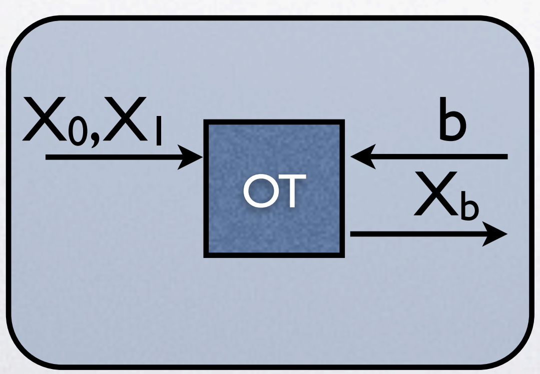
- Wyner's wire-tap channel: information-theoretically secret communication, without shared keys [W'75]
- Oblivious Transfer from noisy channel [CK'88]



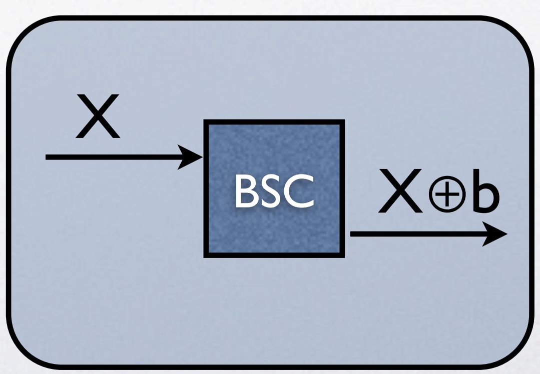
[R'81,W'83]

Noisy Channel & Crypto

- Wyner's wire-tap channel: information-theoretically secret communication, without shared keys [W'75]
- Oblivious Transfer from noisy channel [CK'88]

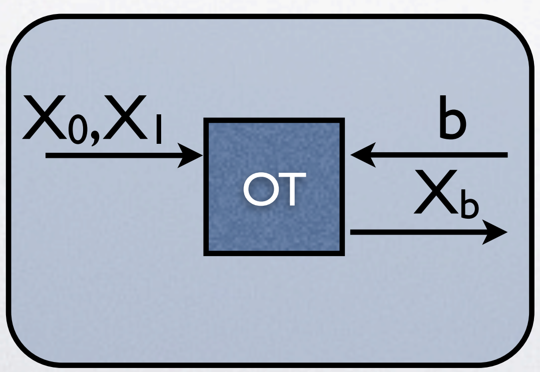


[R'81,W'83]

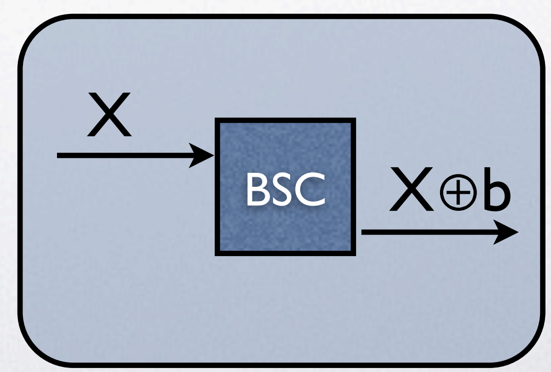


Noisy Channel & Crypto

- Wyner's wire-tap channel: information-theoretically secret communication, without shared keys [W'75]
- Oblivious Transfer from noisy channel [CK'88]
  - OT is complete for secure computation [K'88]



[R'81,W'83]



Constant Rate

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?
  - [CK'88] $O(k^{11})$ to get a security error of 2^{-k}

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?
  - [CK'88] $O(k^{11})$ to get a security error of 2^{-k}
  - [C'97] $O(k^3)$

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?
  - [CK'88] $O(k^{11})$ to get a security error of 2^{-k}
  - [C'97] $O(k^3)$
  - [CMW'04] $O(k^{2+\varepsilon})$

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?
  - [CK'88] $O(k^{11})$ to get a security error of 2^{-k}
  - [C'97] $O(k^3)$
  - [CMW'04] $O(k^{2+\varepsilon})$
  - [HIKN'08] $O(1)$ for *semi-honest* security

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?
  - [CK'88] $O(k^{11})$ to get a security error of 2^{-k}
  - [C'97] $O(k^3)$
  - [CMW'04] $O(k^{2+\varepsilon})$
  - [HIKN'08] $O(1)$ for *semi-honest* security
- Goal: To get $O(1)$ (Can't do better even given free noiseless channels [WW'10])

Constant Rate

- cf. Shannon's Channel Coding Theorem: $O(1)$ many uses of BSC per bit of communication
- How many uses of BSC per OT instance?

  or more general noisy channels

  - [CK'88] $O(k^{11})$ to get a security error of 2^{-k}
  - [C'97] $O(k^3)$
  - [CMW'04] $O(k^{2+\varepsilon})$
  - [HIKN'08] $O(1)$ for *semi-honest* security
- Goal: To get $O(1)$ (Can't do better even given free noiseless channels [WW'10])

Overview

Overview

- Plan: use IPS construction [IPS'08] to compile a semi-honest secure "inner protocol" and an honest-majority secure "outer protocol" using a few *string*-OTs

Overview

- Plan: use IPS construction [IPS'08] to compile a semi-honest secure "inner protocol" and an honest-majority secure "outer protocol" using a few *string*-OTs
  - A modified compiler so that the inner-protocol can use noisy channels. Requires inner protocol to be "error tolerant"

Overview

- Plan: use IPS construction [IPS'08] to compile a semi-honest secure "inner protocol" and an honest-majority secure "outer protocol" using a few *string*-OTs
  - A modified compiler so that the inner-protocol can use noisy channels. Requires inner protocol to be "error tolerant"

Harder to detect cheating in inner-protocol (by partial oblivious monitoring), as there is a noisy channel involved.

Will require the inner-protocol to be secure against *active* corruption of a small fraction of *channel instances*

Overview

- Plan: use IPS construction [IPS'08] to compile a semi-honest secure "inner protocol" and an honest-majority secure "outer protocol" using a few *string*-OTs
  - A modified compiler so that the inner-protocol can use noisy channels. Requires inner protocol to be "error tolerant"
    - Constant-rate inner and outer protocols from literature [GMW'87+HIKN'08,DI'06+CC'06]

Harder to detect cheating in inner-protocol (by partial oblivious monitoring), as there is a noisy channel involved.

Will require the inner-protocol to be secure against *active* corruption of a small fraction of *channel instances*

Overview

- Plan: use IPS construction [IPS'08] to compile a semi-honest secure "inner protocol" and an honest-majority secure "outer protocol" using a few *string*-OTs
  - A modified compiler so that the inner-protocol can use noisy channels. Requires inner protocol to be "error tolerant"
    - Constant-rate inner and outer protocols from literature [GMW'87+HIKN'08,DI'06+CC'06]
  - A constant-rate construction for *string*-OT from noisy channel

Harder to detect cheating in inner-protocol (by partial oblivious monitoring), as there is a noisy channel involved.

Will require the inner-protocol to be secure against *active* corruption of a small fraction of *channel instances*

String-OT

String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

Previously, known from OT-like and erasure channels [BCW'03,IMN'06]

String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

Previously, known from OT-like and erasure channels [BCW'03,IMN'06]

- Can use current constructions with a constant security parameter to get "fuzzy" OT: i.e., with constant security error

String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

  Previously, known from OT-like and erasure channels [BCW'03,IMN'06]

- Can use current constructions with a constant security parameter to get "fuzzy" OT: i.e., with constant security error
  - Challenge: change constant security error to negligible error

String-OT

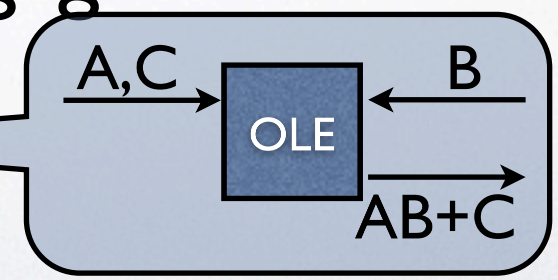
- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

  Previously, known from OT-like and erasure channels [BCW'03,IMN'06]

- Can use current constructions with a constant security parameter to get "fuzzy" OT: i.e., with constant security error
  - Challenge: change constant security error to negligible error
  - String-OT from fuzzy OT (or fuzzy OLE, in fact)

String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)
  - Previously, known from OT-like and erasure channels [BCW'03,IMN'06]
- Can use current constructions with a constant security parameter to get "fuzzy" OT: i.e., with constant security error
  - Challenge: change constant security error to negligible error
  - String-OT from fuzzy OT (or fuzzy OLE, ~~in fact~~)

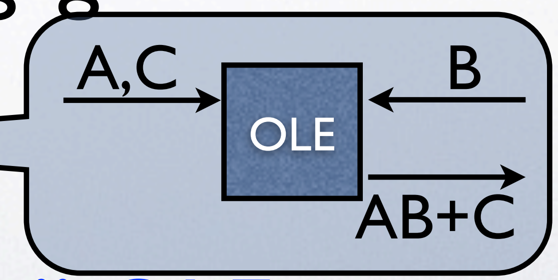


String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

  Previously, known from OT-like and erasure channels [BCW'03,IMN'06]

- Can use current constructions with a constant security parameter to get "fuzzy" OT: i.e., with constant security error
  - Challenge: change constant security error to negligible error
  - String-OT from fuzzy OT (or fuzzy OLE, ~~in fact~~)



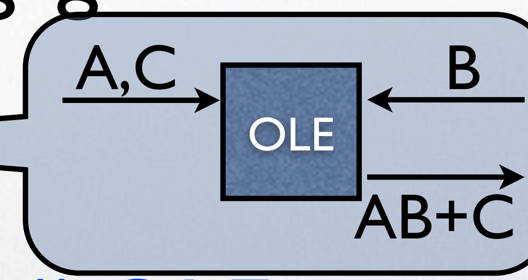
    - First, reinterpret fuzzy OLE as a perfect "shaky" OLE

String-OT

- t-bit string-OT with $O(t)+poly(k)$ communication (over a noisy channel)

  Previously, known from OT-like and erasure channels [BCW'03,IMN'06]

- Can use current constructions with a constant security parameter to get "fuzzy" OT: i.e., with constant security error
  - Challenge: change constant security error to negligible error
  - String-OT from fuzzy OT (or fuzzy OLE, ~~in fact~~)



    - First, reinterpret fuzzy OLE as a perfect "shaky" OLE
    - Next, use shaky OLE to get string-OT

Fuzzy and Shaky

Fuzzy and Shaky

- Fuzzy protocol: realizes F with a constant security error ε (statistical distance between ideal and real executions)

Fuzzy and Shaky

- Fuzzy <u>protocol</u>: realizes F with a constant security error ε (statistical distance between ideal and real executions)
- Shaky <u>functionality</u>: $F^{((\sigma))}$ flips a σ-biased coin, and if heads, then works as F, else (w/ prob σ) surrenders to the adversary

Fuzzy and Shaky

- Fuzzy protocol: realizes F with a constant security error ε (statistical distance between ideal and real executions)
- Shaky functionality: $F^{((\sigma))}$ flips a σ-biased coin, and if heads, then works as F, else (w/ prob σ) surrenders to the adversary
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

Fuzzy and Shaky

- Fuzzy <u>protocol</u>: realizes F with a constant security error ε (statistical distance between ideal and real executions)
- Shaky <u>functionality</u>: $F^{((\sigma))}$ flips a σ-biased coin, and if heads, then works as F, else (w/ prob σ) surrenders to the adversary
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

$\sigma = \#rounds.|X||Y|\varepsilon$

Fuzzy and Shaky

- Fuzzy protocol: realizes F with a constant security error ε (statistical distance between ideal and real executions)
- Shaky functionality: $F^{((\sigma))}$ flips a σ-biased coin, and if heads, then works as F, else (w/ prob σ) surrenders to the adversary
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

