Constrained Pseudorandom Functions From Homomorphic Secret Sharing

Geoffroy Couteau[1], Pierre Meyer[1,2], Alain Passelègue[3,4], and Mahshid Riahinia[4]

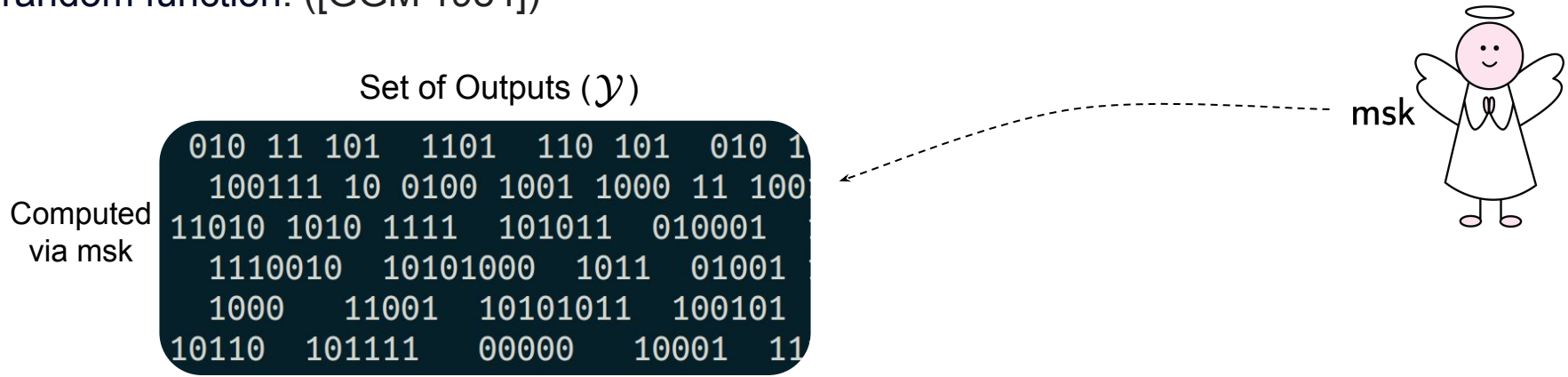
[1] Université Paris Cité, CNRS, IRIF, Paris, France.
[2] Reichman University, Herzliya, Israel.
[3] Inria, France.
[4] ENS de Lyon, Laboratoire LIP (U. Lyon, CNRS, ENSL, Inria, UCBL), France.

Journées C2 2023

Constrained Pseudorandom Function

Pseudorandom function: $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

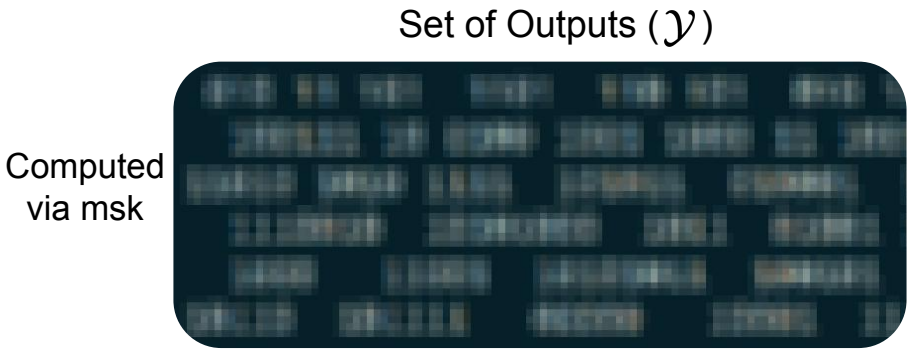
Definition. A deterministic keyed function that is computationally indistinguishable from a truly random function. ([GGM'1984])



Constrained Pseudorandom Function

Pseudorandom function: $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

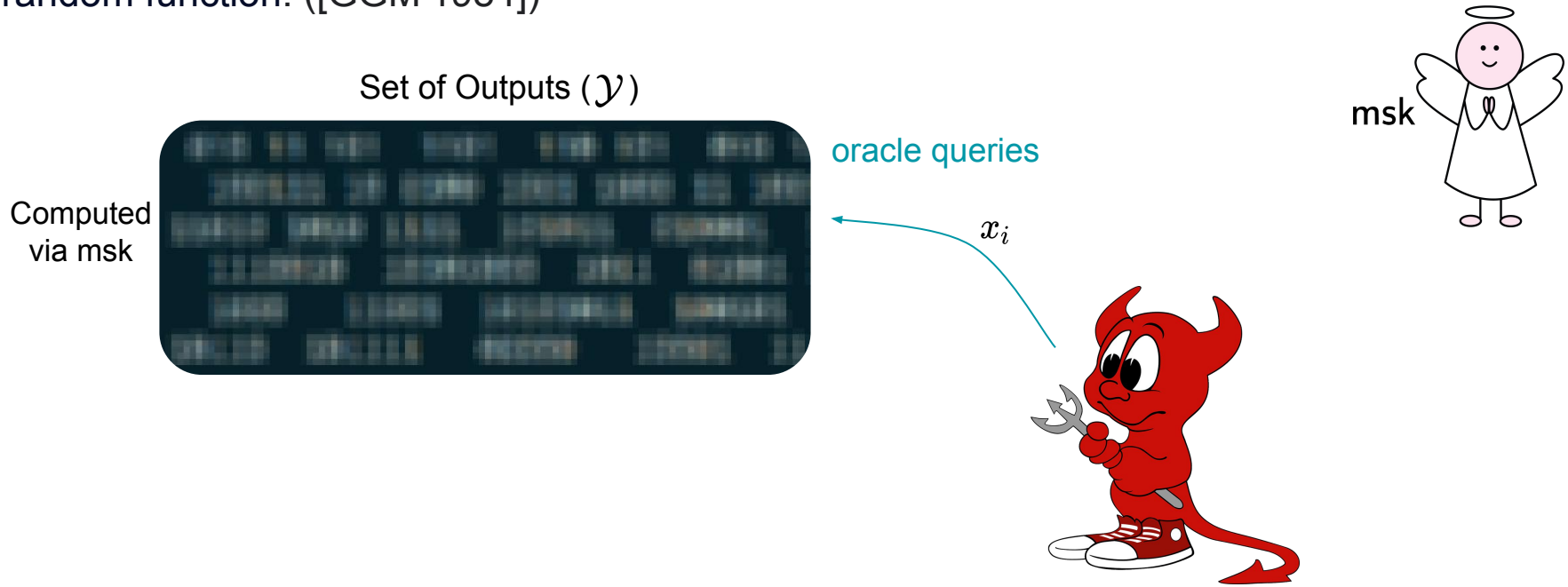
Definition. A deterministic keyed function that is computationally indistinguishable from a truly random function. ([GGM'1984])



Constrained Pseudorandom Function

Pseudorandom function: $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

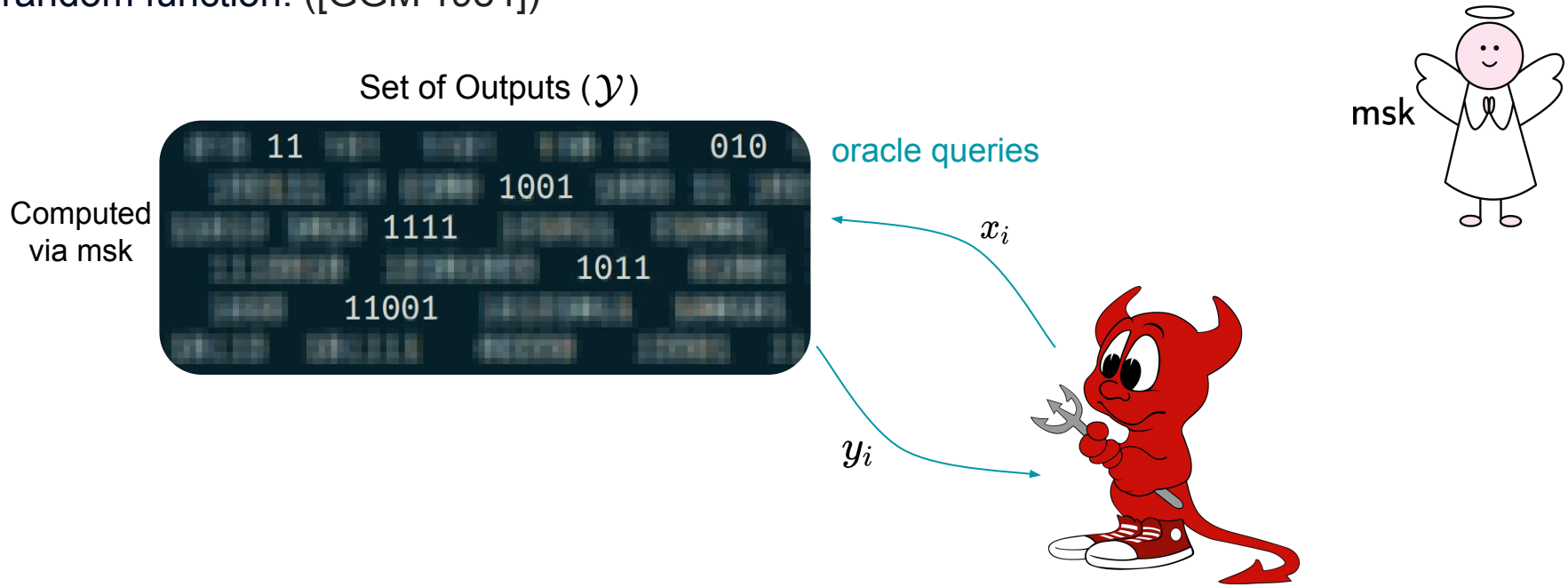
Definition. A deterministic keyed function that is computationally indistinguishable from a truly random function. ([GGM'1984])



Constrained Pseudorandom Function

Pseudorandom function: $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

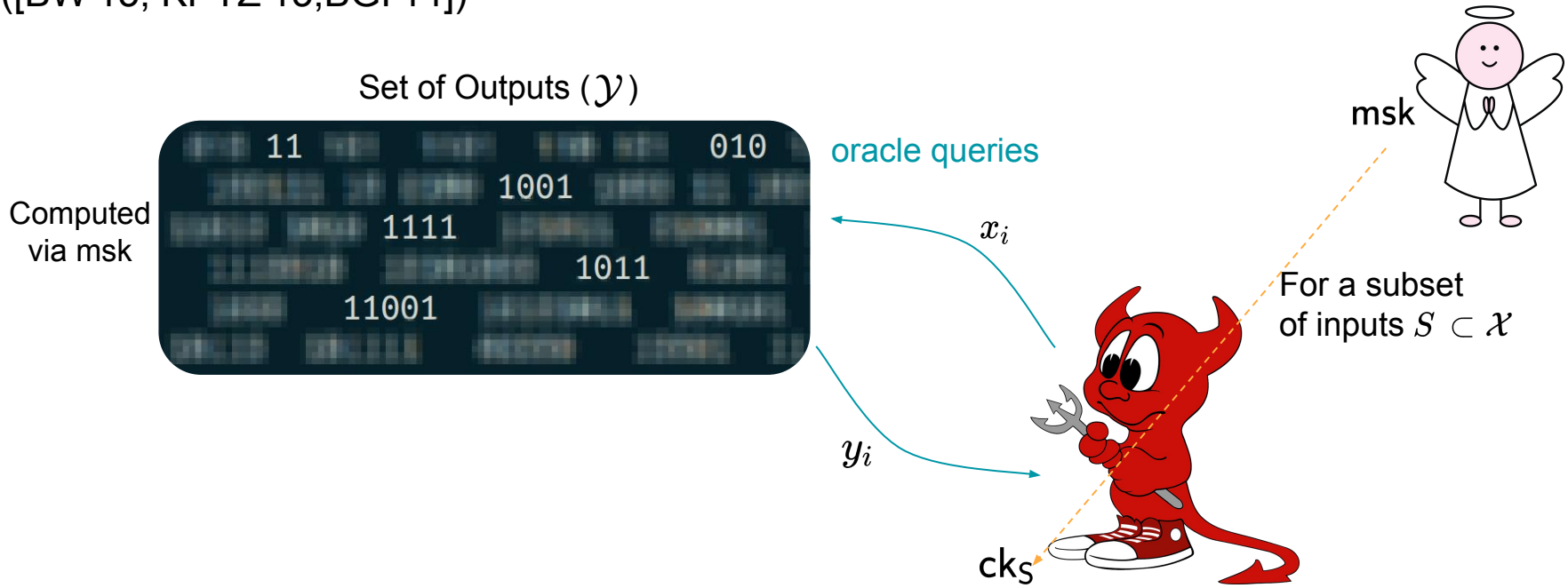
Definition. A deterministic keyed function that is computationally indistinguishable from a truly random function. ([GGM'1984])



Constrained Pseudorandom Function

Constrained *Pseudorandom function:* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

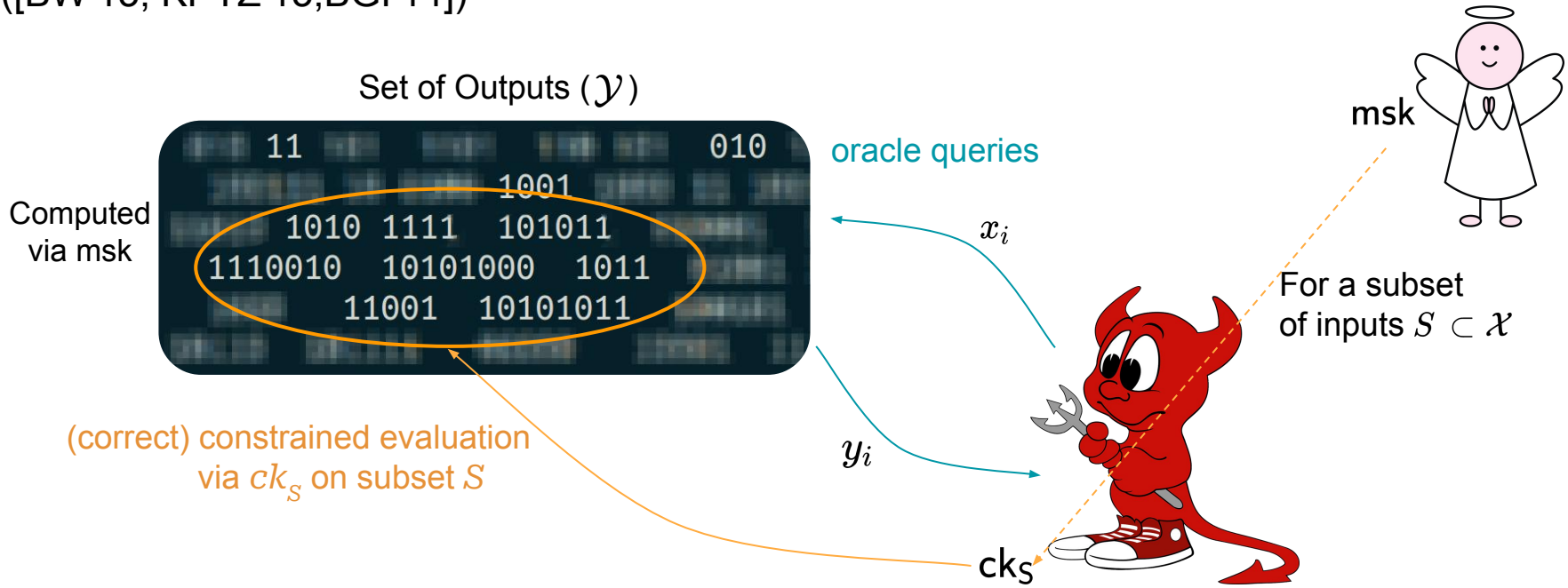
Definition. A pseudorandom function with constrained access to the evaluation.
([BW'13, KPTZ'13,BGI'14])



Constrained Pseudorandom Function

Constrained *Pseudorandom function:* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