  $\sigma = \#rounds.|X||Y|\varepsilon$

  - As a composition theorem: Running n copies of an ε-fuzzy protocol gives about $(1\text{-}\sigma)n$ good copies of F (randomly chosen)

Fuzzy to Shaky

Fuzzy to Shaky

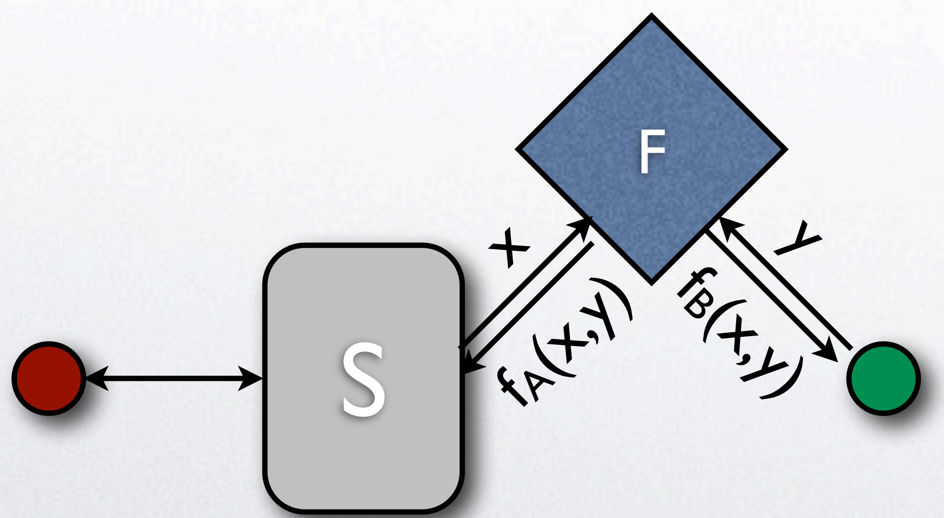
- “Statistical security to Perfect security”

Fuzzy to Shaky

- "Statistical security to Perfect security"
- Works for UC-security (as well as standalone security)

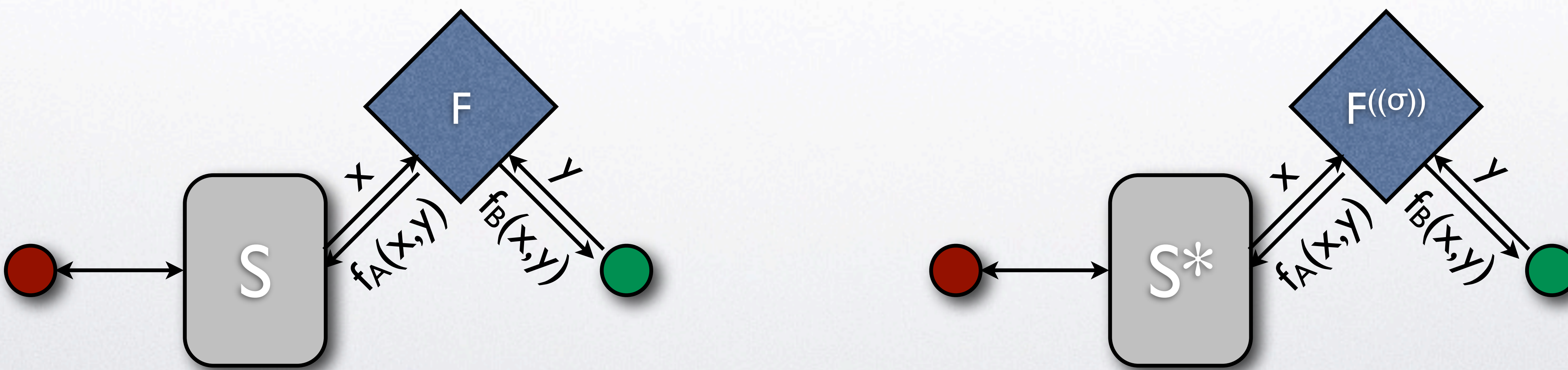
Fuzzy to Shaky

- "Statistical security to Perfect security"
- Works for UC-security (as well as standalone security)
  - Given a simulator for F with error ε, build a perfect simulator for $F^{((\sigma))}$



Fuzzy to Shaky

- "Statistical security to Perfect security"
- Works for UC-security (as well as standalone security)
  - Given a simulator for F with error ε, build a perfect simulator for $F^{((\sigma))}$



Fuzzy→Shaky: Example

Fuzzy→Shaky: Example

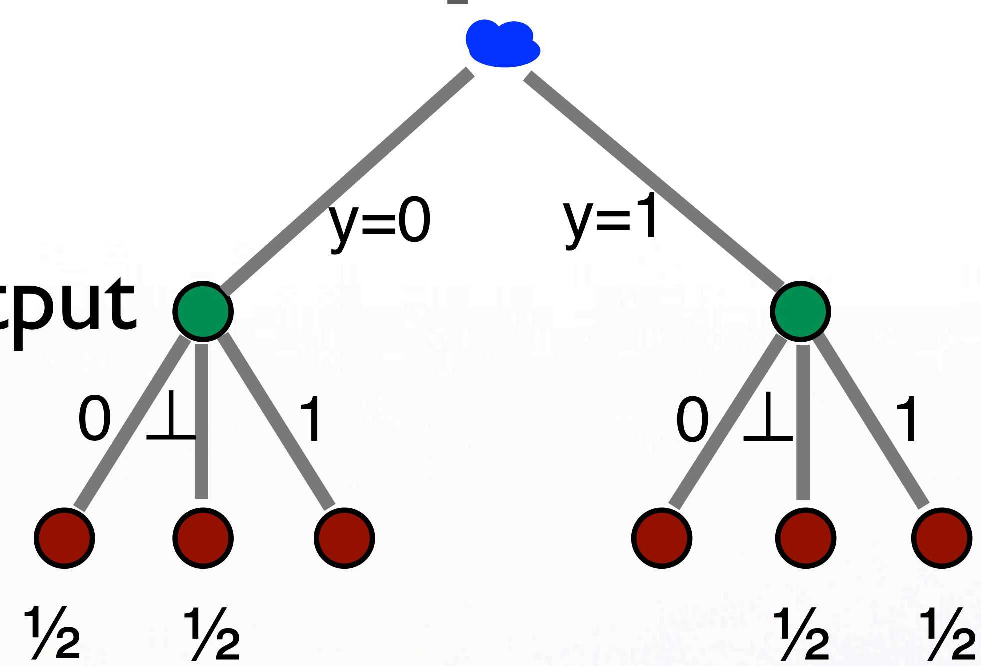
- A degenerate functionality F

Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output

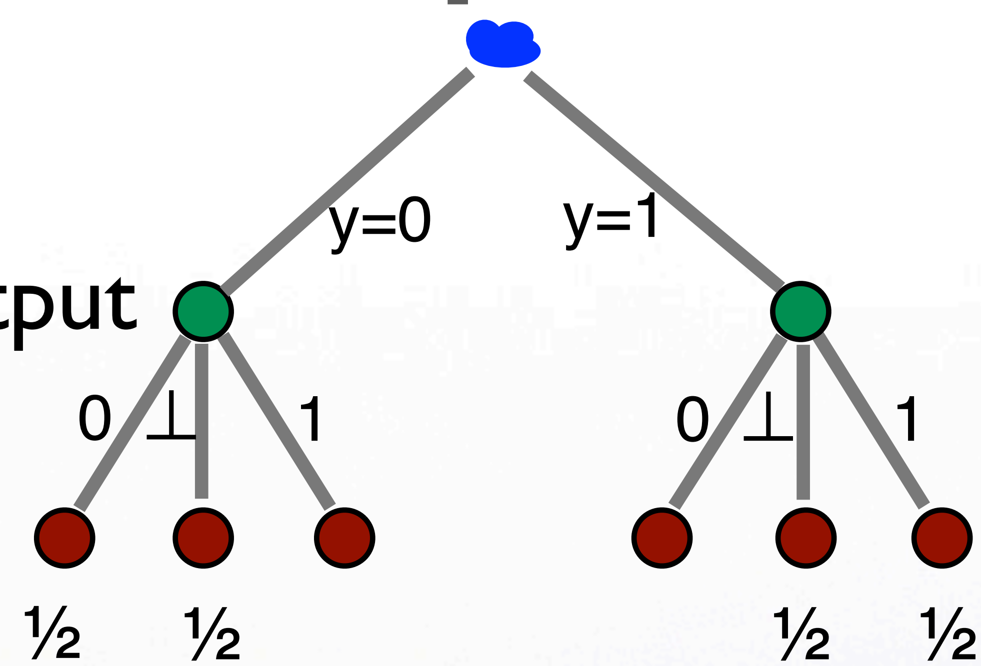
Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output
- A fuzzy protocol: With probability ½ Bob sends his input to Alice, else ⊥



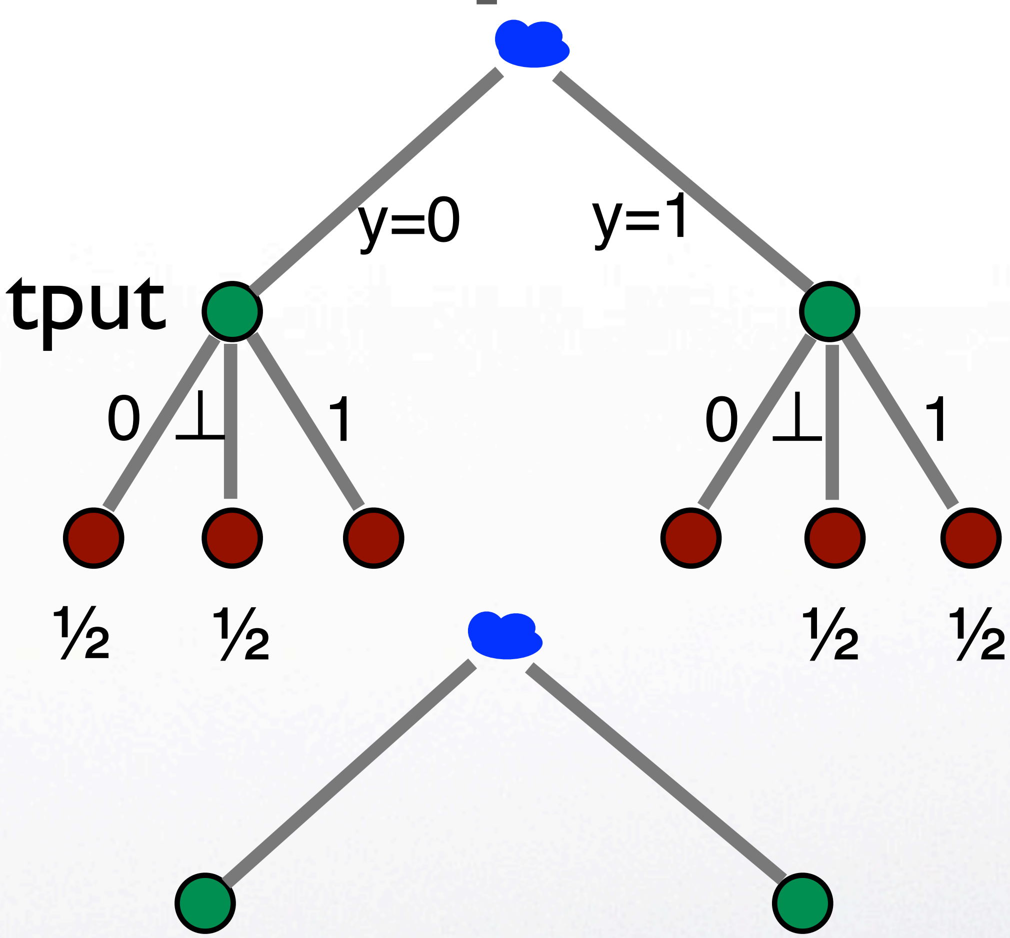
Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output
- A fuzzy protocol: With probability ½ Bob sends his input to Alice, else ⊥
  - For corrupt Alice, simulator in the ideal F execution sends ⊥ with probability ½, and else a random bit



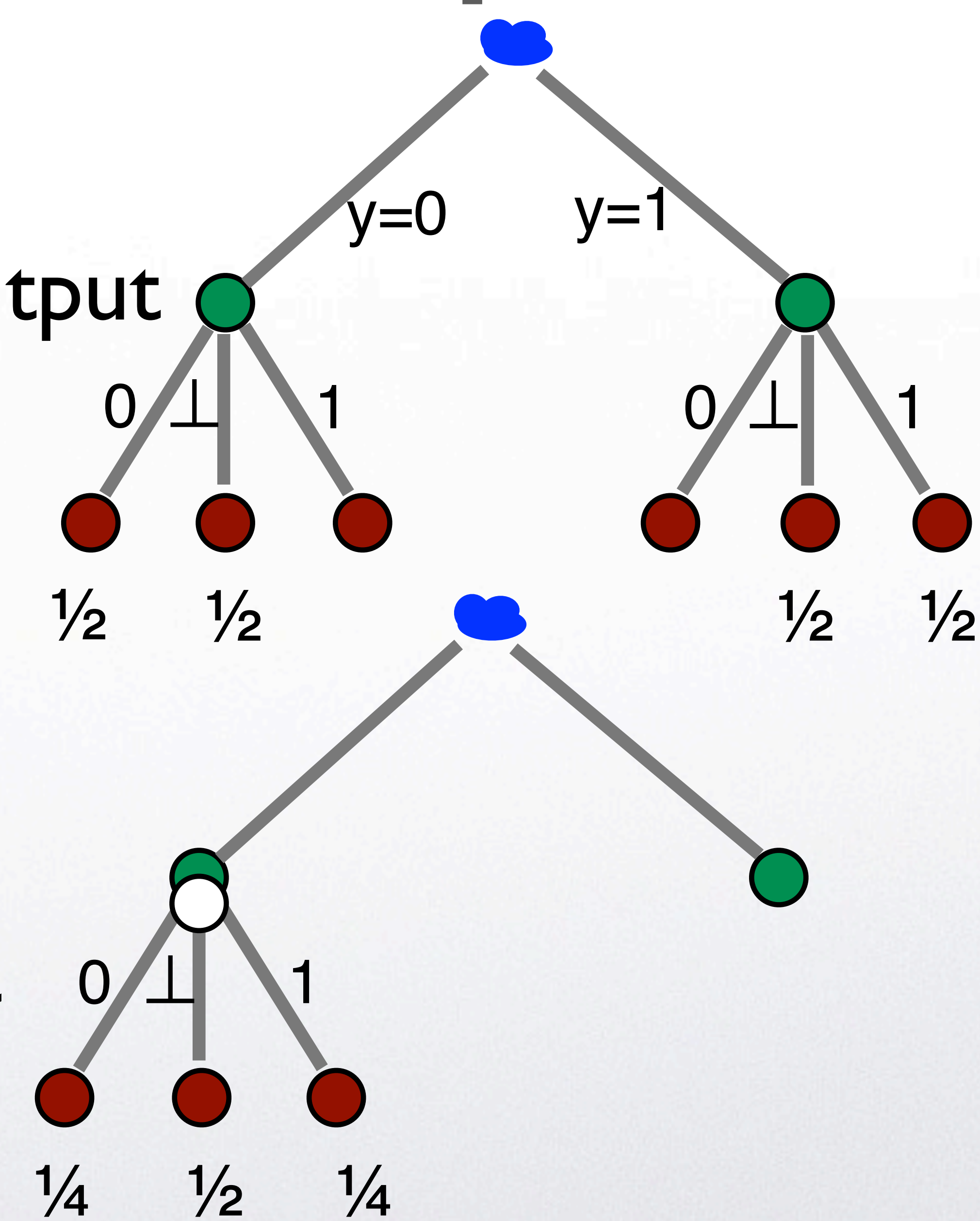
Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output
- A fuzzy protocol: With probability ½ Bob sends his input to Alice, else ⊥
  - For corrupt Alice, simulator in the ideal F execution sends ⊥ with probability ½, and else a random bit



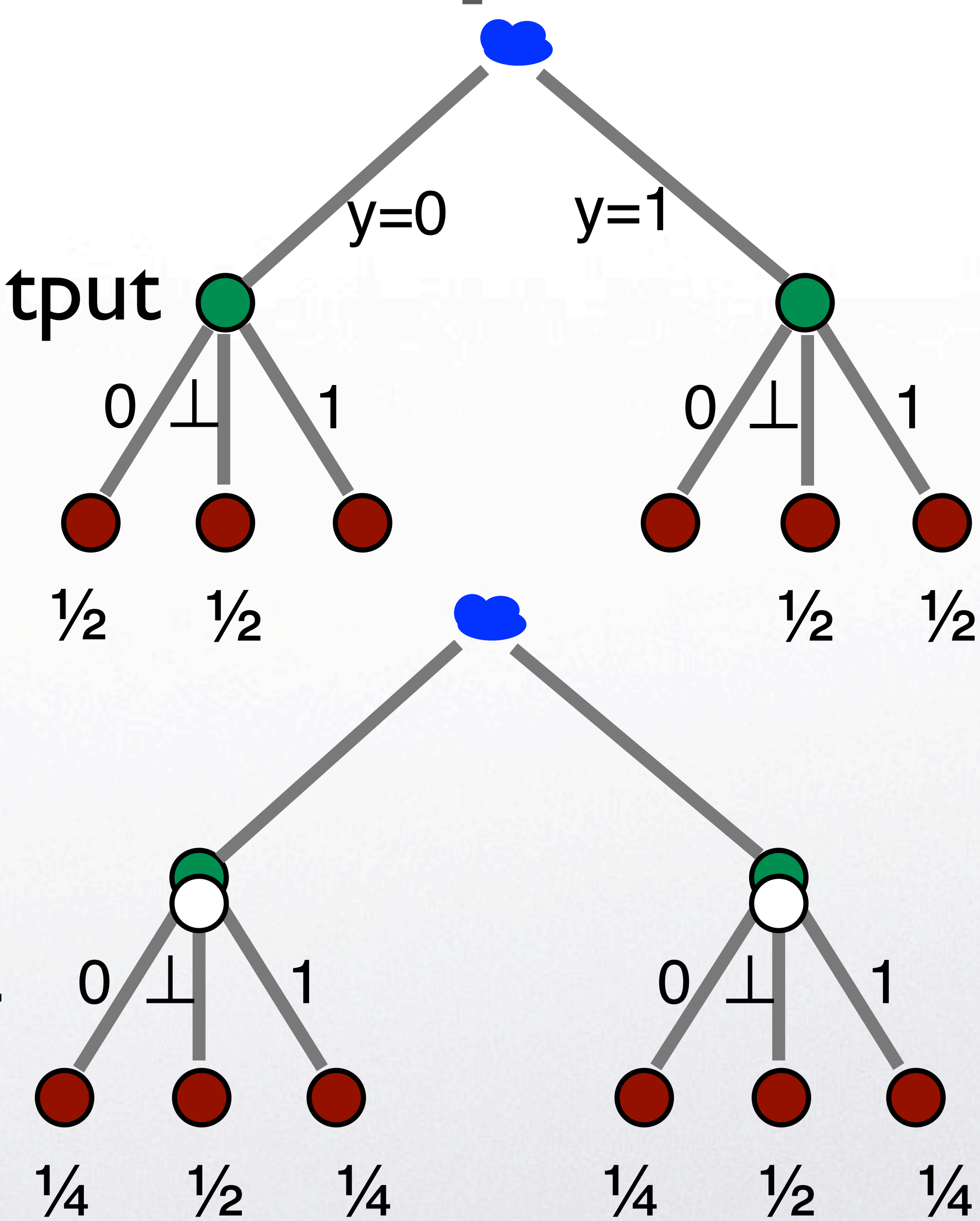
Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output
- A fuzzy protocol: With probability ½ Bob sends his input to Alice, else ⊥
  - For corrupt Alice, simulator in the ideal F execution sends ⊥ with probability ½, and else a random bit



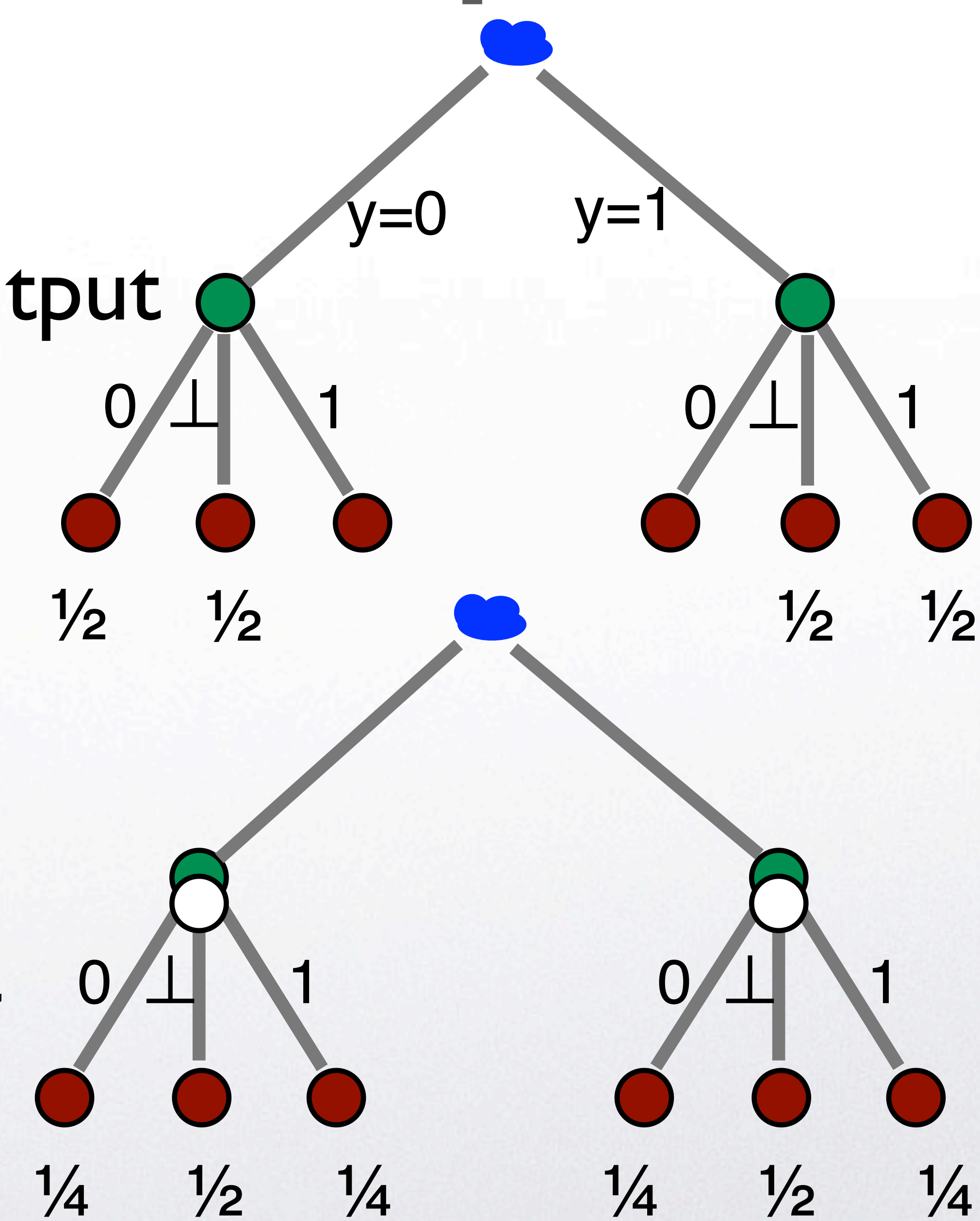
Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output
- A fuzzy protocol: With probability ½ Bob sends his input to Alice, else ⊥
  - For corrupt Alice, simulator in the ideal F execution sends ⊥ with probability ½, and else a random bit