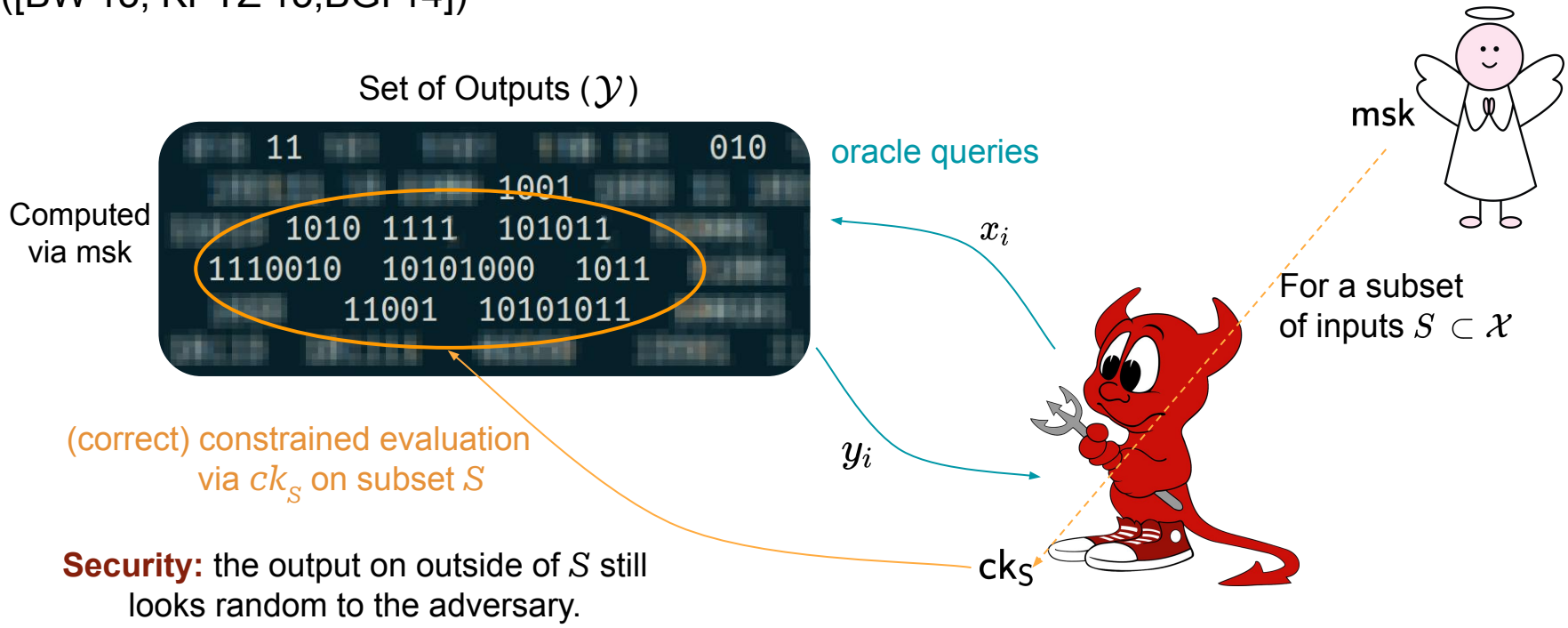
Definition. A pseudorandom function with constrained access to the evaluation.
([BW'13, KPTZ'13,BGI'14])



Constrained Pseudorandom Function

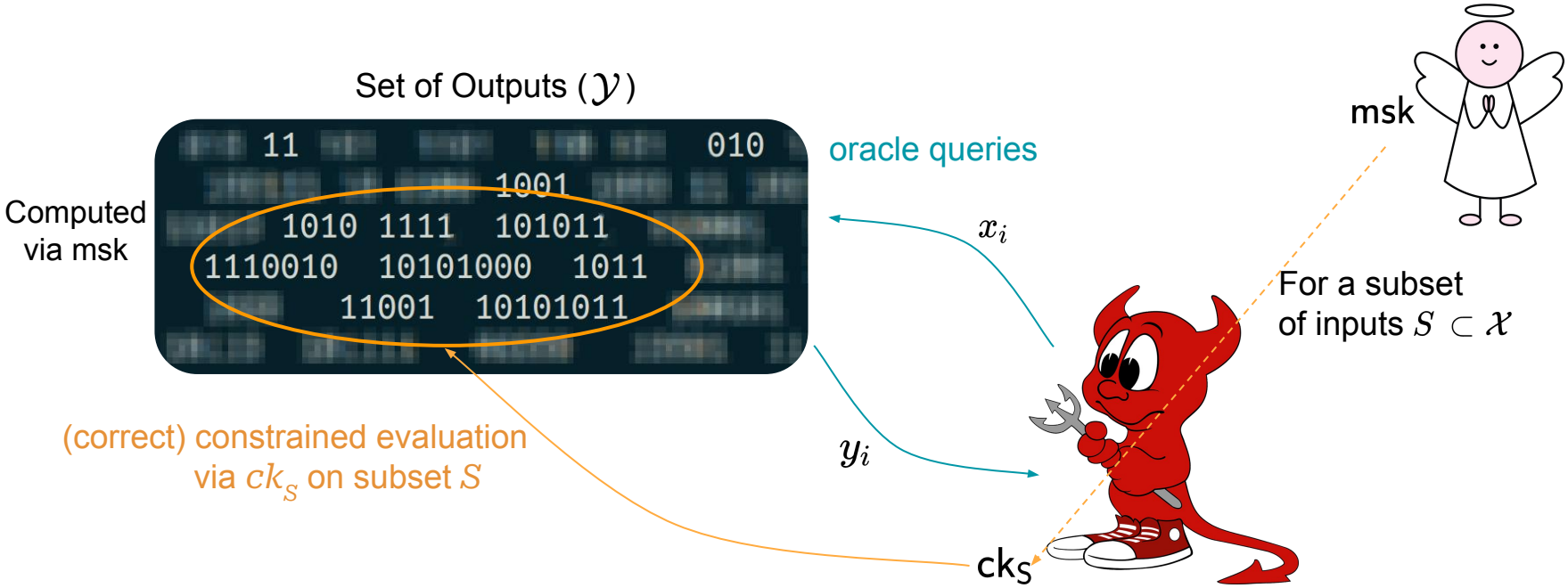
Constrained *Pseudorandom function:* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

Definition. A pseudorandom function with constrained access to the evaluation.
([BW'13, KPTZ'13,BGI'14])



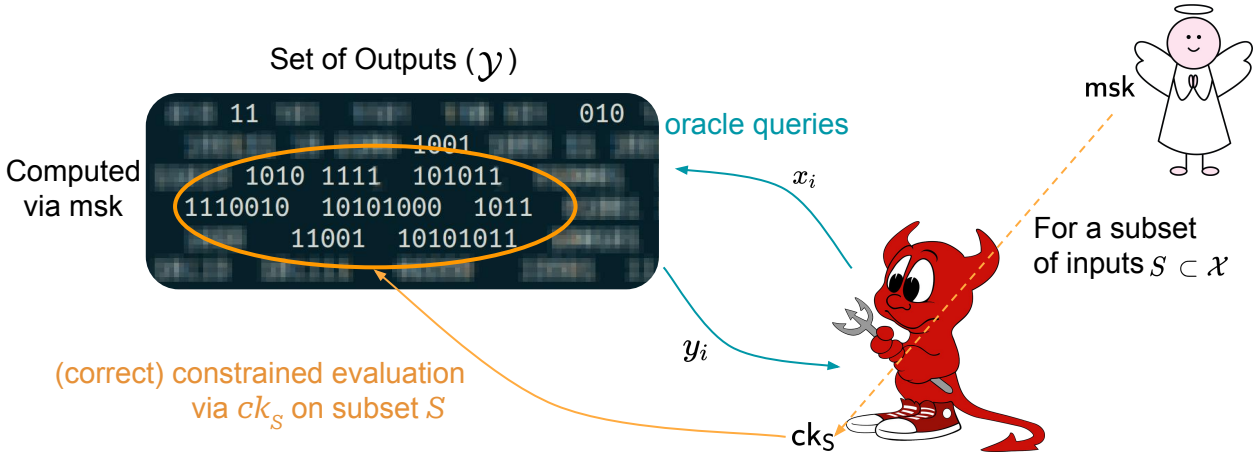
Constrained Pseudorandom Function

Subset S is defined via a predicate $C : \mathcal{X} \to \{0,1\}$; $S_C = \{x \in \mathcal{X} : C(x) = 0\}$



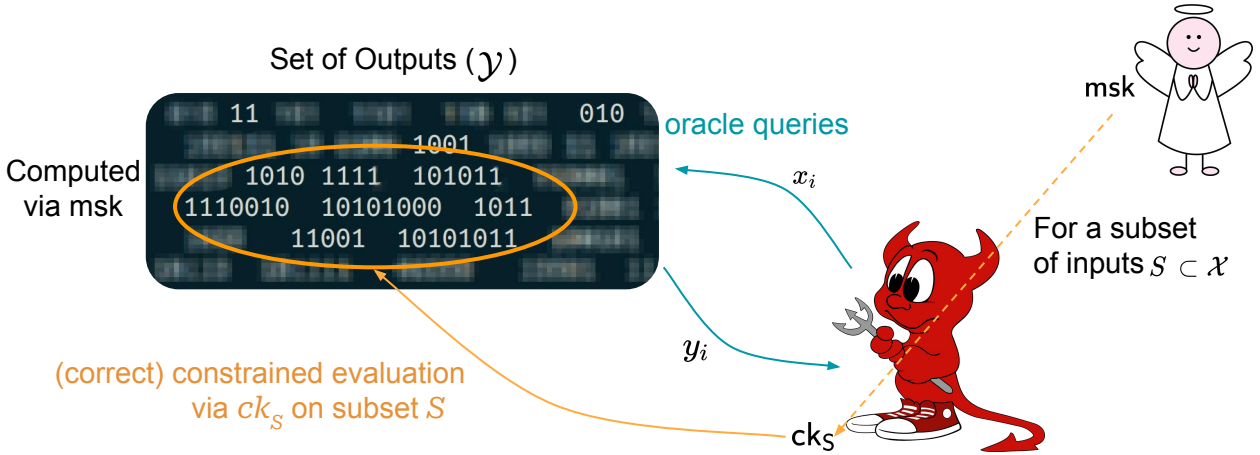
Our contributions

1-key (selectively-secure) constrained PRF for inner-product and NC^1 predicates.



Our contributions

1-key (selectively-secure) constrained PRF for inner-product and NC^1 predicates.



\+ MPC Applications

Homomorphic Secret Sharing

Definition. Protocol for performing distributed evaluation on a secret. ([BGI'16])

Program $P \in \mathcal{P}$

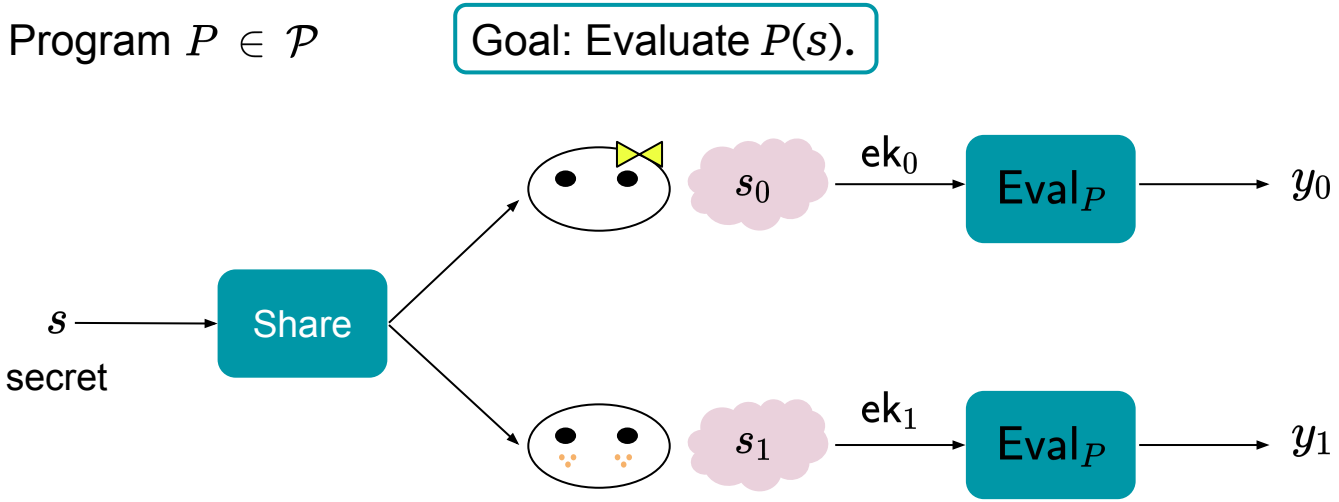
Goal: Evaluate $P(s)$.

s

secret

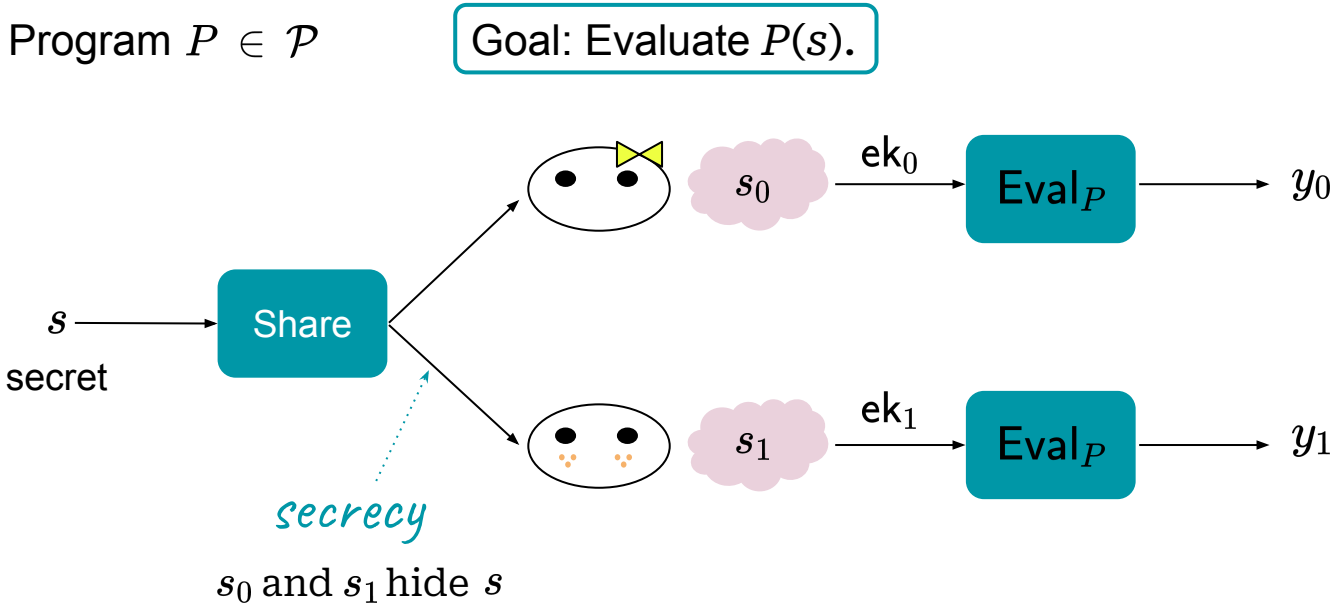
Homomorphic Secret Sharing

Definition. Protocol for performing distributed evaluation on a secret. ([BGI'16])



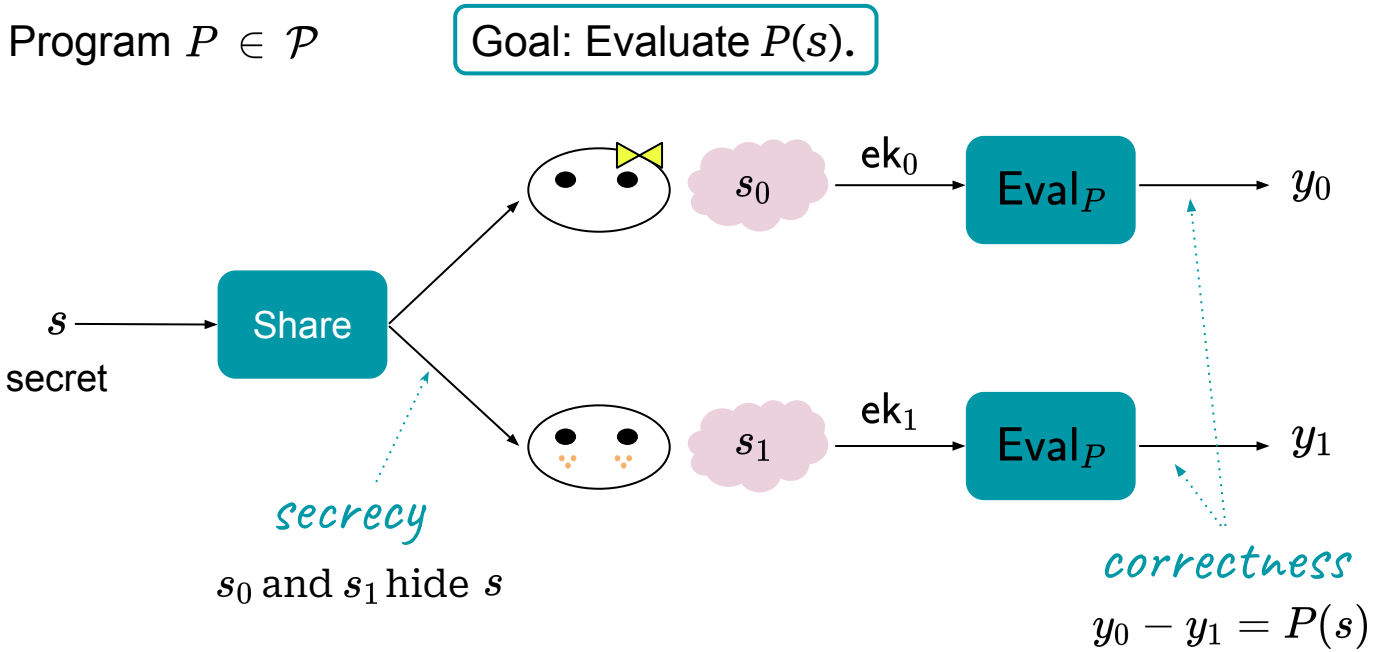
Homomorphic Secret Sharing

Definition. Protocol for performing distributed evaluation on a secret. ([BGI'16])



Homomorphic Secret Sharing

Definition. Protocol for performing distributed evaluation on a secret. ([BGI'16])



Our contributions

1-key constrained PRF for inner-product and NC^1 predicates from homomorphic secret sharing.

- Extending homomorphic secret sharing properties.
- (most of) Existing HSS schemes satisfy these properties.

  ⇝ new constructions of constrained PRF.

- Revisiting Applications of HSS to Secure Computation.
  - Secure computation with silent preprocessing, and
  - 1-sided statistically-secure computation with sublinear communication.

Our contributions

1-key constrained PRF for inner-product and NC^1 predicates from homomorphic secret sharing.

- Extending homomorphic secret sharing properties.
- (most of) Existing HSS schemes satisfy these properties.

  ⇝ new constructions of constrained PRF.

- Revisiting Applications of HSS to Secure Computation.
  - Secure computation with silent preprocessing, and
  - 1-sided statistically-secure computation with sublinear communication.

(1) ***D**ecisional **C**omposite **R**esiduosity*

(2) ***LWE** with superpolynomial modulus*

(3) *Hardness of the **Joye-Libert** encryption scheme*

(4) ***DDH & DXDH** over class groups*

(5) ***H**ard **M**embership **S**ubgroup over class groups*

Constrained PRF from Homomorphic Secret Sharing

General strategy

Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : \ C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Constrained PRF from Homomorphic Secret Sharing

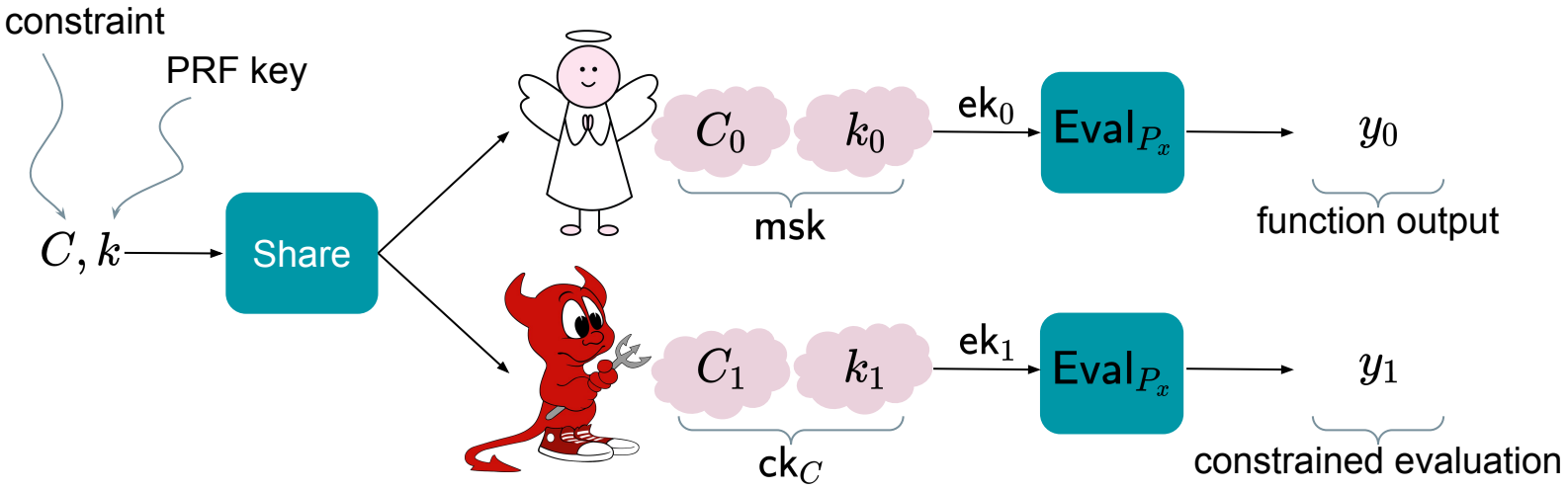
For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : \ C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

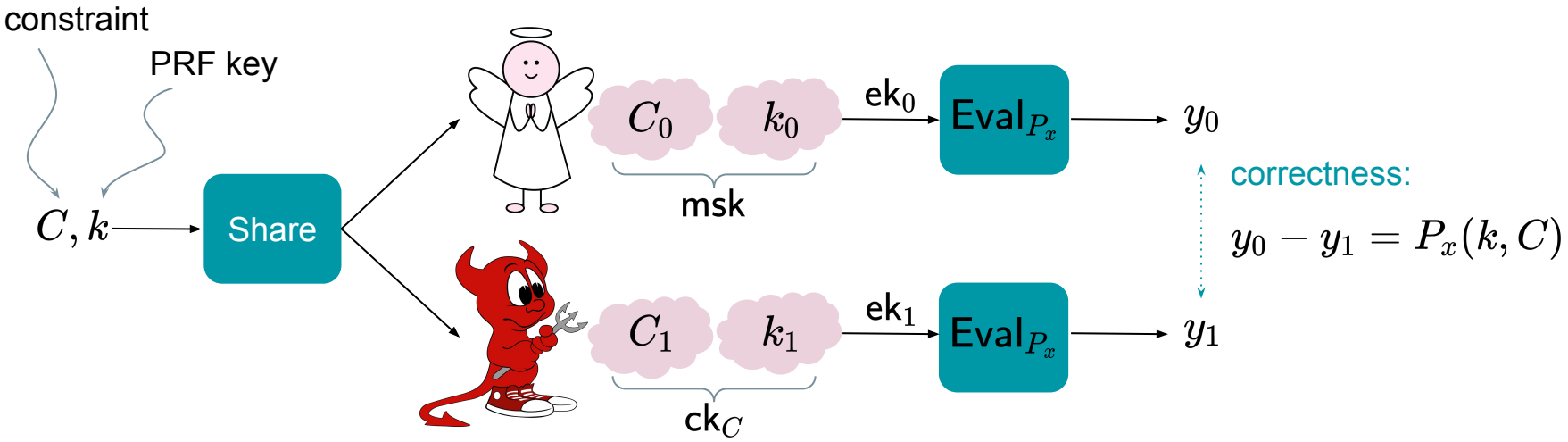
Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.



Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

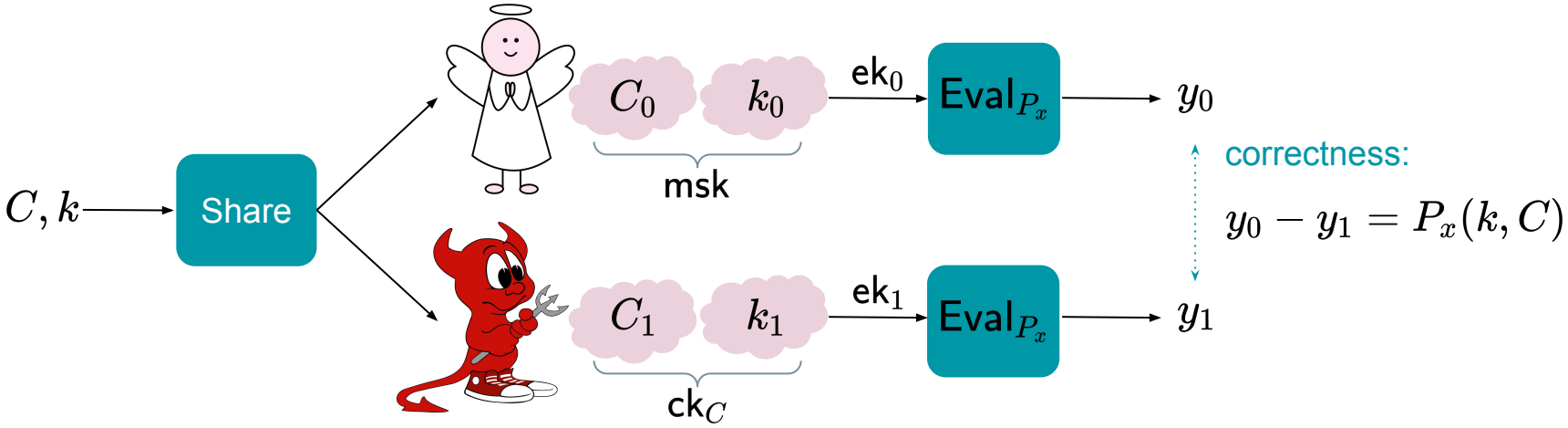


Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

$\Rightarrow x \in S_C \Rightarrow P_x(k, C) = 0$

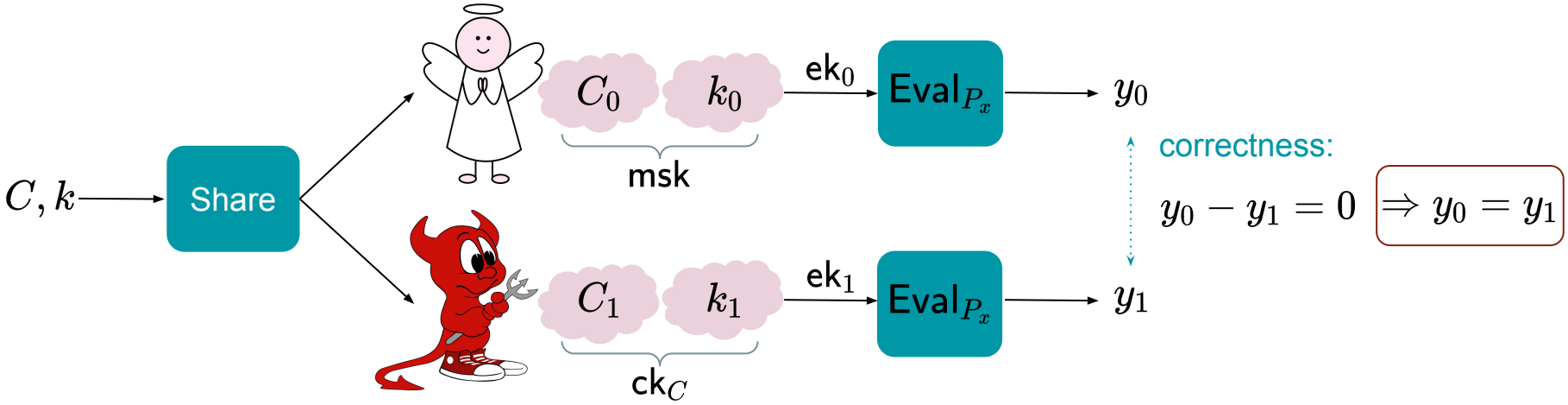


Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

$\Rightarrow x \in S_C \Rightarrow P_x(k, C) = 0$



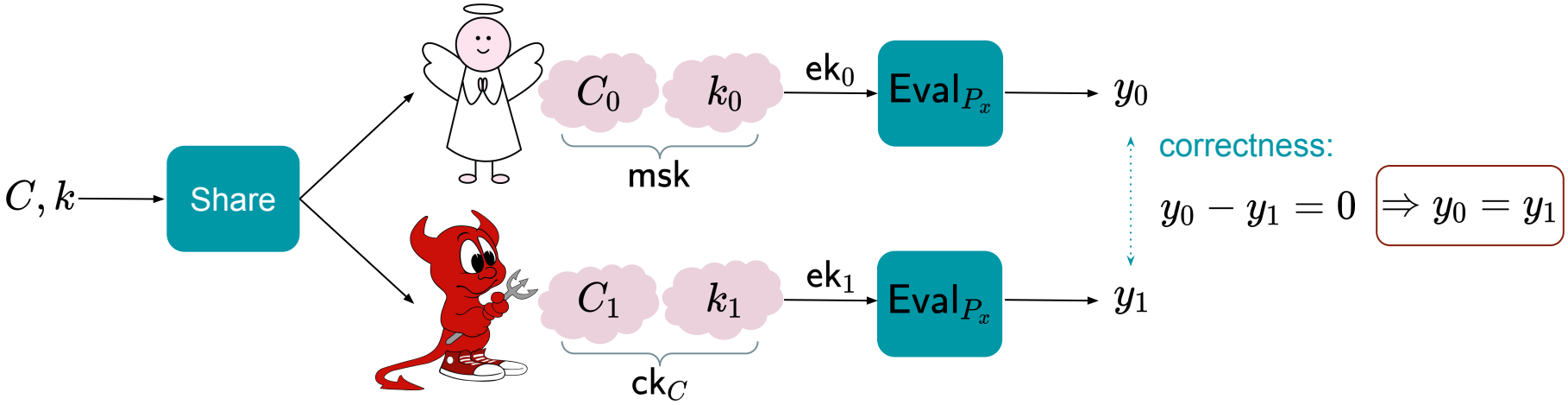
Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

idea

$\Rightarrow x \in S_C \Rightarrow P_x(k, C) = 0$ ~ Equal outputs



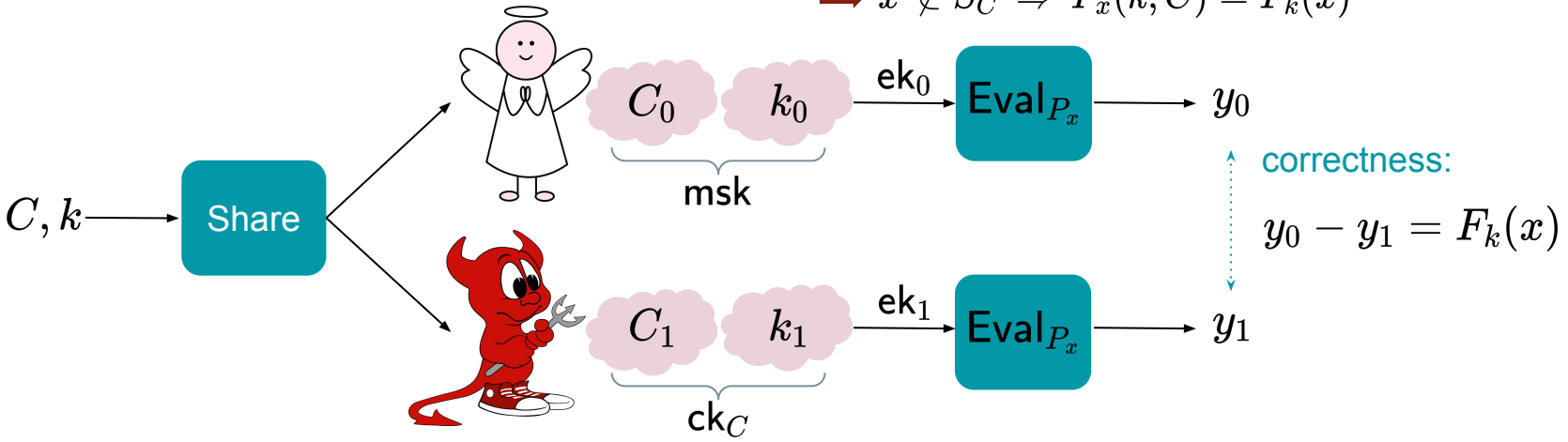
Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

$$x \in S_C \Rightarrow P_x(k, C) = 0 \rightsquigarrow \text{Equal outputs}$$

$$\Rightarrow x \notin S_C \Rightarrow P_x(k, C) = F_k(x)$$



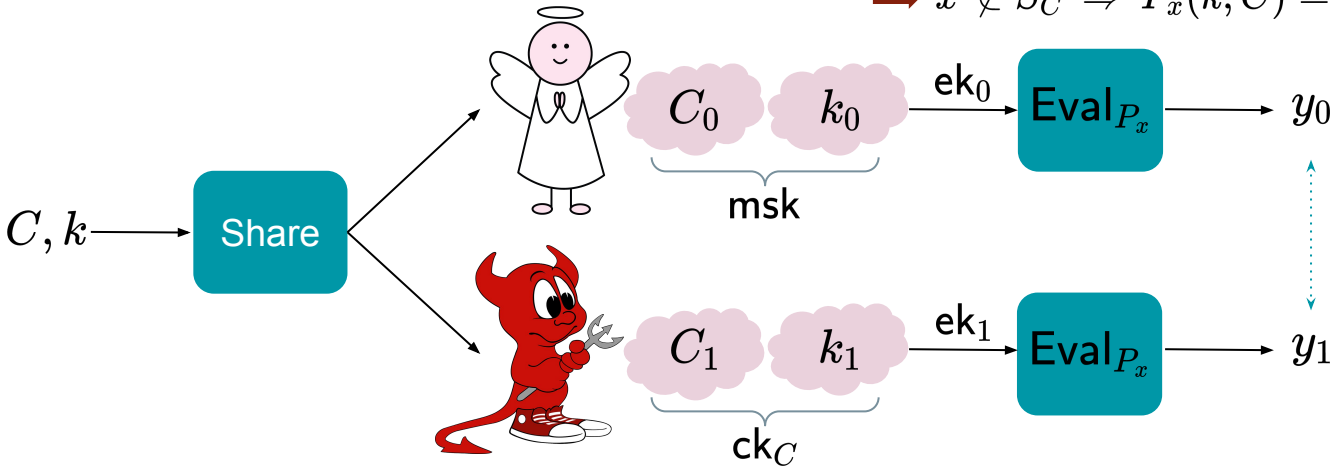
Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

$$x \in S_C \Rightarrow P_x(k, C) = 0$$ ~ Equal outputs

$$\Rightarrow x \notin S_C \Rightarrow P_x(k, C) = F_k(x)$$



correctness:

$$y_0 - y_1 = F_k(x)$$

$$\Rightarrow y_0 = y_1 + F_k(x)$$

looks random
(k is hidden)

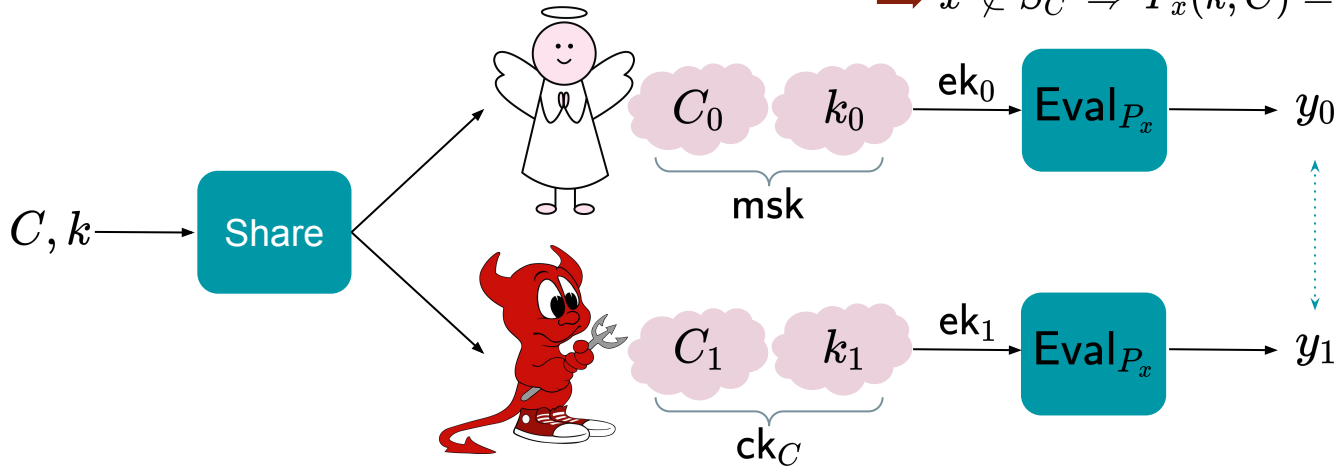
Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.