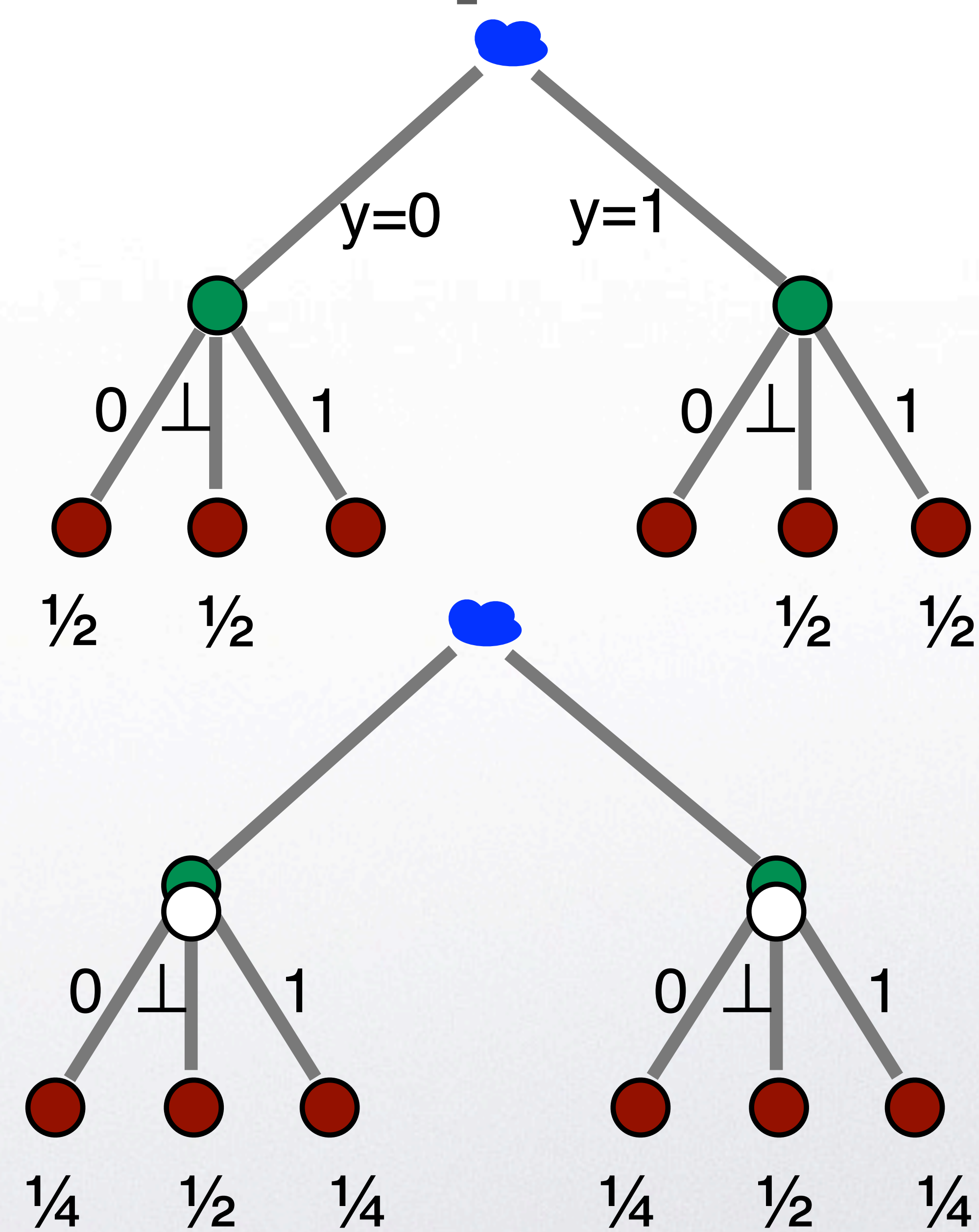


Fuzzy→Shaky: Example

- A degenerate functionality F
  - Takes a bit from Bob as input; no output
- A fuzzy protocol: With probability ½ Bob sends his input to Alice, else ⊥
  - For corrupt Alice, simulator in the ideal F execution sends ⊥ with probability ½, and else a random bit
  - Simulation error = ¼

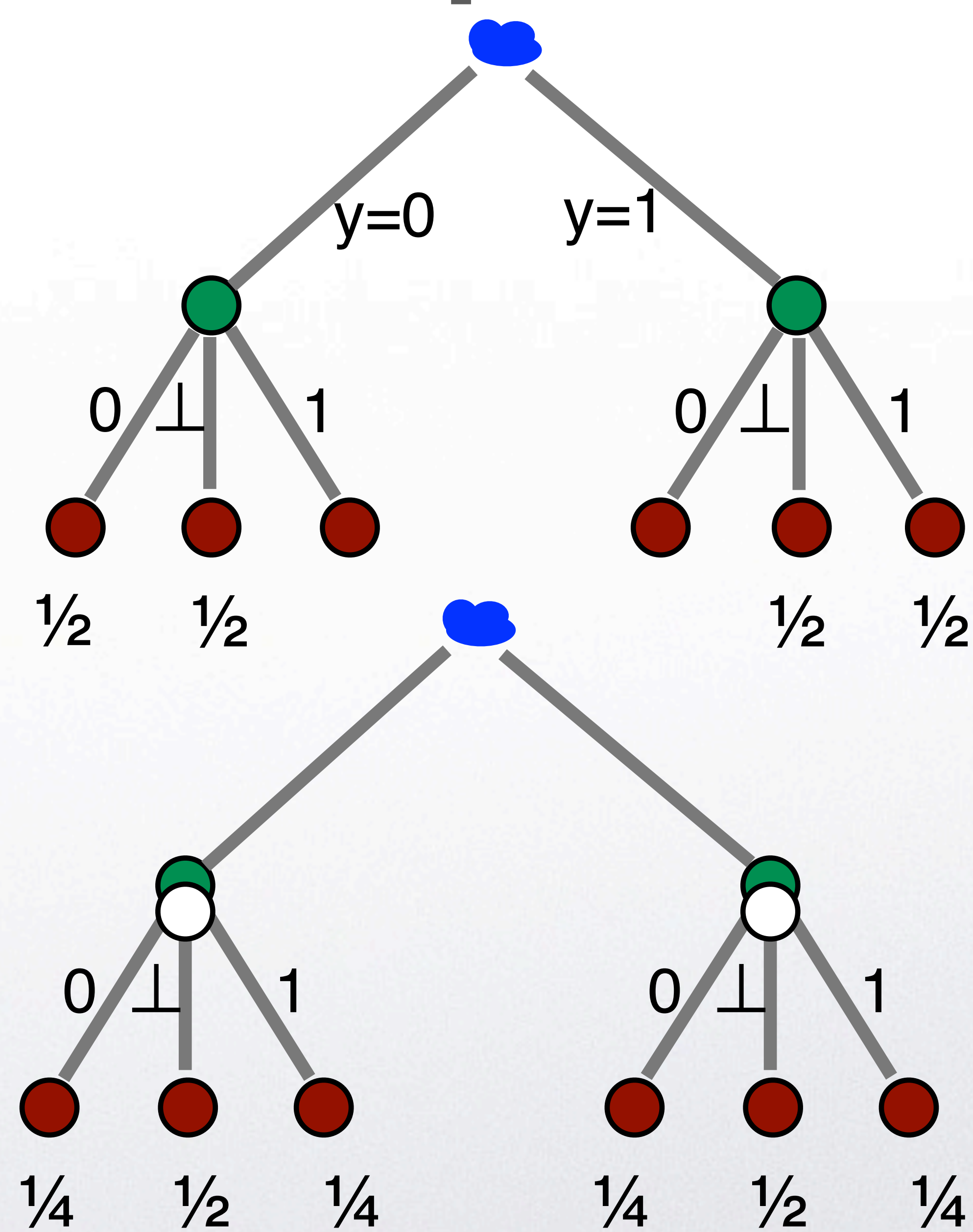


Fuzzy→Shaky: Example



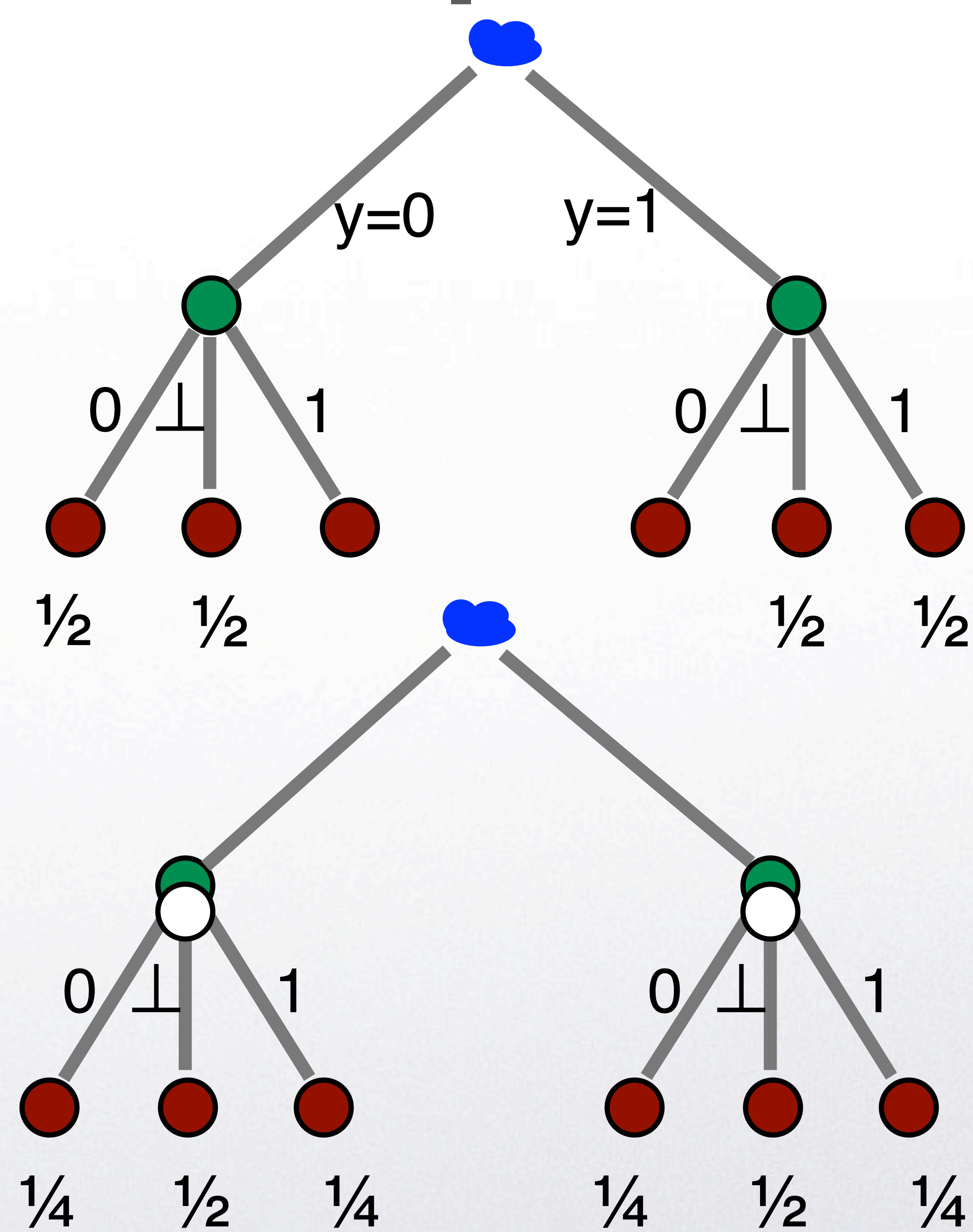
Fuzzy→Shaky: Example

- Simulator for $F^{((1/2))}$ in two parts:



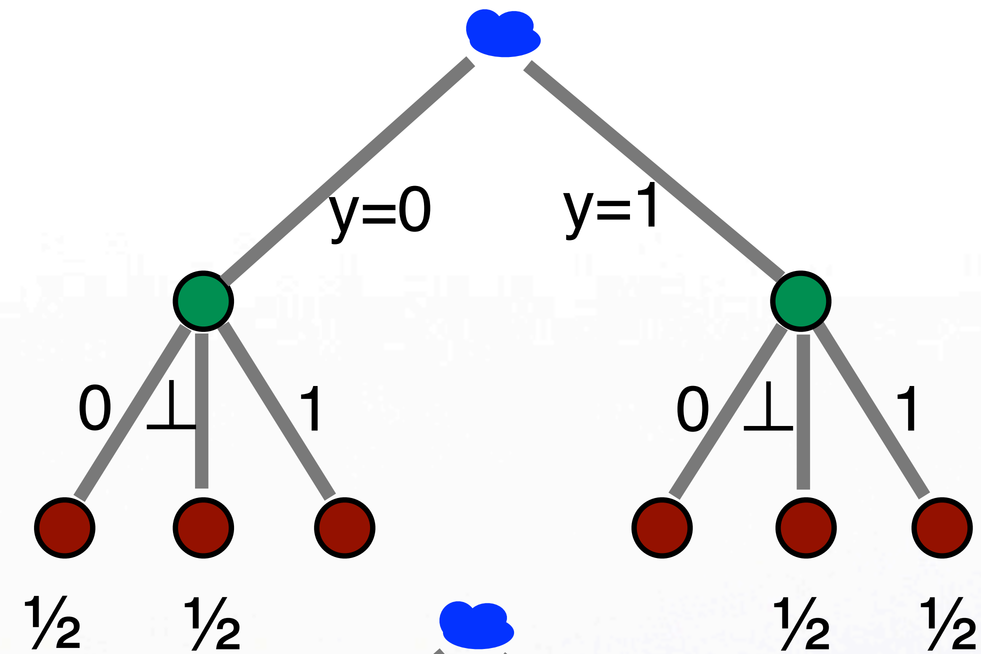
Fuzzy→Shaky: Example

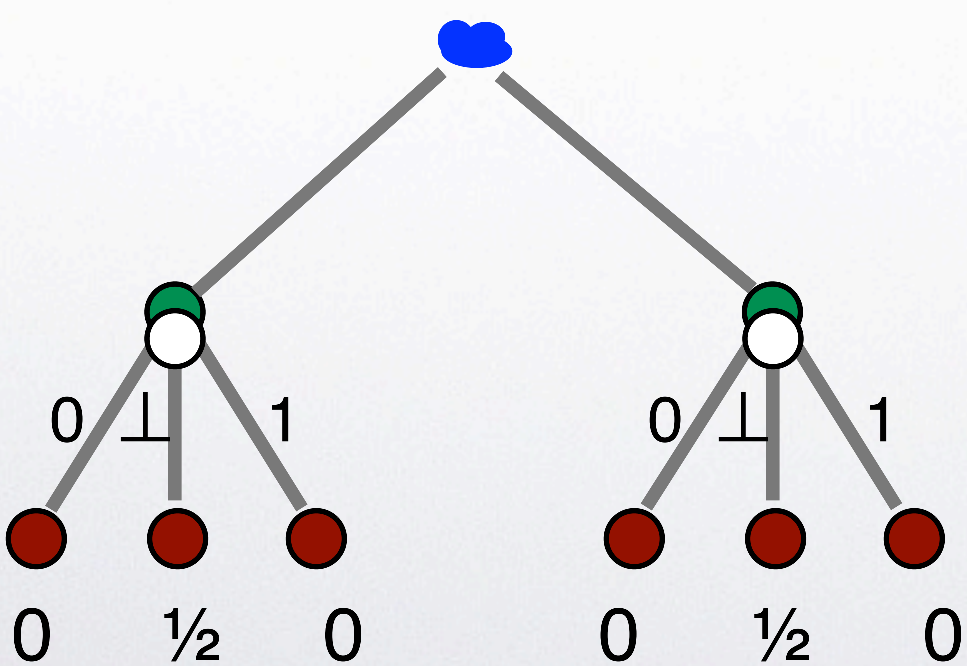
- Simulator for $F^{((1/2))}$ in two parts:
  - A part "dominated" both by the protocol and the given simulation

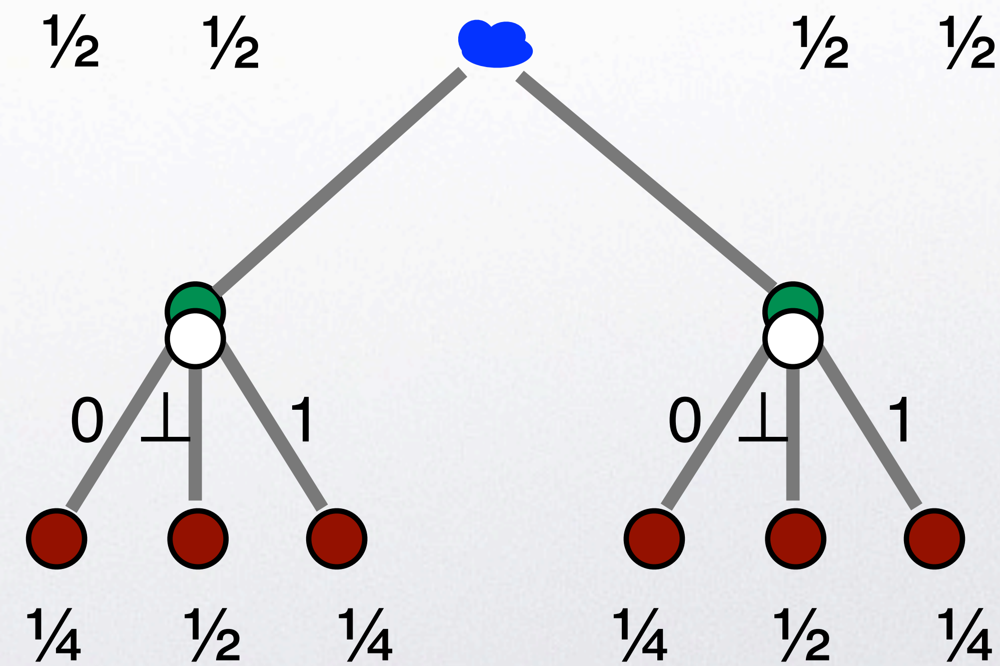


Fuzzy→Shaky: Example

- Simulator for $F^{((1/2))}$ in two parts:
  - A part "dominated" both by the protocol and the given simulation

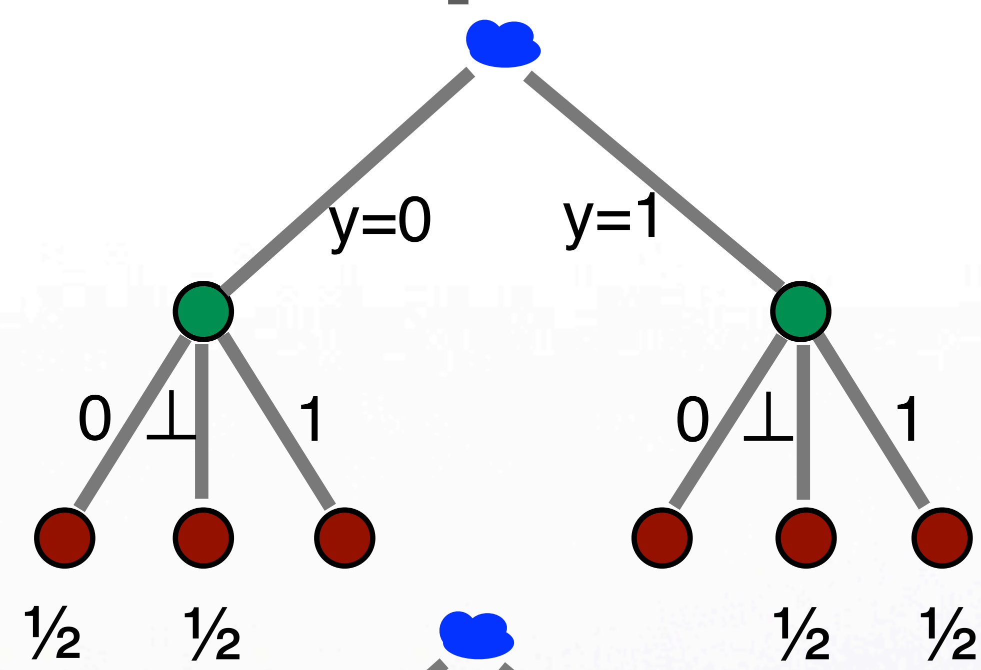


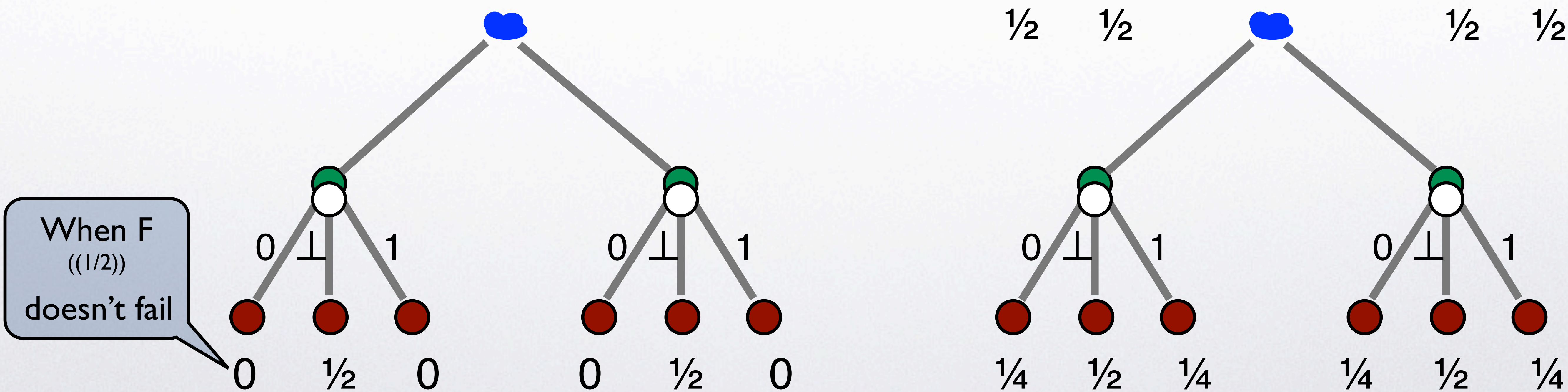




Fuzzy→Shaky: Example

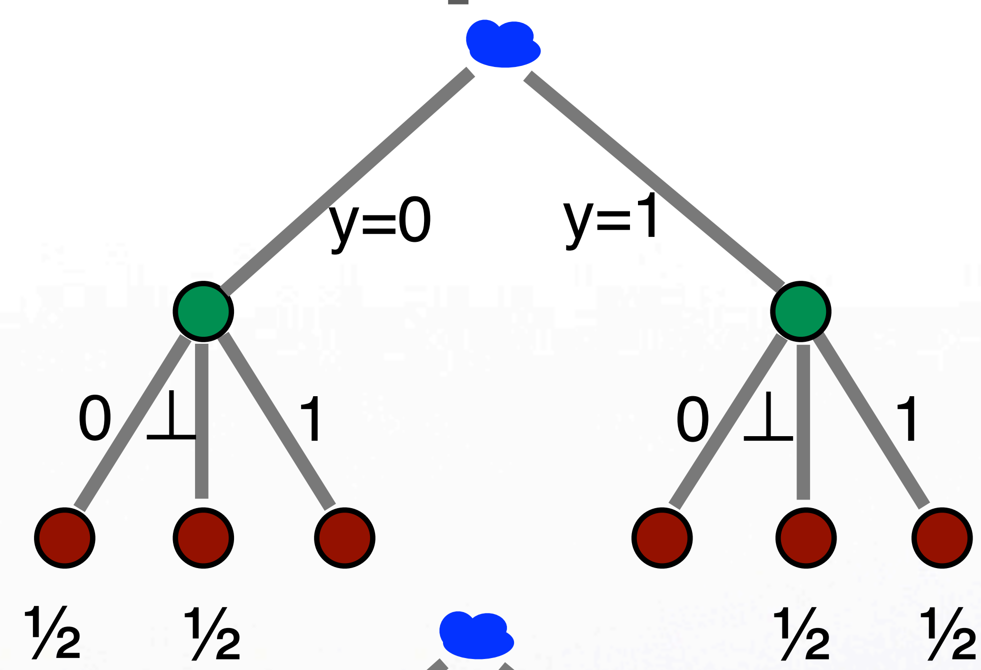
- Simulator for $F^{((1/2))}$ in two parts:
  - A part "dominated" both by the protocol and the given simulation