$x \in S_C \Rightarrow P_x(k, C) = 0$ ⇝ Equal outputs

$x \notin S_C \Rightarrow P_x(k, C) = F_k(x)$ ⇝ Random-looking function output



correctness:

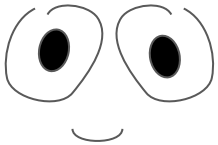
$y_0 - y_1 = F_k(x)$

$\Rightarrow y_0 = y_1 + F_k(x)$
looks random
(k is hidden)

Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it. ✓

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.



$$\begin{cases} x \in S_C \Rightarrow P_x(k,C) = 0 \rightsquigarrow \text{Equal outputs} \\ x \notin S_C \Rightarrow P_x(k,C) = F_k(x) \rightsquigarrow \text{Random-looking function output} \end{cases}$$

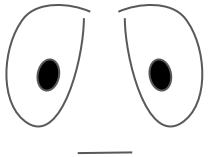
C, k → Share → C_0 k_0 (msk) → ek_0 → Eval_{P_x} → y_0 (function output)

Share → C_1 k_1 (ck_C) → ek_1 → Eval_{P_x} → y_1 (constrained evaluation)

Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.



msk depends on C

C_0 k_0 → ek_0 → Eval_{P_x} → y_0

msk

function output

C, k → Share

C_1 k_1 → ek_1 → Eval_{P_x} → y_1

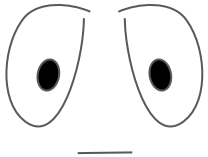
ck_C

constrained evaluation

Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.

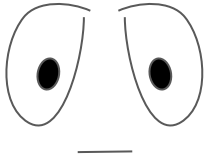
Take a PRF F with key k, and use an HSS to compute $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$.



msk depends on C

Constrained PRF from Homomorphic Secret Sharing

For a constraint $C: \mathcal{X} \to \{0,1\}$: $S_C = \{x \in \mathcal{X} : C(x) = 0\}$
The adversary can evaluate on S_C, while learning nothing about the output outside of it.



msk depends on C

Question

How to generate C_0 before knowing C?

C, k → Share

C_0 k_0 → msk → ek_0 → Eval_{P_x} → y_0 → function output

C_1 k_1 → ck_C → ek_1 → Eval_{P_x} → y_1 → constrained evaluation

Constrained PRF
from
Homomorphic Secret Sharing

What really happens!

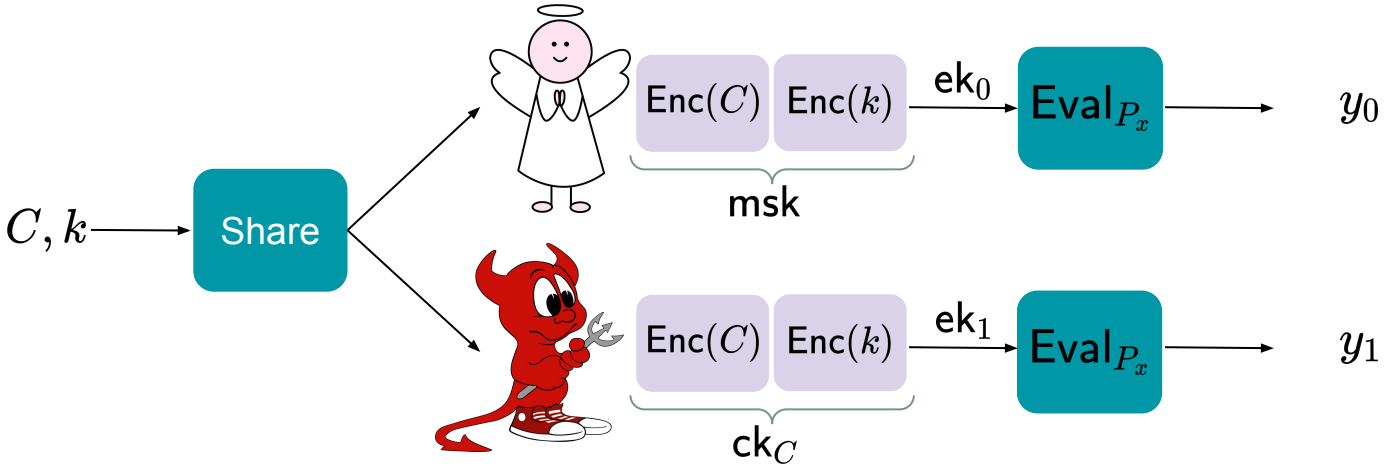
Homomorphic Secret Sharing supporting $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$

Homomorphic Secret Sharing supporting NC^1 programs

Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

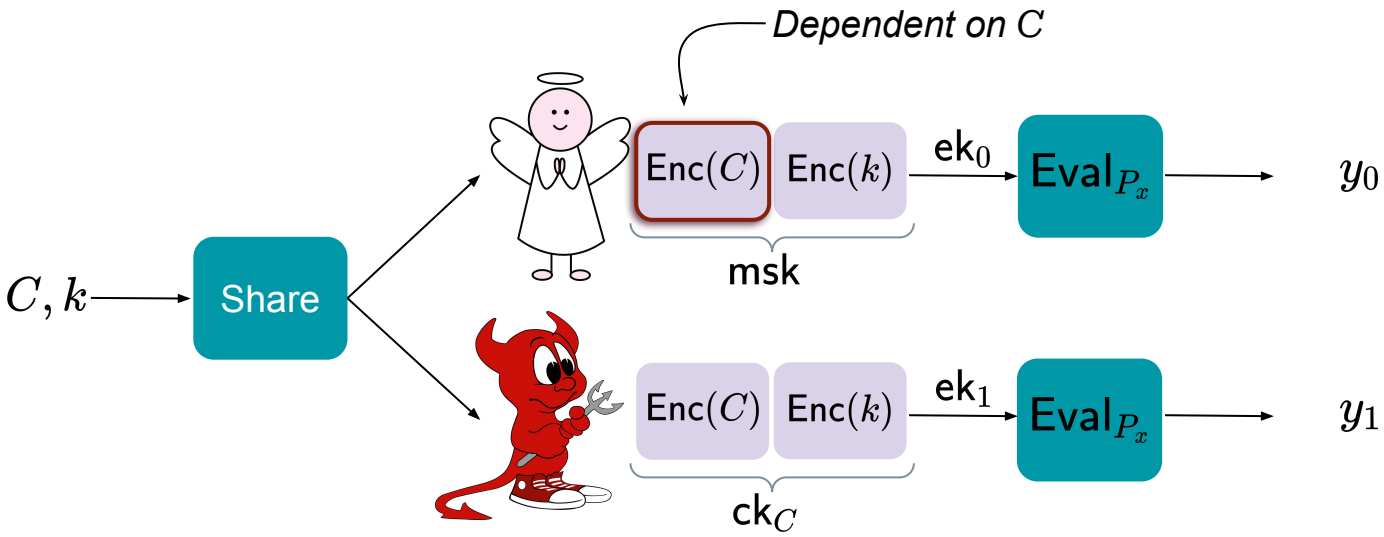
Shares: Encryptions



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

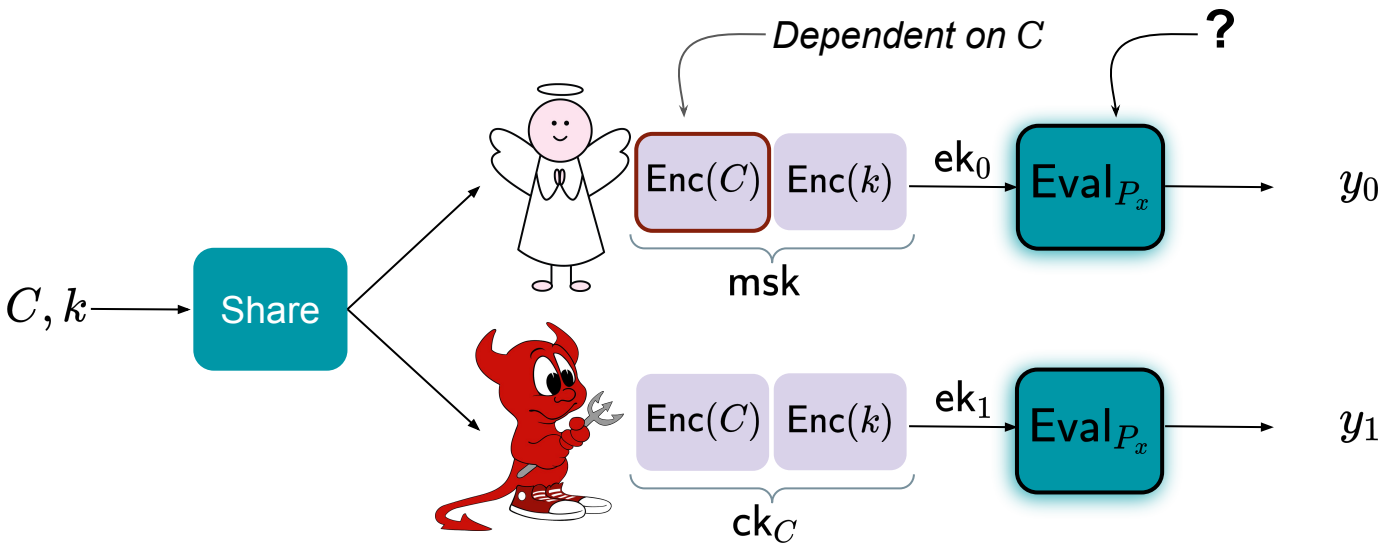
Shares: Encryptions



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

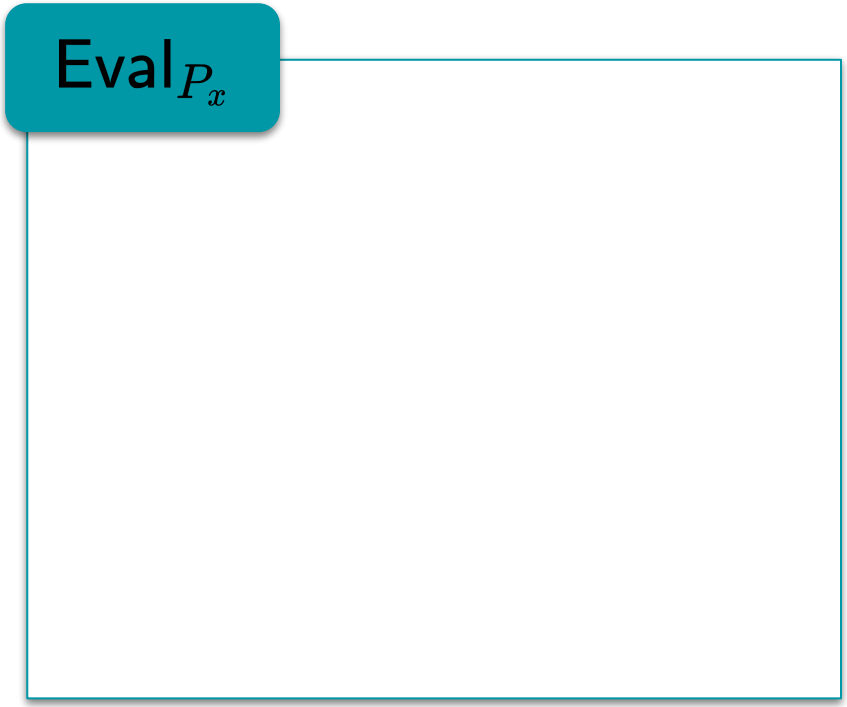
Shares: Encryptions



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

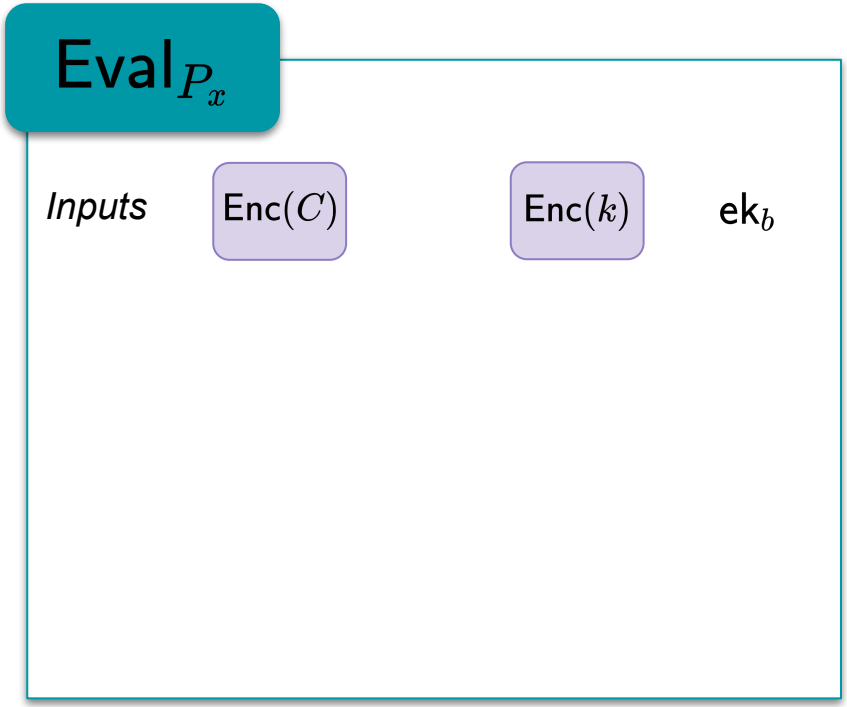
$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

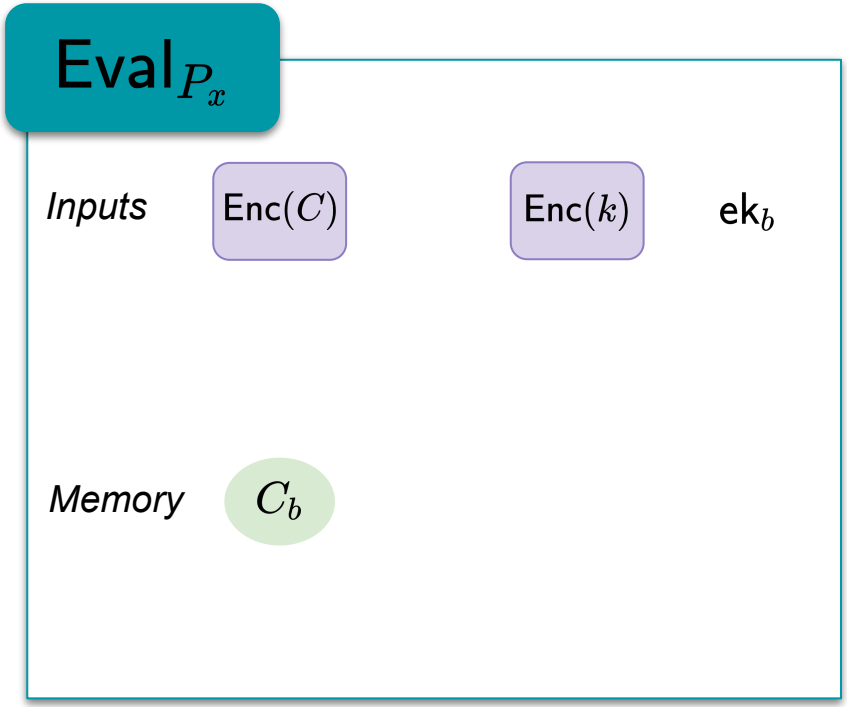
$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

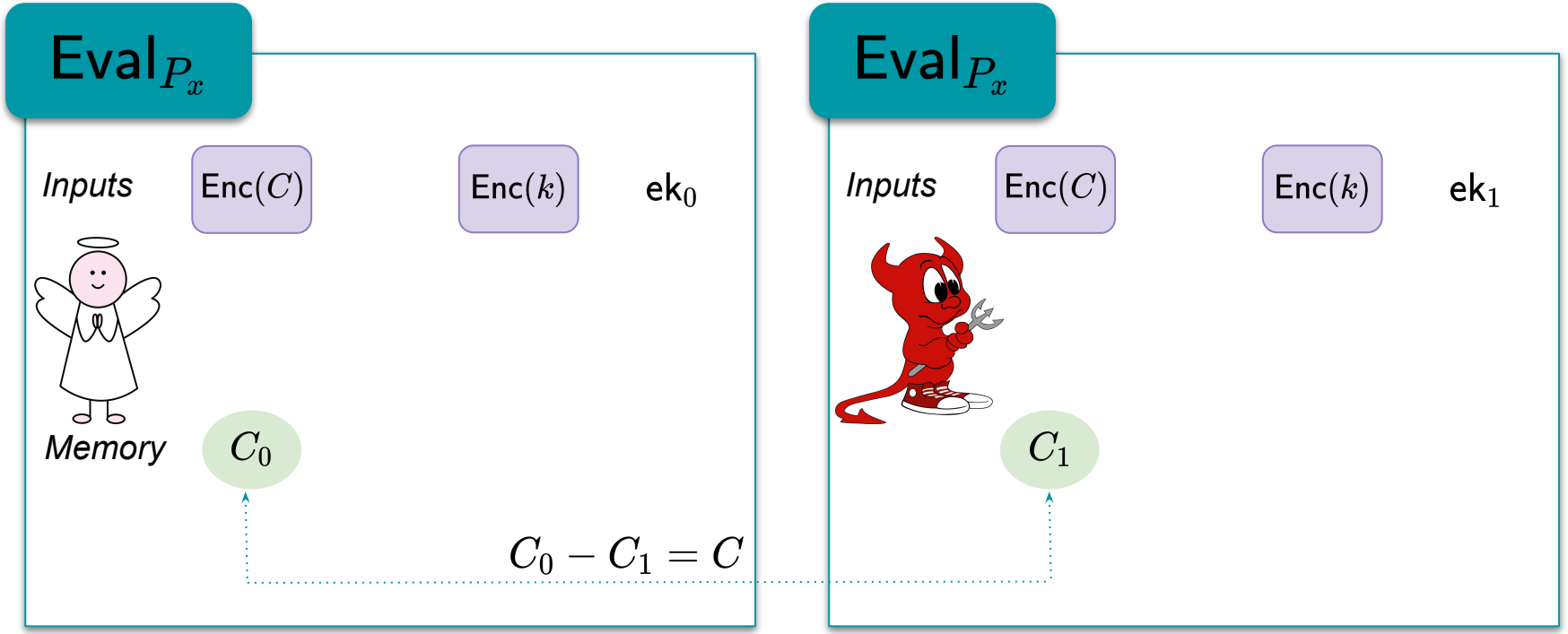
$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$

Allowed operation: Multiply(E(m), n_b) = $(mn)_b$

(Restricted Multiplication)

Eval$_{P_x}$

Inputs Enc(C) Enc(k) ek_0

C_0

Eval$_{P_x}$

Inputs Enc(C) Enc(k) ek_1

C_1

Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$

Allowed operation: Multiply($\mathsf{E}(m)$, 1_b) = m_b

Eval$_{P_x}$

Inputs Enc(C) Enc(k) ek_0

C_0

Eval$_{P_x}$

Inputs Enc(C) Enc(k) ek_1

C_1

Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

Loading inputs into memory

$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$

Allowed operation: Multiply($\mathsf{E}(m)$, 1_b) = m_b

Eval_{P_x}

Inputs $\mathsf{Enc}(C)$ $\mathsf{Enc}(k)$ ek_0

C_0

Eval_{P_x}

Inputs $\mathsf{Enc}(C)$ $\mathsf{Enc}(k)$ ek_1

C_1

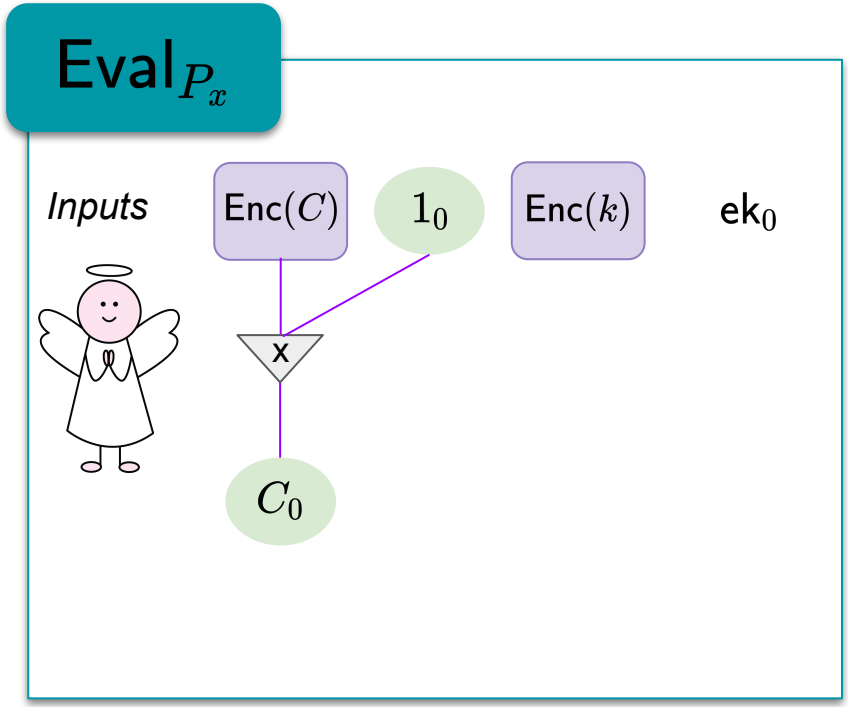
Homomorphic Secret Sharing supporting NC^1 programs

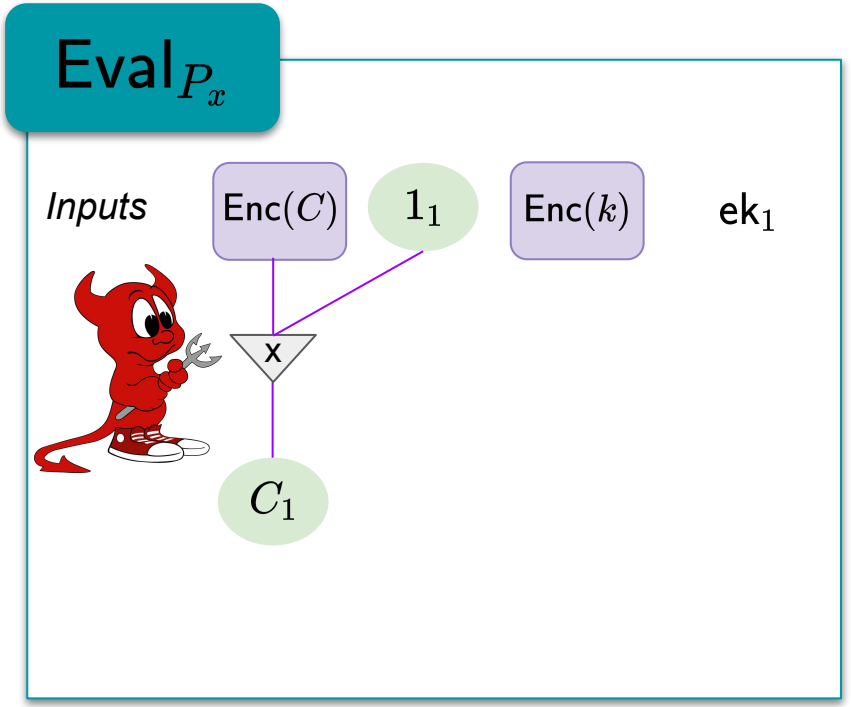
Using (additively homomorphic) public-key encryption scheme.

Loading inputs into memory

$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$

Allowed operation: Multiply(E(m), 1_b) = m_b





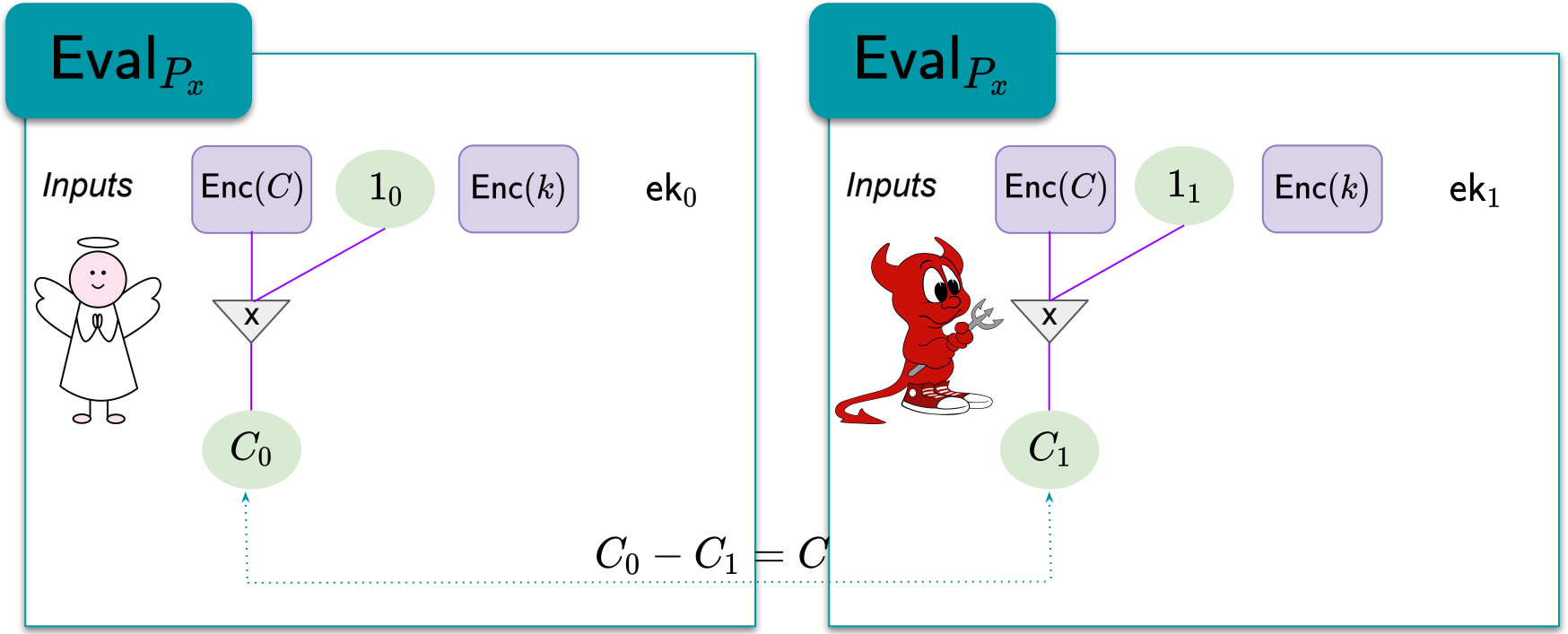
Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

Loading inputs into memory

$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$

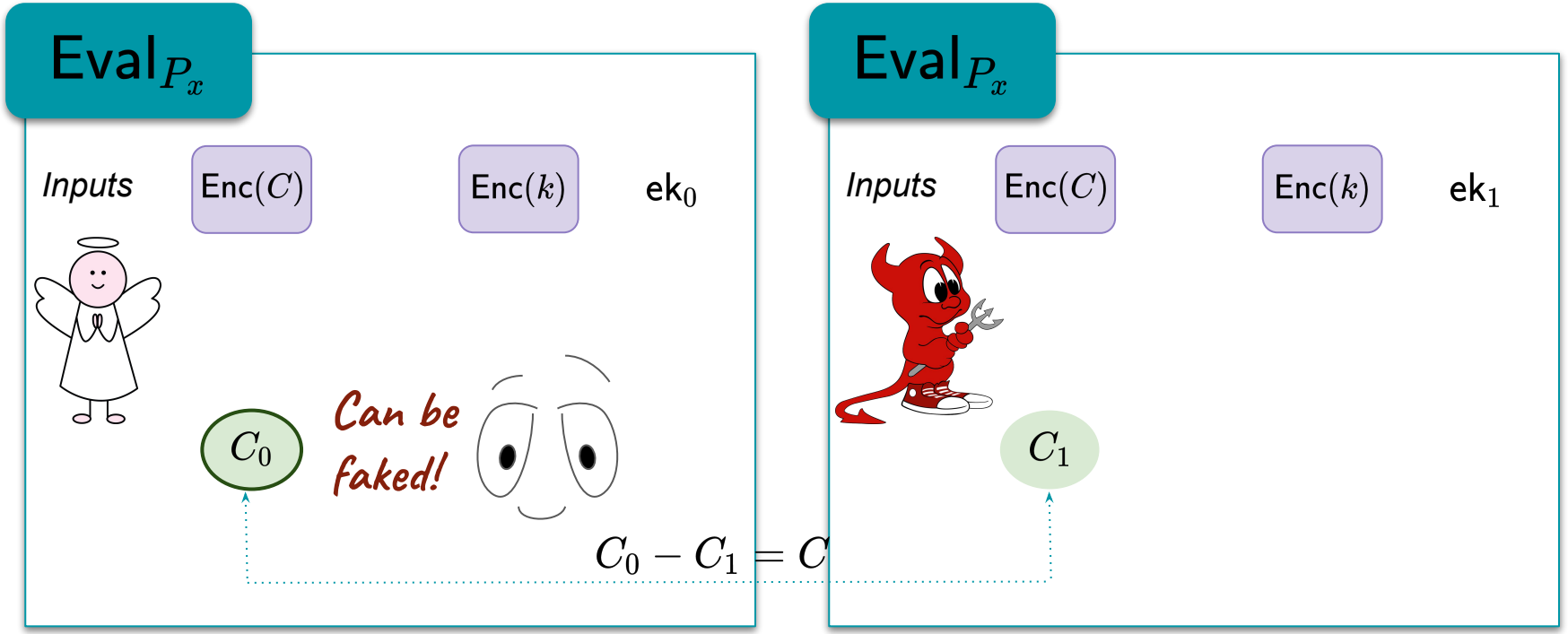
Allowed operation: Multiply($\mathsf{E}(m)$, 1_b) = m_b



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$

Eval$_{P_x}$

before knowing C

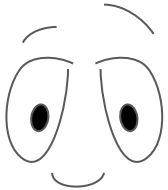
Inputs

Enc(k)

ek$_0$

C_0

Can be faked!



Eval$_{P_x}$

Inputs

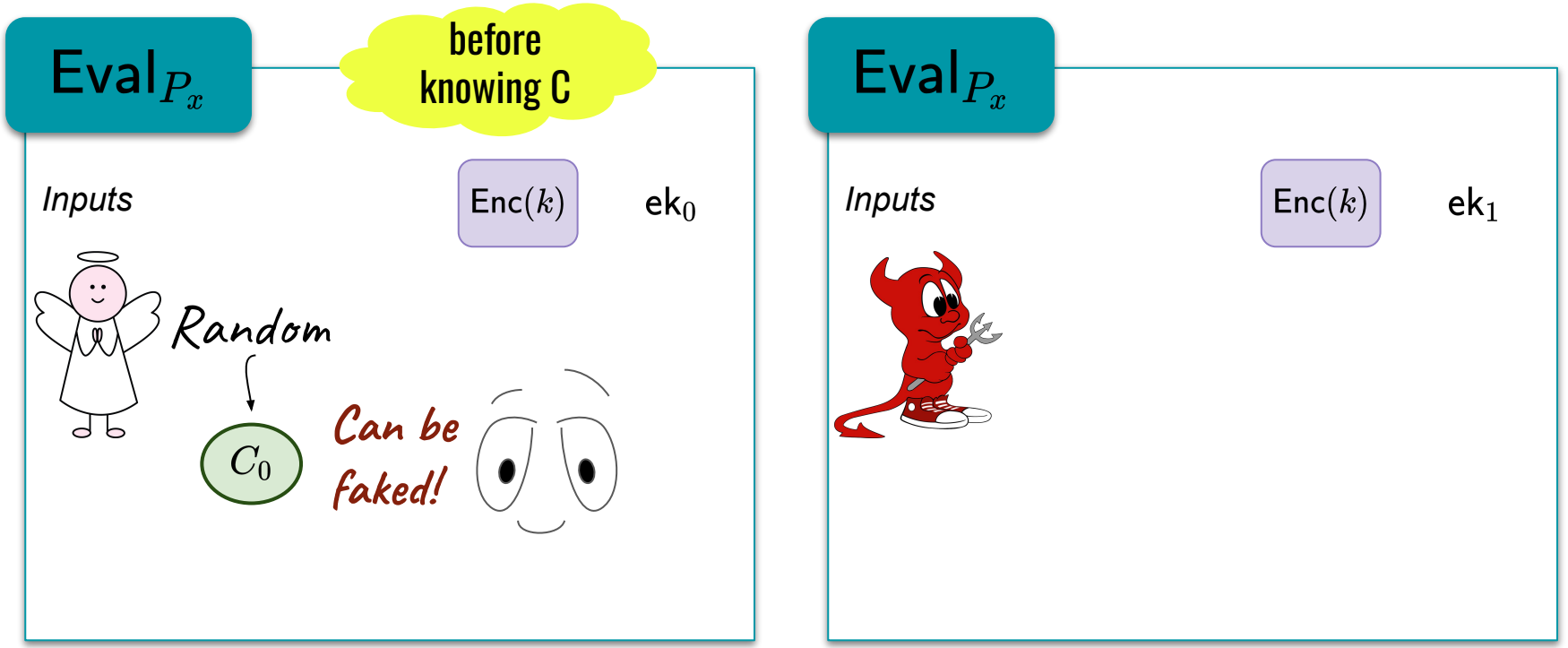
Enc(k)