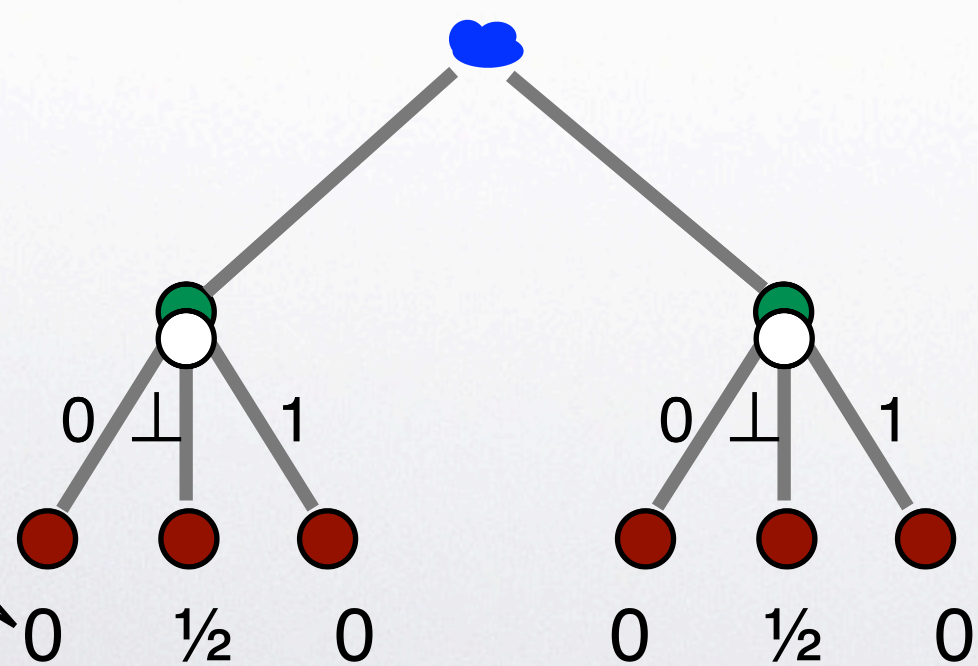


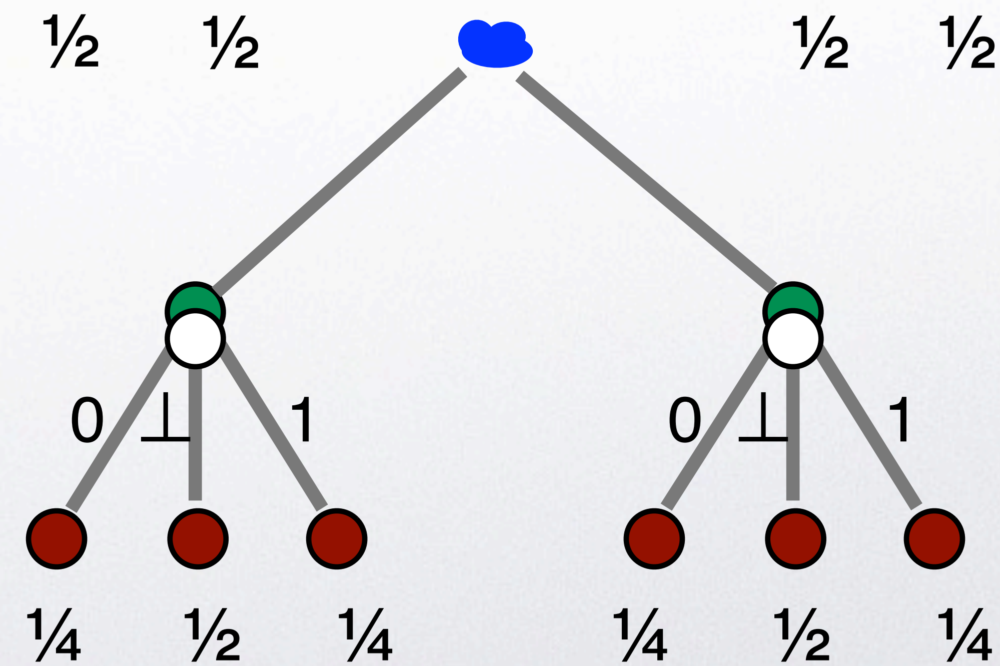
Fuzzy→Shaky: Example

- Simulator for $F^{((1/2))}$ in two parts:
  - A part "dominated" both by the protocol and the given simulation
  - The "remainder" to make it perfect



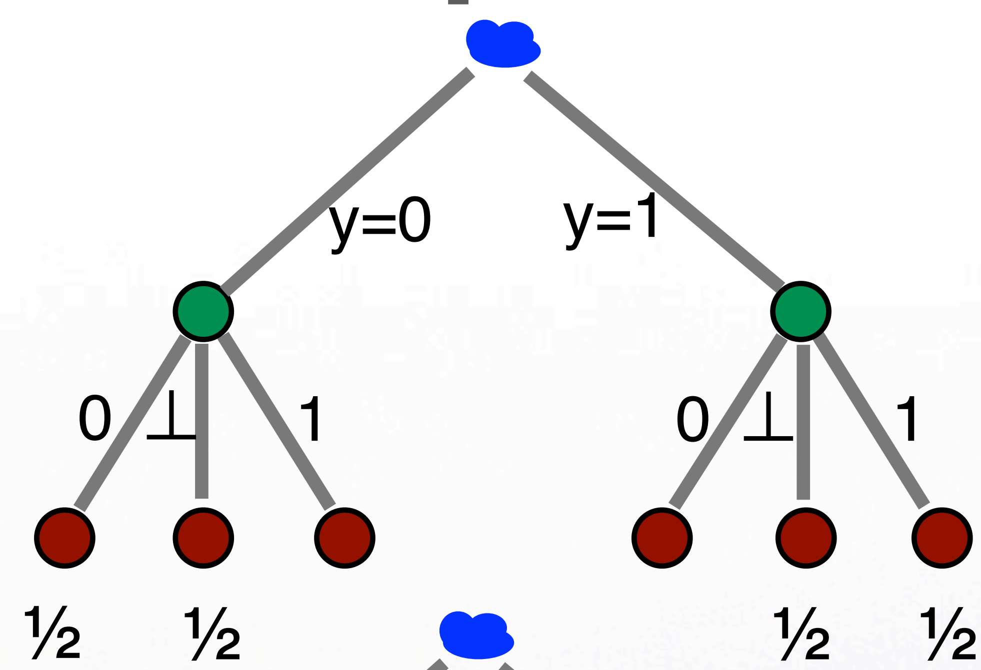
When F ((1/2)) doesn't fail

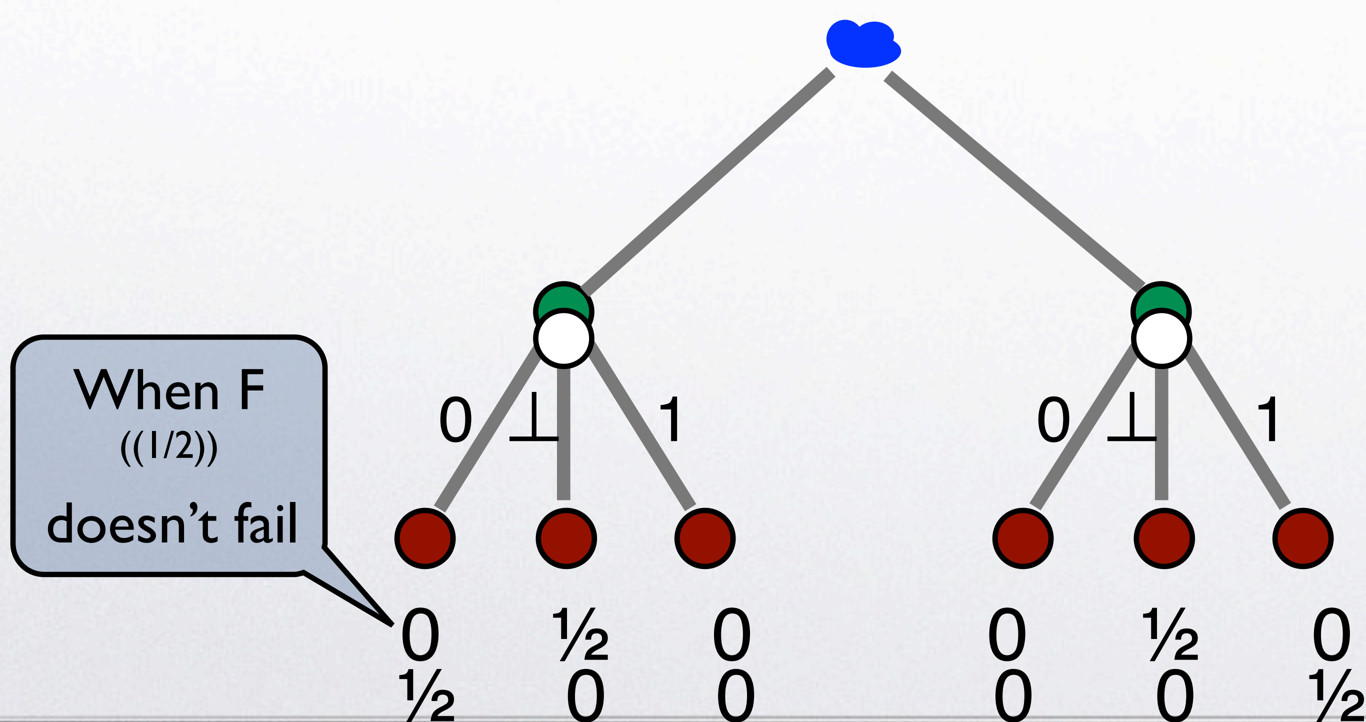


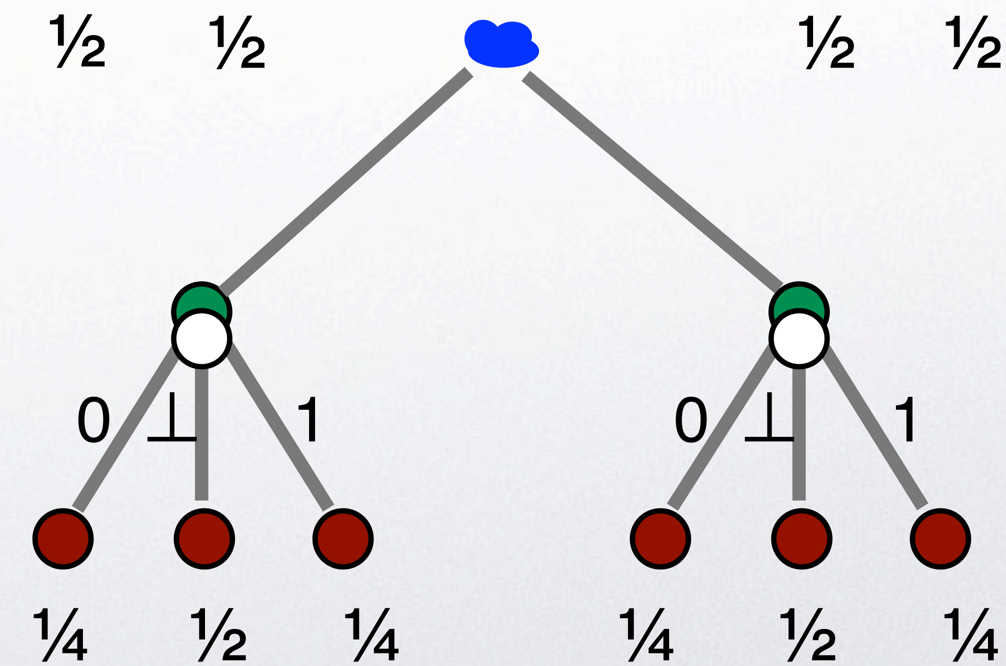


Fuzzy→Shaky: Example

- Simulator for $F^{((1/2))}$ in two parts:
  - A part "dominated" both by the protocol and the given simulation
  - The "remainder" to make it perfect