ek$_1$

Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

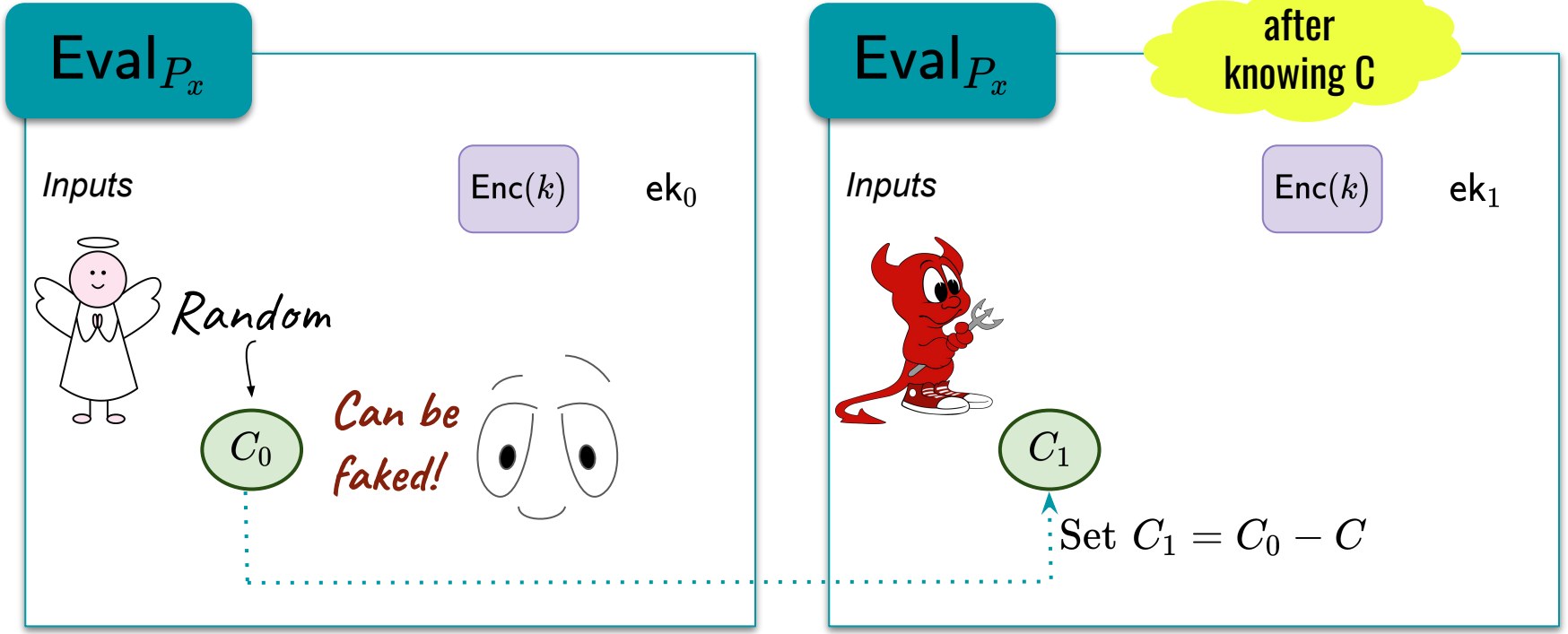
$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

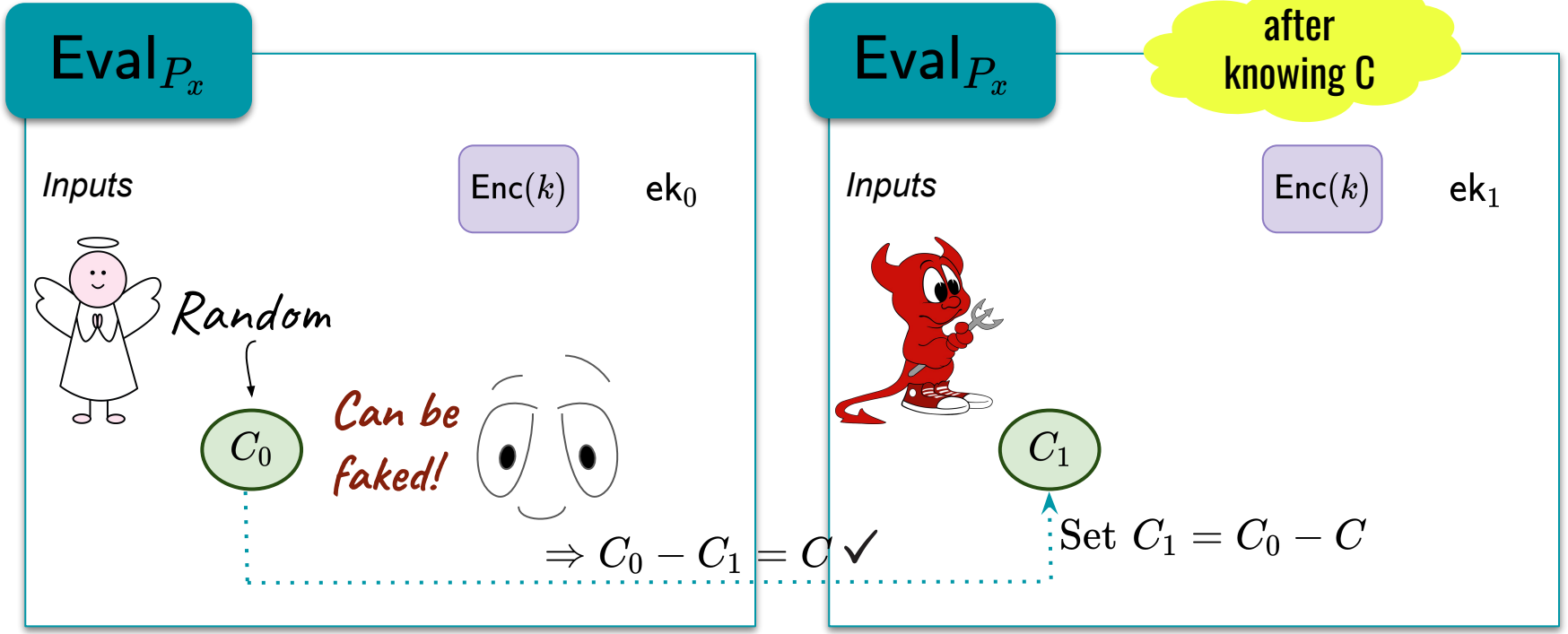
$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

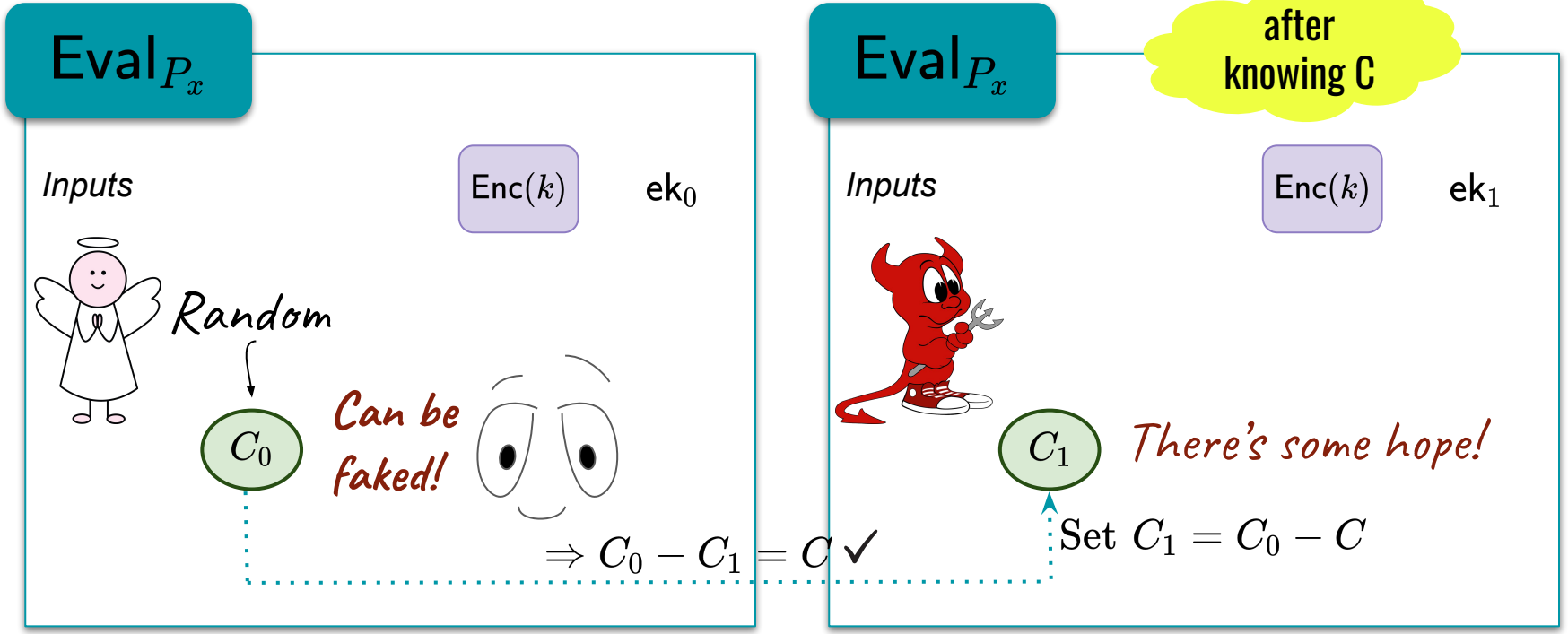
$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Homomorphic Secret Sharing supporting NC^1 programs

Using (additively homomorphic) public-key encryption scheme.

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$



Constrained PRF from Homomorphic Secret Sharing

For Inner-Product.

Constrained PRF for Inner-Product constraint

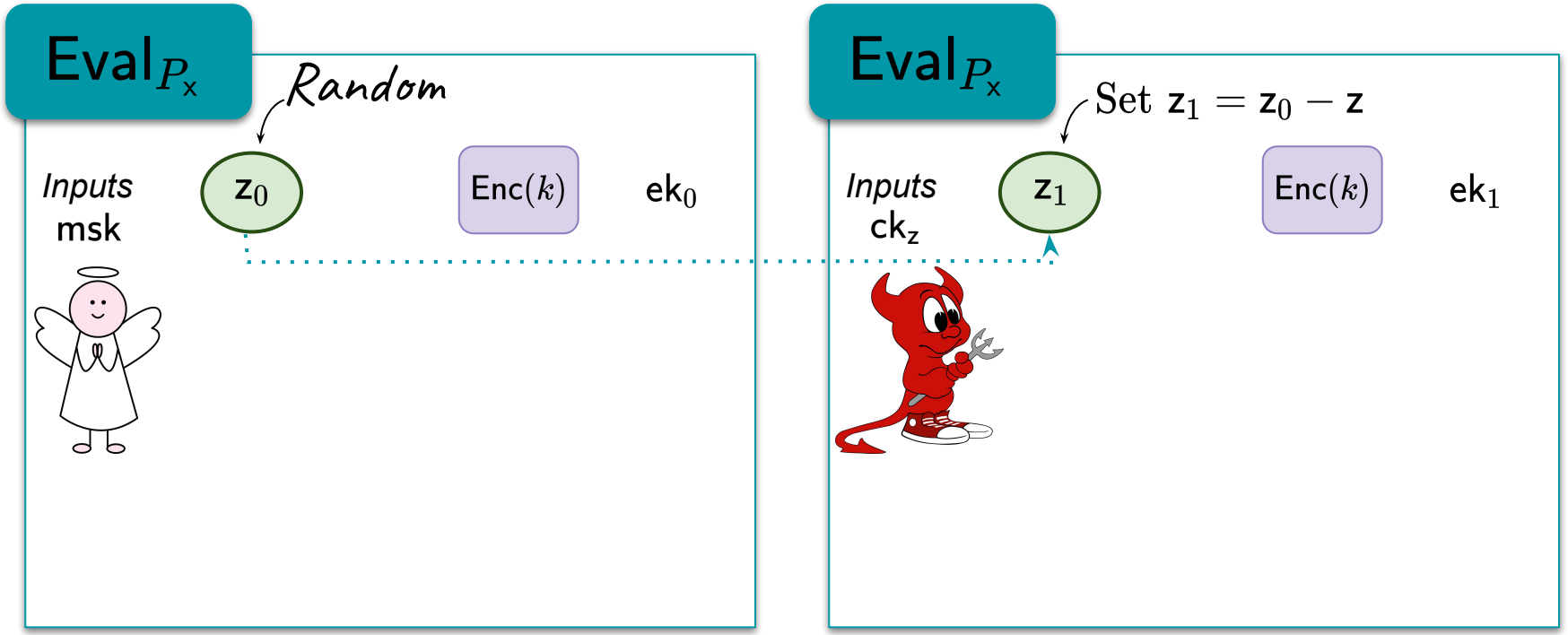
$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**. Adversary can compute on **x** iff < **z**,**x** > =0.

Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

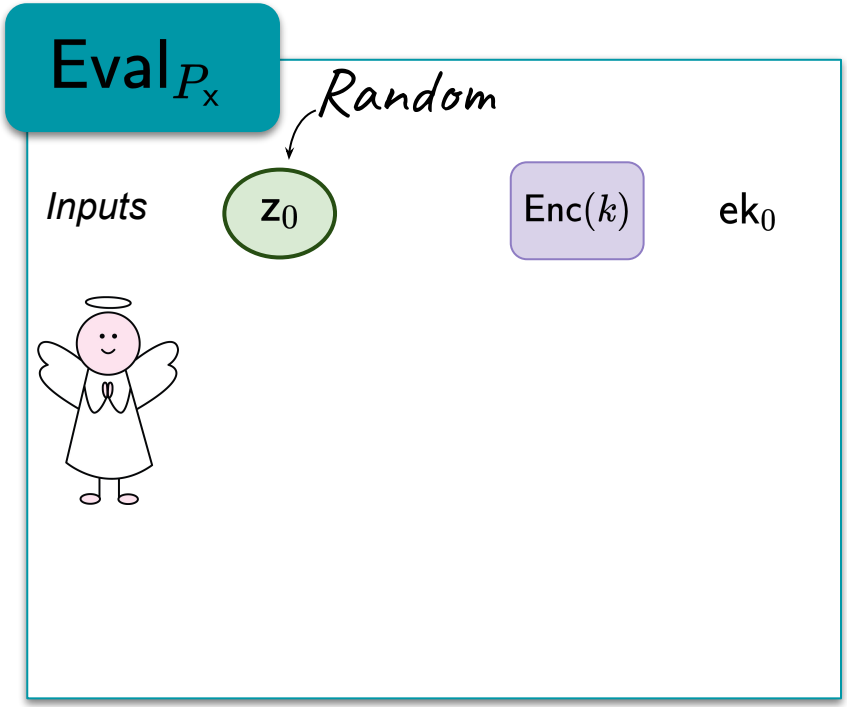


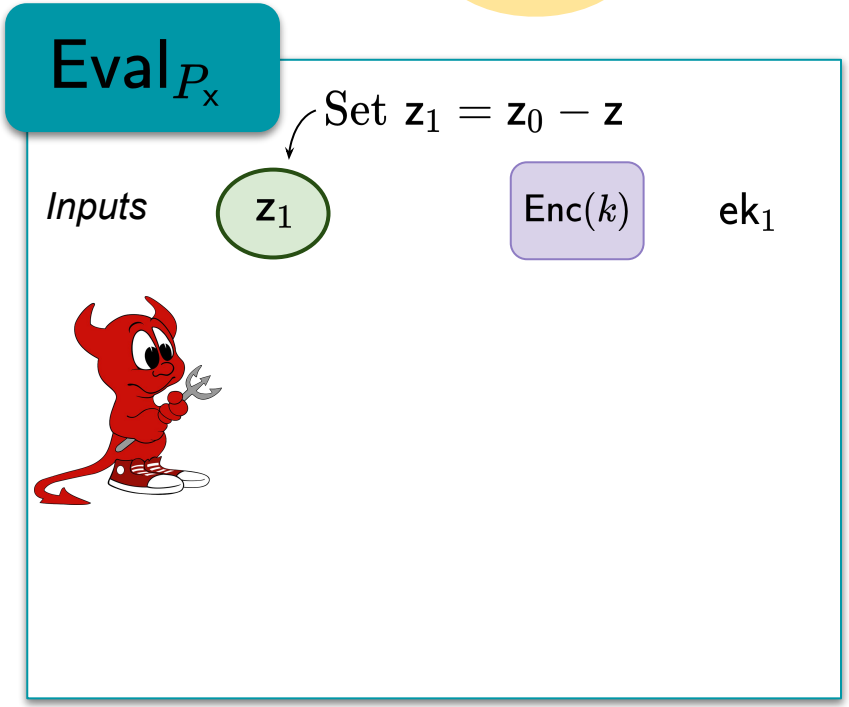
Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

To load an input into memory:

Multiply by z_b





Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

To load an input into memory:

Multiply by z_b

ExtEval$_{P_{\mathsf{x}}}$

Random

Inputs z_0 $\mathsf{Enc}(k)$ ek_0

ExtEval$_{P_{\mathsf{x}}}$

Set $\mathsf{z}_1 = \mathsf{z}_0 - \mathsf{z}$

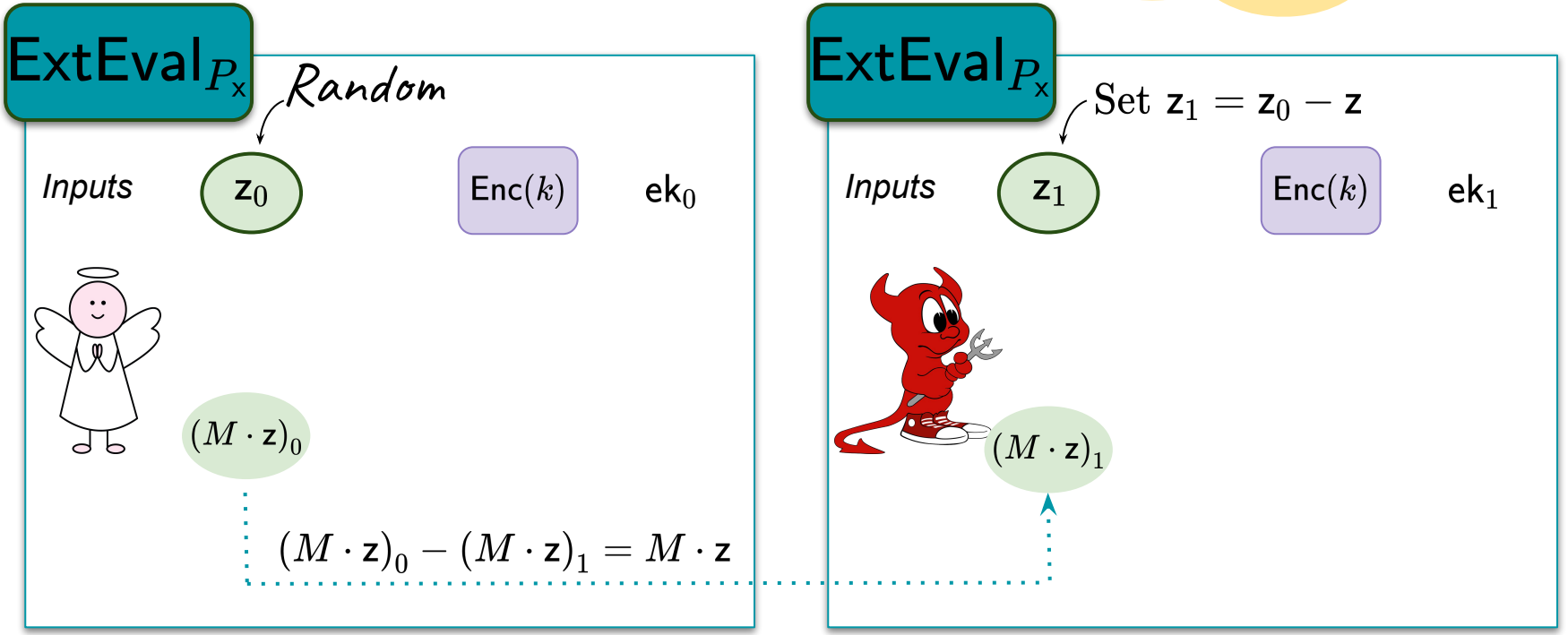
Inputs z_1 $\mathsf{Enc}(k)$ ek_1

Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

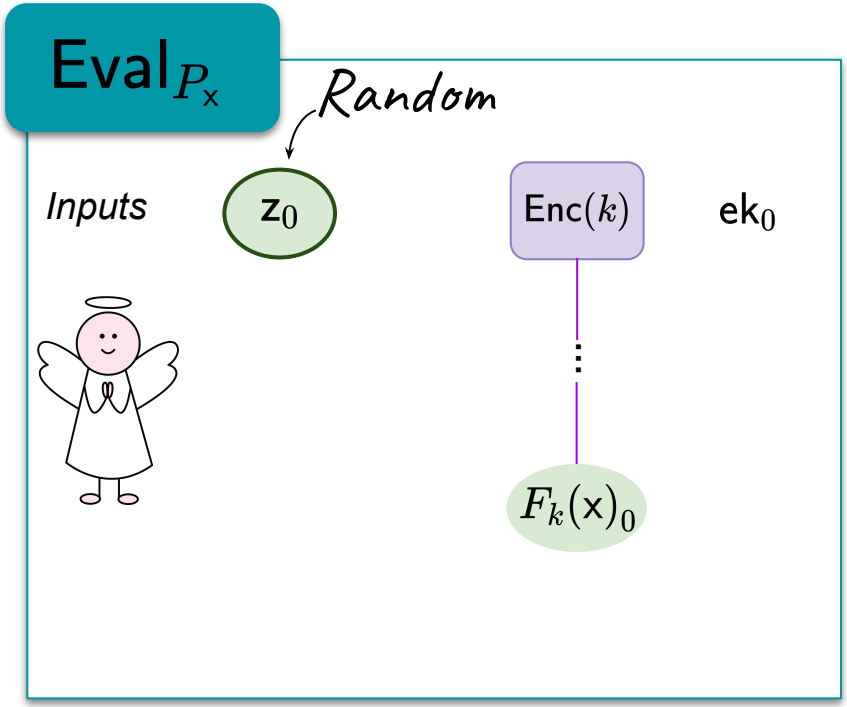
To load an input into memory:

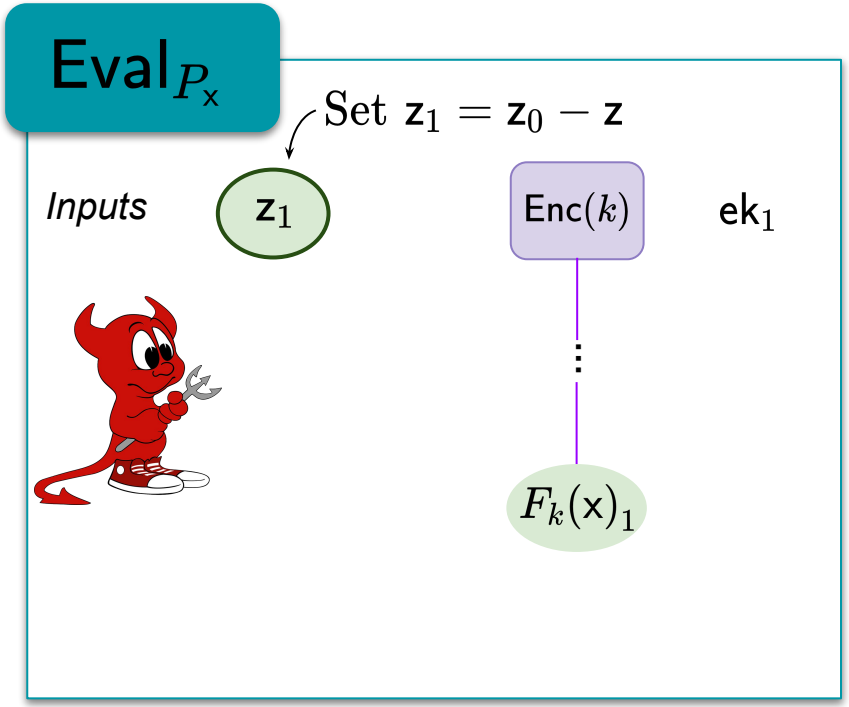
Multiply by z_b



Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.



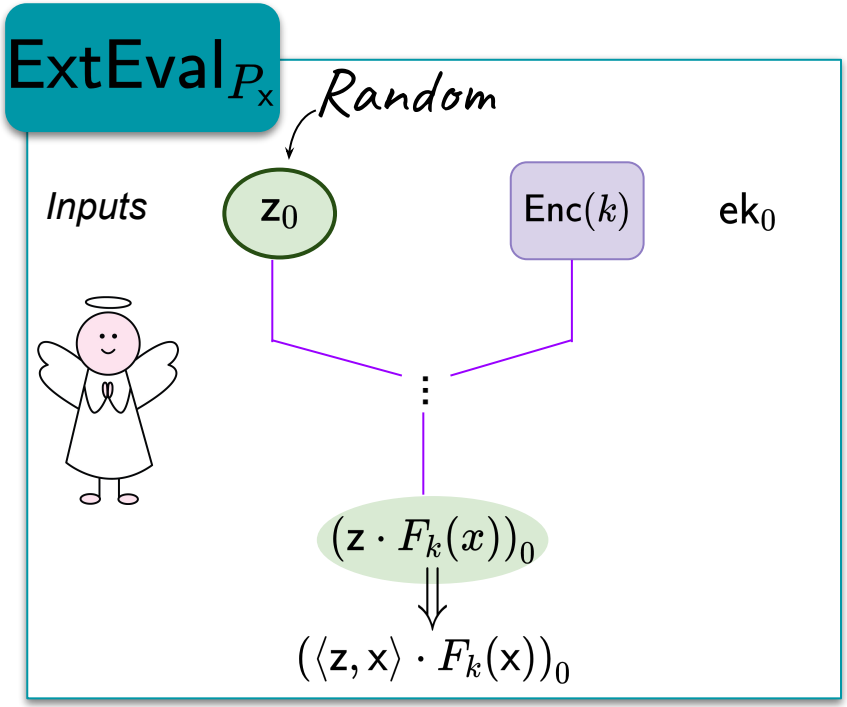


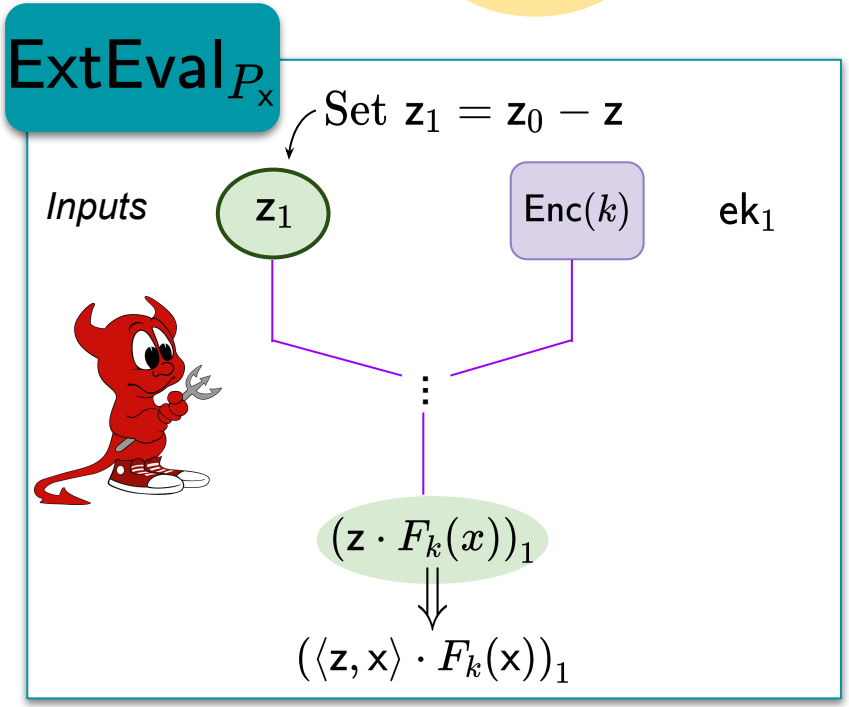
Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

To load an input into memory:

Multiply by z_b



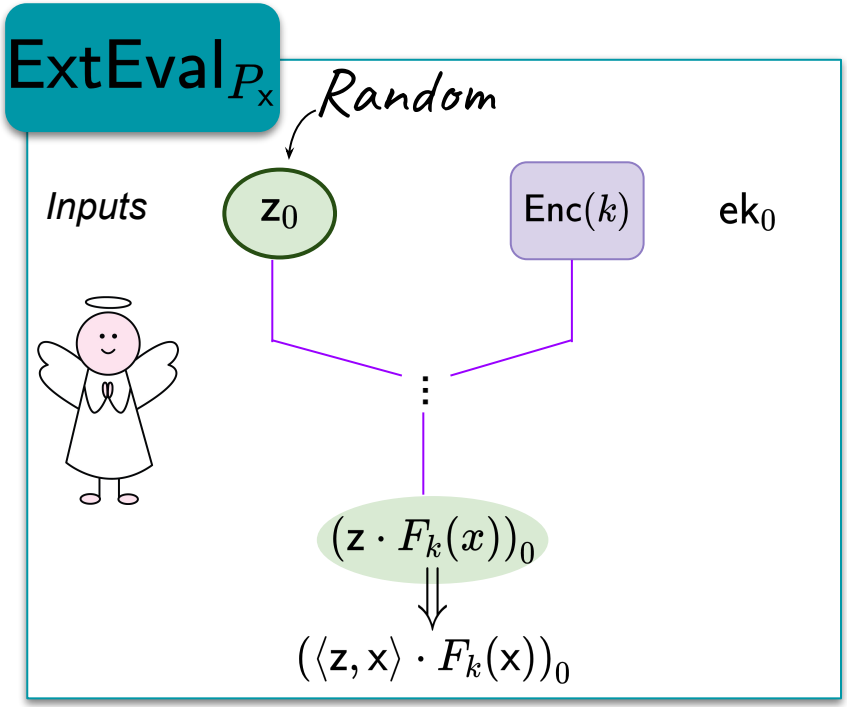


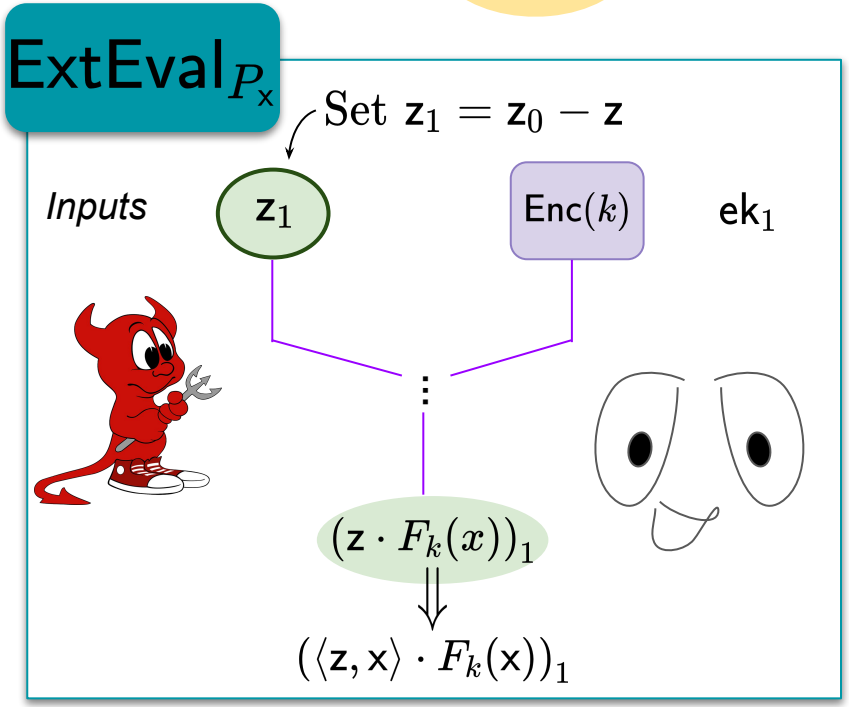
Constrained PRF for Inner-Product constraint

$P_{\mathsf{x}} : (k, \mathsf{z}) \mapsto \langle \mathsf{z}, \mathsf{x} \rangle \cdot F_k(\mathsf{x})$ for a vector **z**.