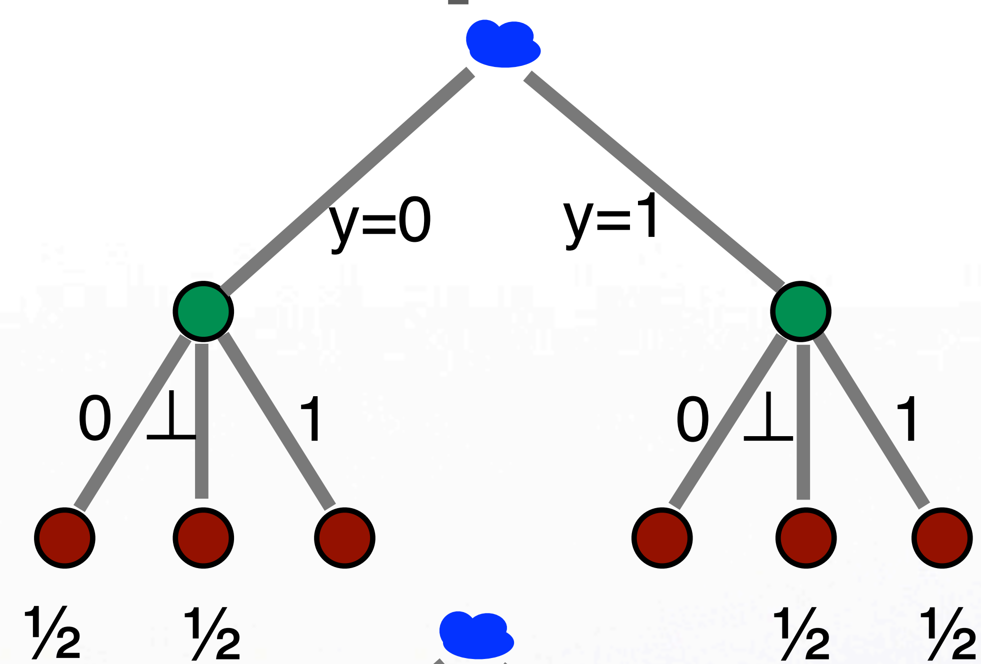


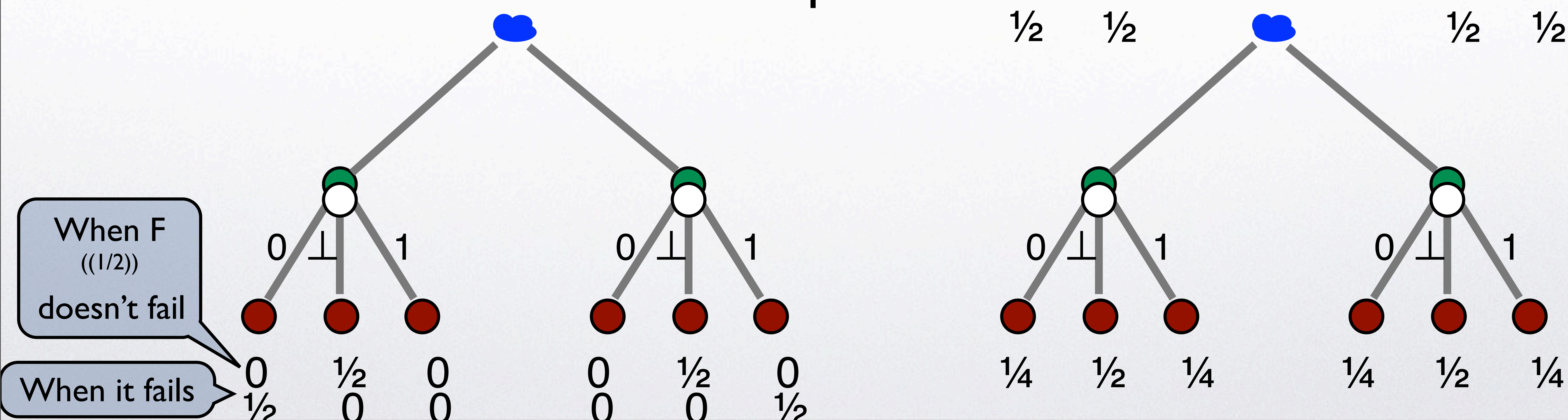




Fuzzy→Shaky: Example

- Simulator for $F^{((1/2))}$ in two parts:
  - A part "dominated" both by the protocol and the given simulation
  - The "remainder" to make it perfect





Fuzzy to Shaky

Fuzzy to Shaky

- Much more complicated when Alice has an input or output

Fuzzy to Shaky

- Much more complicated when Alice has an input or output
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

Fuzzy to Shaky

- Much more complicated when Alice has an input or output
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

$\sigma = \#rounds.|X||Y|\varepsilon$

Fuzzy to Shaky

- Much more complicated when Alice has an input or output
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$
  $\sigma = \#rounds.|X||Y|\varepsilon$
- Holds for any deterministic function F

Fuzzy to Shaky

- Much more complicated when Alice has an input or output
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

  $\sigma = \#rounds.|X||Y|\varepsilon$
- Holds for any deterministic function F
- Simulator's description is exponential in the fuzzy protocol's communication complexity

Fuzzy to Shaky

- Much more complicated when Alice has an input or output
- **Theorem**
  An ε-fuzzy protocol for F is a perfectly secure protocol for $F^{((\sigma))}$

  $\sigma = \#rounds.|X||Y|\varepsilon$
- Holds for any deterministic function F
- Simulator's description is exponential in the fuzzy protocol's communication complexity
  - But for us, this is a constant: fuzzy OLE is a (non-constant rate) OLE protocol instantiated with a constant security parameter

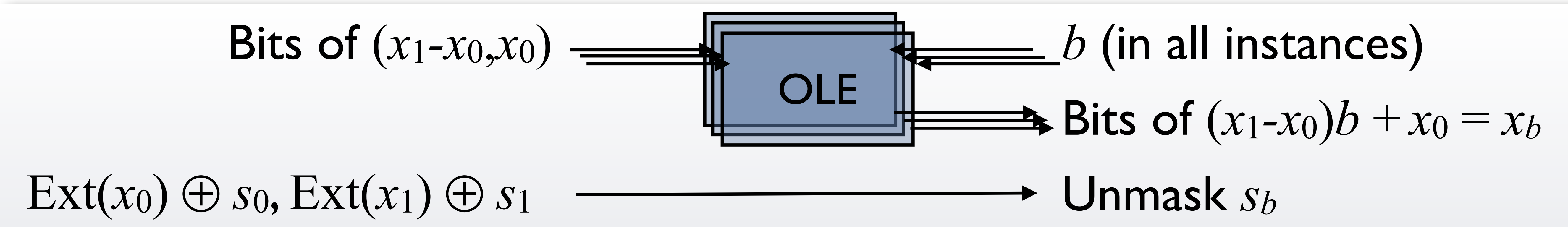
Shaky OLE to String-OT

Shaky OLE to String-OT

- (Non-shaky) OLE to String-OT:

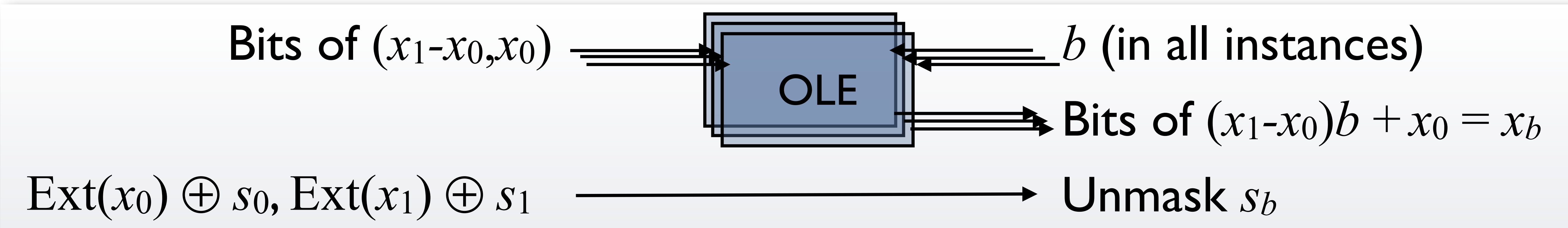
Shaky OLE to String-OT

- (Non-shaky) OLE to String-OT:



Shaky OLE to String-OT

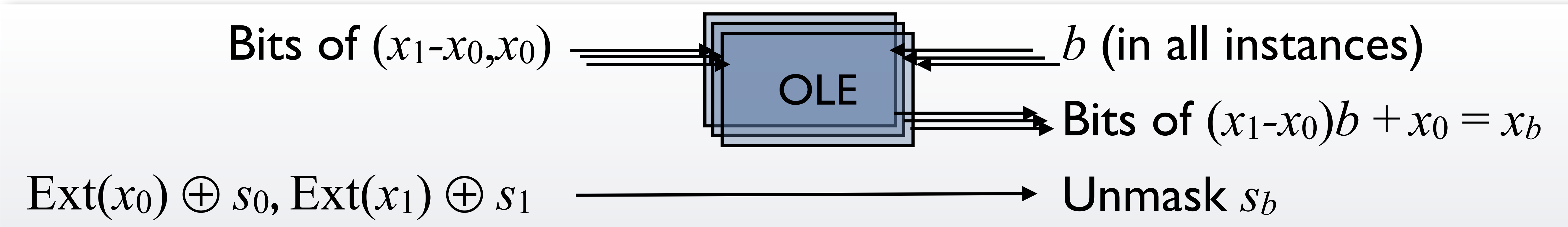
- (Non-shaky) OLE to String-OT:



  - Alice "extracts" fewer than $n/2$ bits from each of x_0 and x_1 and sends $\mathrm{Ext}(x_0) \oplus s_0$ and $\mathrm{Ext}(x_1) \oplus s_1$ to Bob

Shaky OLE to String-OT

- (Non-shaky) OLE to String-OT:

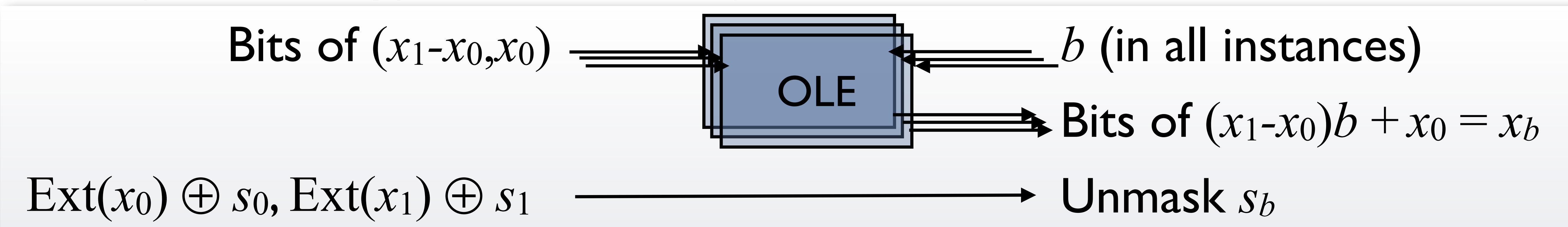


  - Alice "extracts" fewer than $n/2$ bits from each of x_0 and x_1 and sends $\mathrm{Ext}(x_0) \oplus s_0$ and $\mathrm{Ext}(x_1) \oplus s_1$ to Bob

- But with shaky OLE, Alice may learn Bob's input b (and Bob may learn more than $n/2$ bits each of x_0 and x_1)

Shaky OLE to String-OT

- (Non-shaky) OLE to String-OT:



  - Alice "extracts" fewer than $n/2$ bits from each of x_0 and x_1 and sends $\mathrm{Ext}(x_0) \oplus s_0$ and $\mathrm{Ext}(x_1) \oplus s_1$ to Bob
- But with shaky OLE, Alice may learn Bob's input b (and Bob may learn more than $n/2$ bits each of x_0 and x_1)
- Fix: using a constant-rate encoding of x_0, x_1 and b

Shaky OLE to String-OT

Shaky OLE to String-OT

- Const. rate encodings $\mathrm{Enc}:\mathbb{F}^m \rightarrow \mathbb{F}^n$ and $\mathrm{Enc}^2:\mathbb{F}^m \rightarrow \mathbb{F}^n$ such that:

Shaky OLE to String-OT

- Const. rate encodings $\mathrm{Enc}:\mathbb{F}^m \rightarrow \mathbb{F}^n$ and $\mathrm{Enc}^2:\mathbb{F}^m \rightarrow \mathbb{F}^n$ such that:
  - $\mathrm{Enc}(A) * \mathrm{Enc}(B) + \mathrm{Enc}^2(C) \in \mathrm{Enc}^2(AB+C)$

Shaky OLE to String-OT

- Const. rate encodings $\mathrm{Enc}:\mathbb{F}^m \rightarrow \mathbb{F}^n$ and $\mathrm{Enc}^2:\mathbb{F}^m \rightarrow \mathbb{F}^n$ such that:
  - $\mathrm{Enc}(A) * \mathrm{Enc}(B) + \mathrm{Enc}^2(C) \in \mathrm{Enc}^2(AB+C)$

co-ordinate wise mult.

Shaky OLE to String-OT

- Const. rate encodings $\mathrm{Enc}:\mathbb{F}^m \rightarrow \mathbb{F}^n$ and $\mathrm{Enc}^2:\mathbb{F}^m \rightarrow \mathbb{F}^n$ such that:
  - $\mathrm{Enc}(A) * \mathrm{Enc}(B) + \mathrm{Enc}^2(C) \in \mathrm{Enc}^2(AB+C)$

    co-ordinate wise mult.
  - Error-correcting & Secret-sharing: For d = a (small) constant fraction of n, Enc^2 allows (efficient) decoding up to d errors; also, any d co-ordinates of Enc independent of the message

Shaky OLE to String-OT

- Const. rate encodings $\mathrm{Enc}:\mathbb{F}^m \to \mathbb{F}^n$ and $\mathrm{Enc}^2:\mathbb{F}^m \to \mathbb{F}^n$ such that:
  - $\mathrm{Enc}(A) * \mathrm{Enc}(B) + \mathrm{Enc}^2(C) \in \mathrm{Enc}^2(AB+C)$

    co-ordinate wise mult.
  - Error-correcting & Secret-sharing: For d = a (small) constant fraction of n, Enc^2 allows (efficient) decoding up to d errors; also, any d co-ordinates of Enc independent of the message
  - Enc^2 is sufficiently randomizing: $\mathrm{Enc}^2(A)$ is uniform over an n-$m(1+\delta)$-dimensional subspace of $\mathbb{F}^n$

Shaky OLE to String-OT

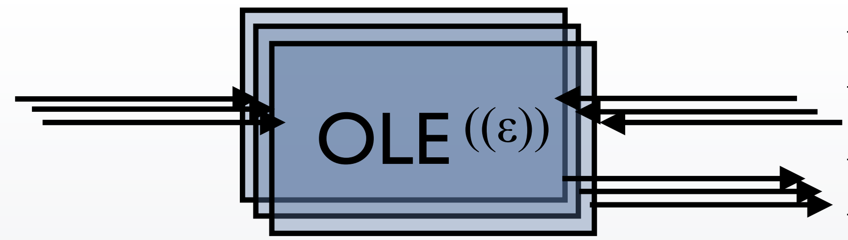
- Const. rate encodings $\mathrm{Enc}:\mathbb{F}^m \to \mathbb{F}^n$ and $\mathrm{Enc}^2:\mathbb{F}^m \to \mathbb{F}^n$ such that:
  - $\mathrm{Enc}(A) * \mathrm{Enc}(B) + \mathrm{Enc}^2(C) \in \mathrm{Enc}^2(AB+C)$

    co-ordinate wise mult.
  - Error-correcting & Secret-sharing: For d = a (small) constant fraction of n, Enc^2 allows (efficient) decoding up to d errors; also, any d co-ordinates of Enc independent of the message
  - Enc^2 is sufficiently randomizing: $\mathrm{Enc}^2(A)$ is uniform over an n-$m(1+\delta)$-dimensional subspace of $\mathbb{F}^n$
- Instantiated from an "MPC-friendly code" (a.k.a codex) of appropriate parameters [CC'06,IKOS'09, next talk]

Shaky OLE to String-OT

Shaky OLE to String-OT

Enc(x_1-x_0), Enc^2(x_0)



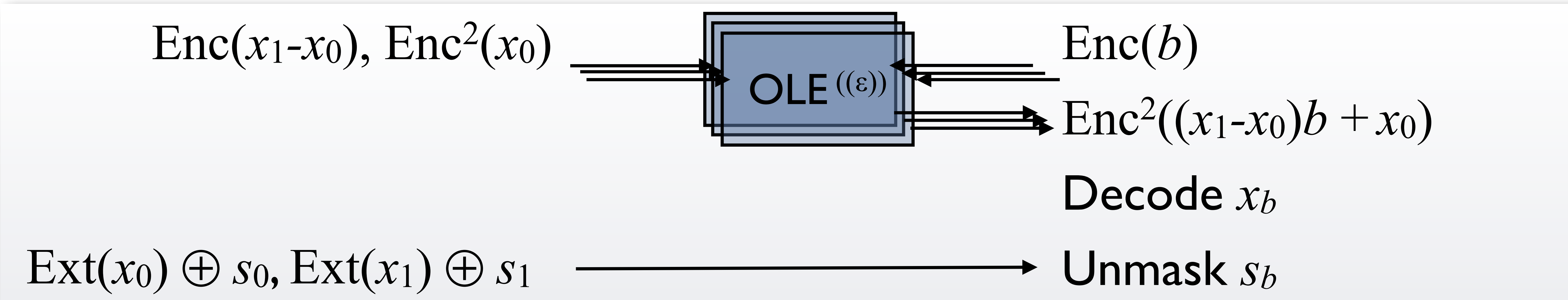
Enc(b)

Enc^2((x_1-x_0)b + x_0)

Decode x_b

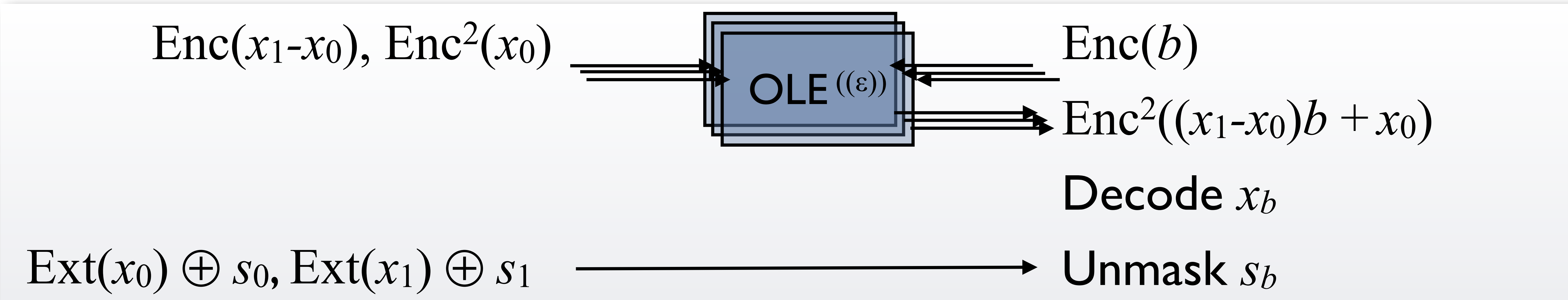
Ext(x_0) $\oplus$ s_0, Ext(x_1) $\oplus$ s_1 → Unmask s_b

Shaky OLE to String-OT



- Secure against Alice, since Bob can correct a constant fraction of errors, and since a small fraction of $\mathrm{Enc}(b)$ reveals nothing of b

Shaky OLE to String-OT



- Secure against Alice, since Bob can correct a constant fraction of errors, and since a small fraction of $\mathrm{Enc}(b)$ reveals nothing of b
- Secure against Bob, since he knows nothing of at least one of the extracted strings (even given the other one, and all that he gets in the protocol; relies on the randomization of $\mathrm{Enc}^2(x_0)$)

Summary

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
  - Using (a slightly modified) IPS compiler [IPS'08] to compile:

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
  - Using (a slightly modified) IPS compiler [IPS'08] to compile:
    - "Outer protocol" [DI'06+CC'06] for n instances of OT

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
    - Using (a slightly modified) IPS compiler [IPS'08] to compile:
        - "Outer protocol" [DI'06+CC'06] for n instances of OT
        - "Inner protocol" [GMW'87+HIKN'08] for implementing its servers

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
  - Using (a slightly modified) IPS compiler [IPS'08] to compile:
    - "Outer protocol" [DI'06+CC'06] for n instances of OT
    - "Inner protocol" [GMW'87+HIKN'08] for implementing its servers
    - For "watchlist channels" a new constant-rate protocol for *string*-OT from noisy channel (previously, only from an erasure channel)

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
  - Using (a slightly modified) IPS compiler [IPS'08] to compile:
    - "Outer protocol" [DI'06+CC'06] for n instances of OT
    - "Inner protocol" [GMW'87+HIKN'08] for implementing its servers
    - For "watchlist channels" a new constant-rate protocol for *string*-OT from noisy channel (previously, only from an erasure channel)
      - Uses a homomorphic arithmetic encoding scheme

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
  - Using (a slightly modified) IPS compiler [IPS'08] to compile:
    - "Outer protocol" [DI'06+CC'06] for n instances of OT
    - "Inner protocol" [GMW'87+HIKN'08] for implementing its servers
    - For "watchlist channels" a new constant-rate protocol for *string*-OT from noisy channel (previously, only from an erasure channel)
      - Uses a homomorphic arithmetic encoding scheme
      - Relies on "fuzzy to shaky" security

Thank You!

Summary

- Constant rate OT from BSC (and in fact, any noisy channel that gives OT)
  - Using (a slightly modified) IPS compiler [IPS'08] to compile:
    - "Outer protocol" [DI'06+CC'06] for n instances of OT
    - "Inner protocol" [GMW'87+HIKN'08] for implementing its servers
    - For "watchlist channels" a new constant-rate protocol for *string*-OT from noisy channel (previously, only from an erasure channel)
      - Uses a homomorphic arithmetic encoding scheme
      - Relies on "fuzzy to shaky" security