To load an input into memory:

Multiply by z_b

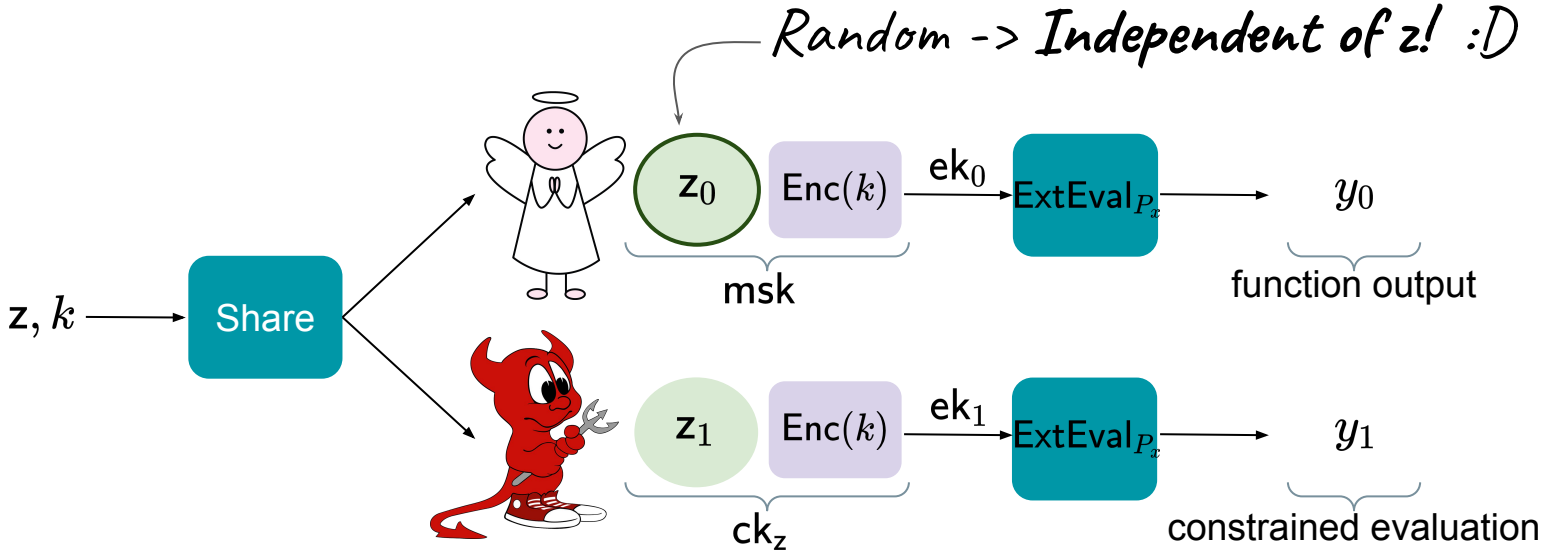




Constrained PRF from Homomorphic Secret Sharing

Inner-Product Constraint constraint C : vector $\mathbf{z}$ $P_x : (k, z) \mapsto \langle z, x \rangle \cdot F_k(x)$

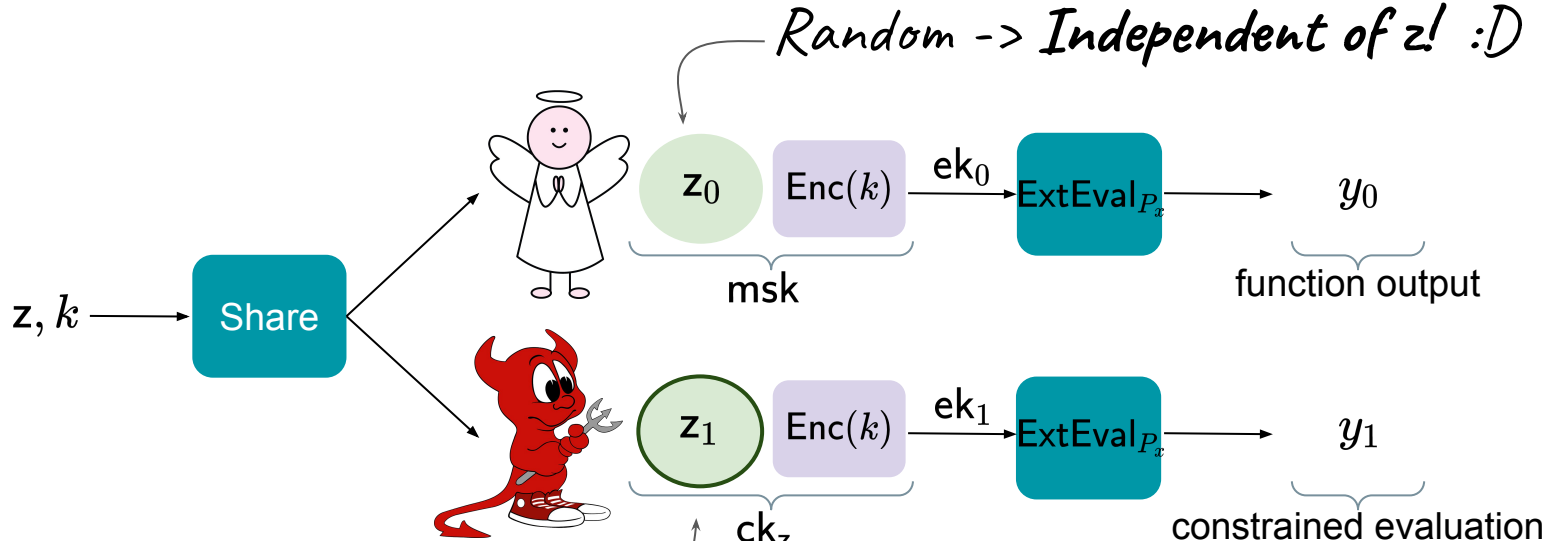
Construction



Constrained PRF from Homomorphic Secret Sharing

Inner-Product Constraint constraint C : vector $\mathbf{z}$ $P_x : (k, z) \mapsto \langle z, x \rangle \cdot F_k(x)$

Construction



Constrained PRF from Homomorphic Secret Sharing

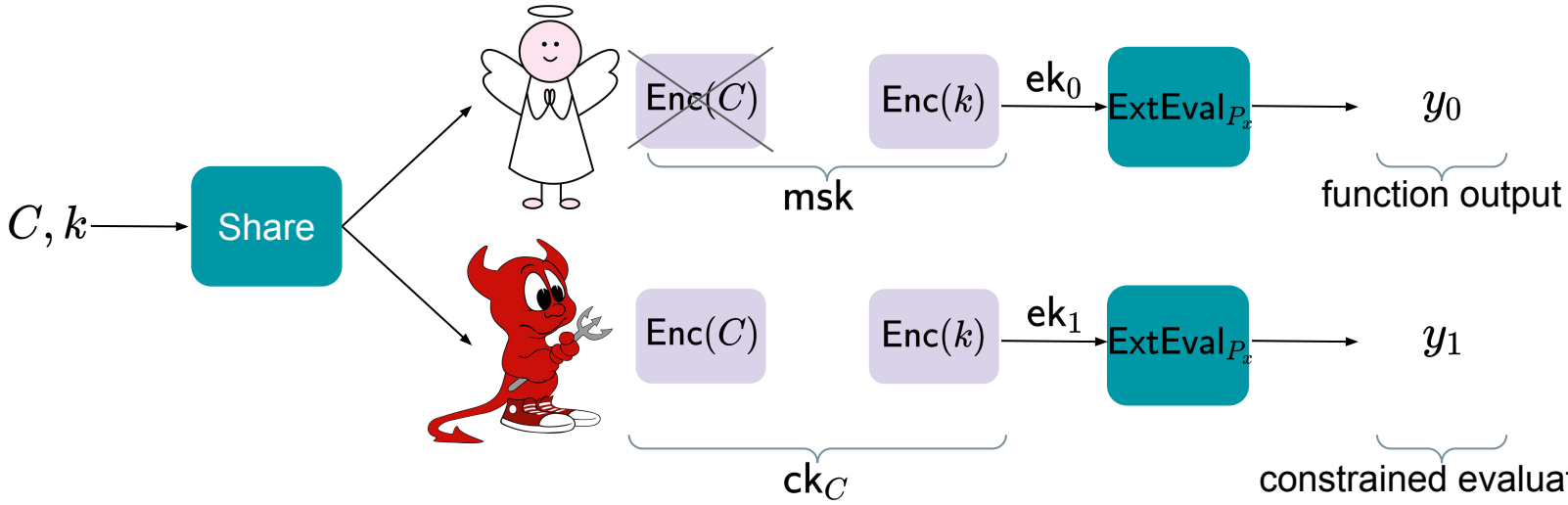
For NC^1

Constrained PRF from Homomorphic Secret Sharing

NC[1] Constraint

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$

Idea



Constrained PRF from Homomorphic Secret Sharing

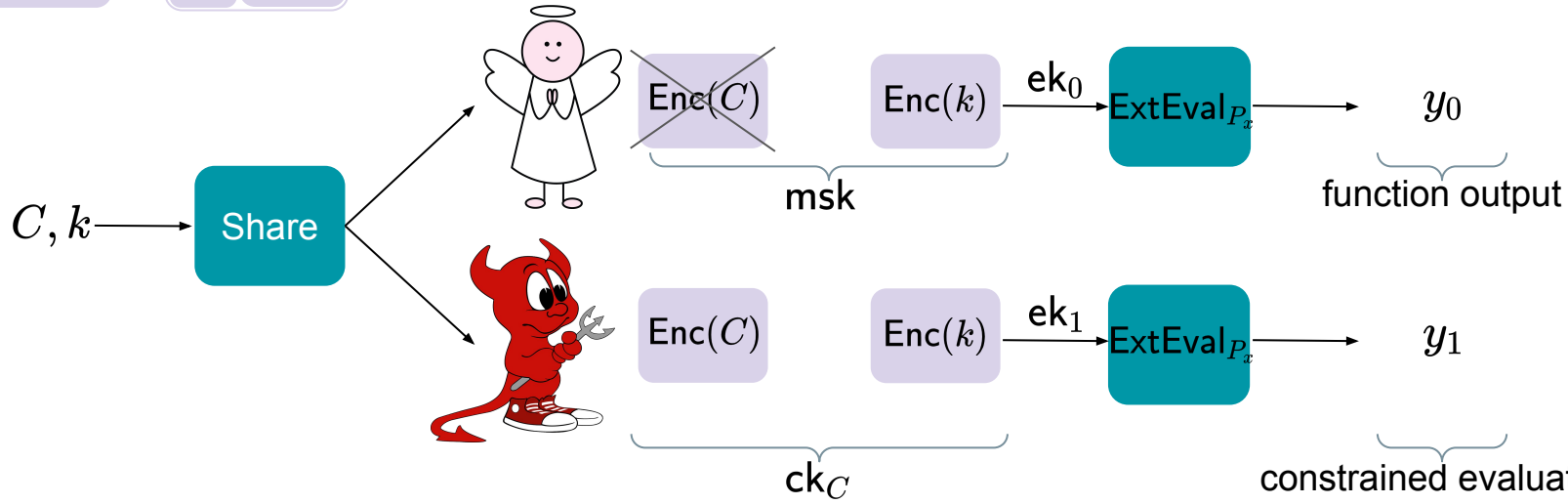
NC¹ Constraint

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$

$g^r, h^r . g^C$ (ElGamal)

$\mathsf{Enc}(C) = \mathsf{Ind}\ \mathsf{Dep}$

Idea

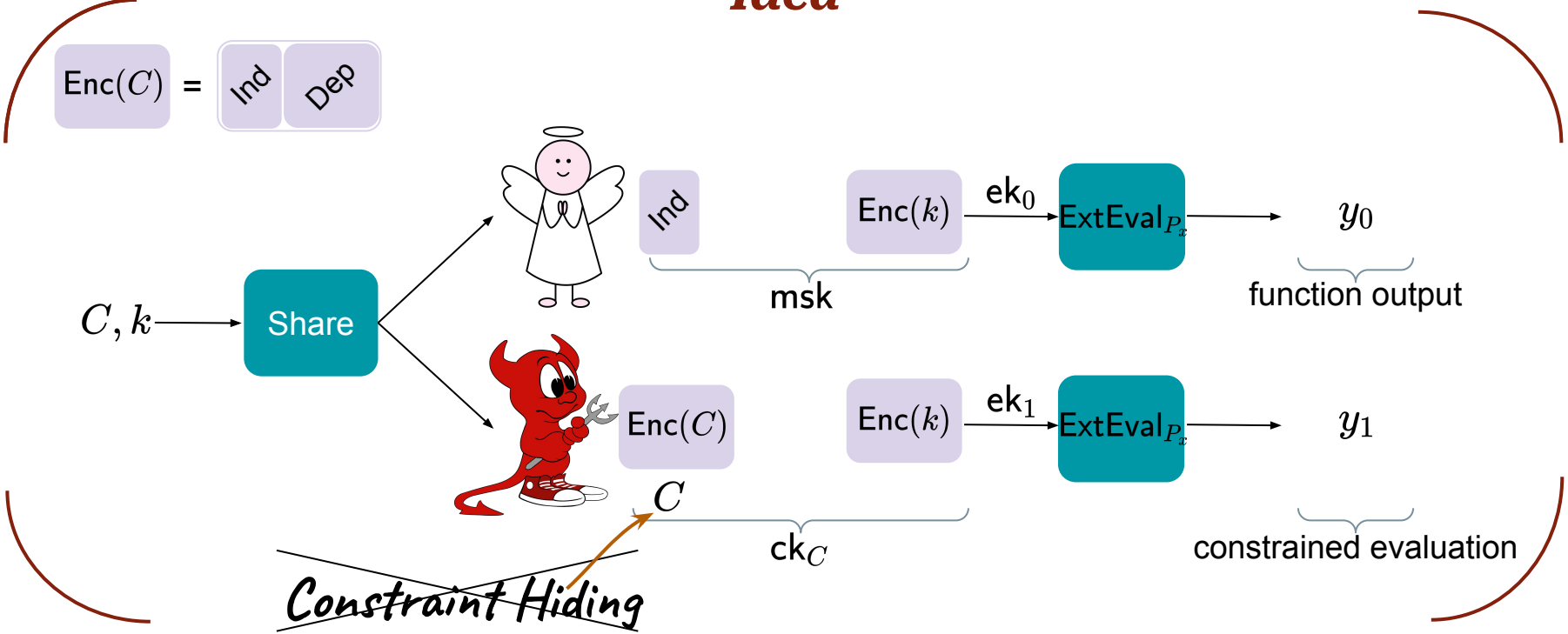


Constrained PRF from Homomorphic Secret Sharing

NC¹ Constraint

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$

Idea

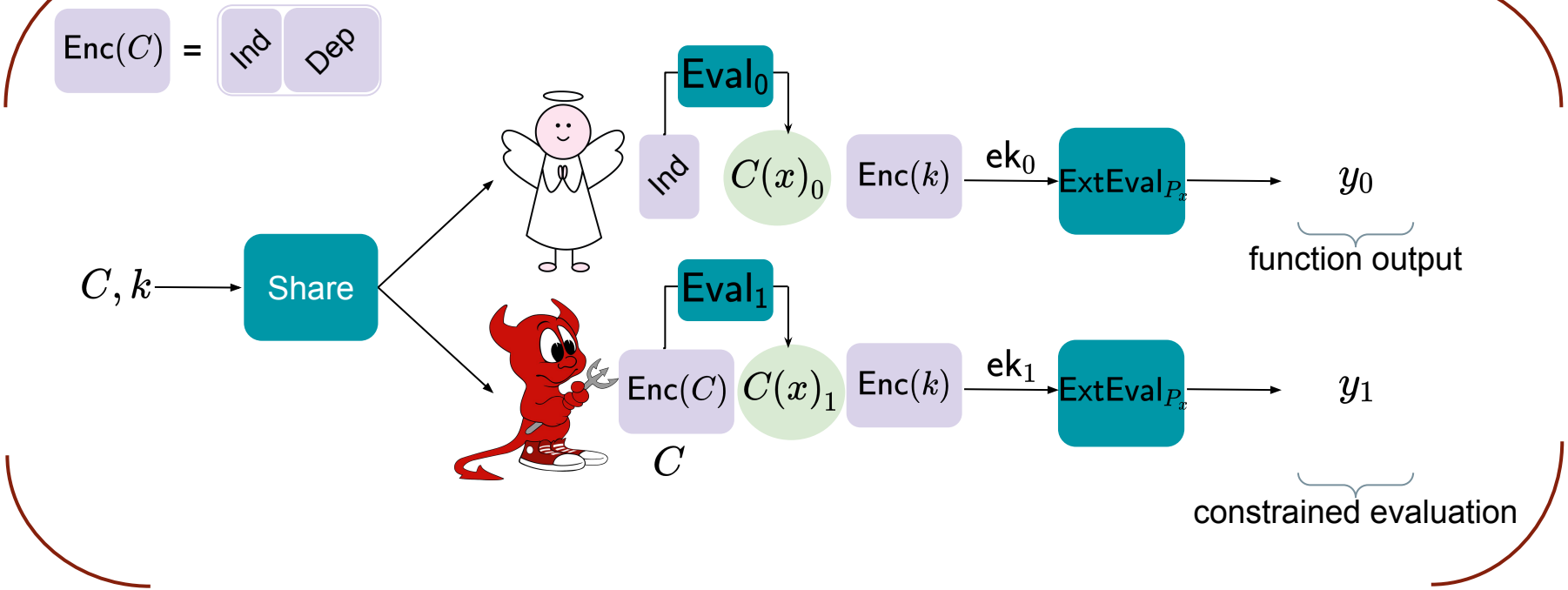


Constrained PRF from Homomorphic Secret Sharing

NC^1 Constraint

$$P_x : (k, C) \mapsto C(x) \cdot F_k(x)$$

Idea



Conclusion

- HSS + (some level of) Programmability -> Constrained PRF (for inner-product and NC^1)
- New constructions of constrained PRF.

  (1) **D**ecisional **C**omposite **R**esiduoisity, (2) **LWE** with superpolynomial modulus, (3) Hardness of the **Joye-Libert** encryption scheme, (4) **DDH & DXDH** over class groups, (5) **H**ard **M**embership **S**ubgroup over class groups
- Revisiting Applications of HSS to Secure Computation.
  - Secure computation with silent preprocessing. (one party can preprocess even before knowing the identity of the other party)
  - One-sided statistically secure computation with sublinear communication. (without FHE!)

Conclusion

- HSS + (some level of) Programmability -> Constrained PRF (for inner-product and NC^1)
- New constructions of constrained PRF.

  (1) **D**ecisional **C**omposite **R**esiduoisity, (2) **LWE** with superpolynomial modulus, (3) Hardness of the **Joye-Libert** encryption scheme, (4) **DDH & DXDH** over class groups, (5) **H**ard **M**embership **S**ubgroup over class groups
- Revisiting Applications of HSS to Secure Computation.
  - Secure computation with silent preprocessing. (one party can preprocess even before knowing the identity of the other party)
  - One-sided statistically secure computation with sublinear communication. (without FHE!)

Thank You!

eprint.iacr.org/2023/387