# Ideal Pseudorandom Code, Revisited

Ganyuan Cao

CANS 2026

November 24, 2026









# Outline

A Background on Pseudorandom Code

UC Security of PRC

(Decoder) Non-Committing Pseudorandom Code

A Gap on Code Structure

Construction for NC-PRCs

# Pseudorandom Code (PRC)

First introduced at [CG24][1]

$$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PRC}.\mathtt{KGen}(1^\lambda)$$

[1] M. Christ and S. Gunn. "Pseudorandom Error-Correcting Codes". *CRYPTO 2024, Part VI*. vol. 14925. LNCS. 2024. DOI: 10.1007/978-3-031-68391-6_10

# Pseudorandom Code (PRC)

First introduced at [CG24][1]

$$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathrm{PRC.}\mathtt{KGen}(1^\lambda)$$

$$c \leftarrow \mathrm{PRC.}\mathtt{Encode}(\mathsf{ek}, m)$$

[1] M. Christ and S. Gunn. "Pseudorandom Error-Correcting Codes". *CRYPTO 2024, Part VI*. vol. 14925. LNCS. 2024. DOI: 10.1007/978-3-031-68391-6_10

# Pseudorandom Code (PRC)

First introduced at [CG24][1]

$$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PRC}.\mathtt{KGen}(1^\lambda)$$

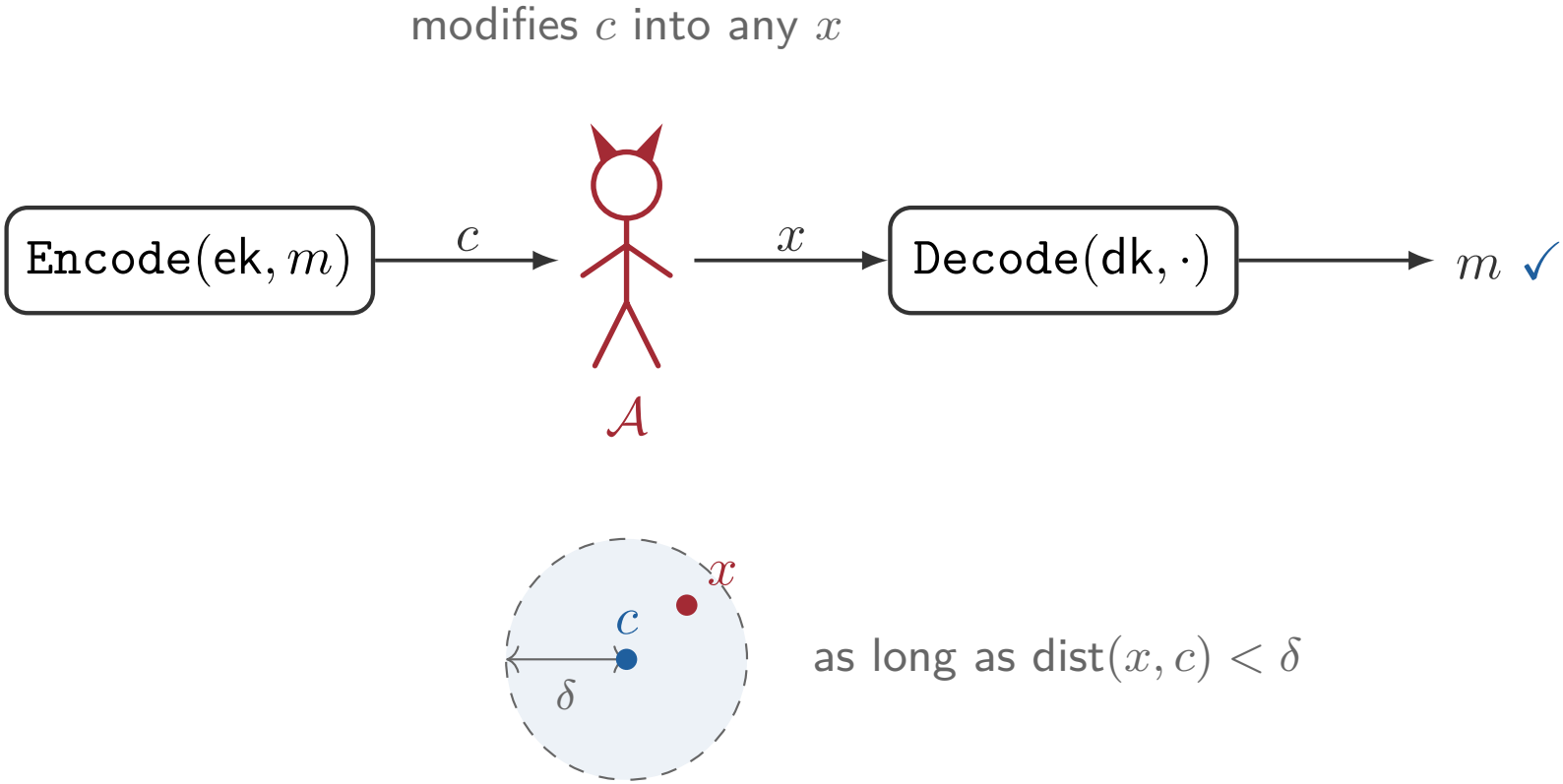
$$c \leftarrow \mathsf{PRC}.\mathtt{Encode}(\mathsf{ek}, m)$$

$$m/\perp \leftarrow \mathsf{PRC}.\mathtt{Decode}(\mathsf{dk}, x)$$

[1] M. Christ and S. Gunn. "Pseudorandom Error-Correcting Codes". *CRYPTO 2024, Part VI*. vol. 14925. LNCS. 2024. DOI: 10.1007/978-3-031-68391-6_10

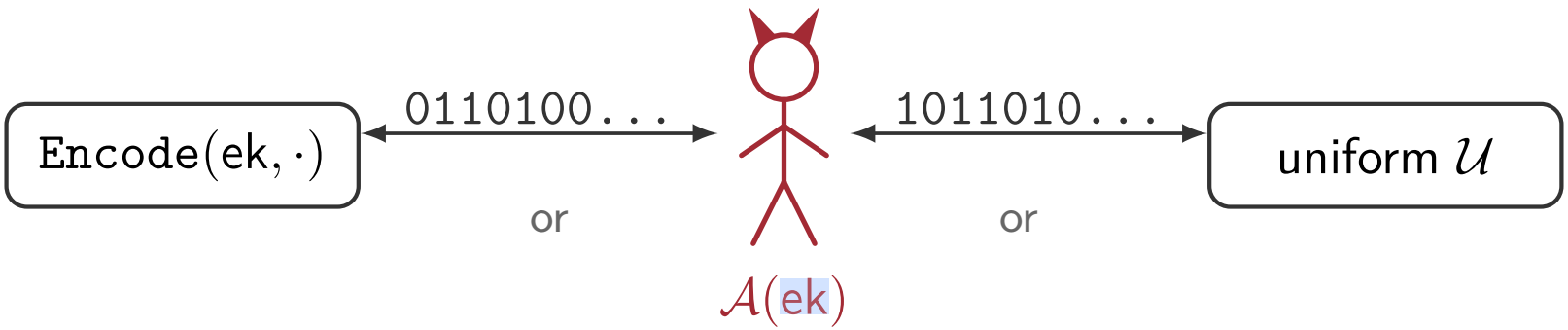
# Two Basic Properties of PRC

(Error-Correcting)



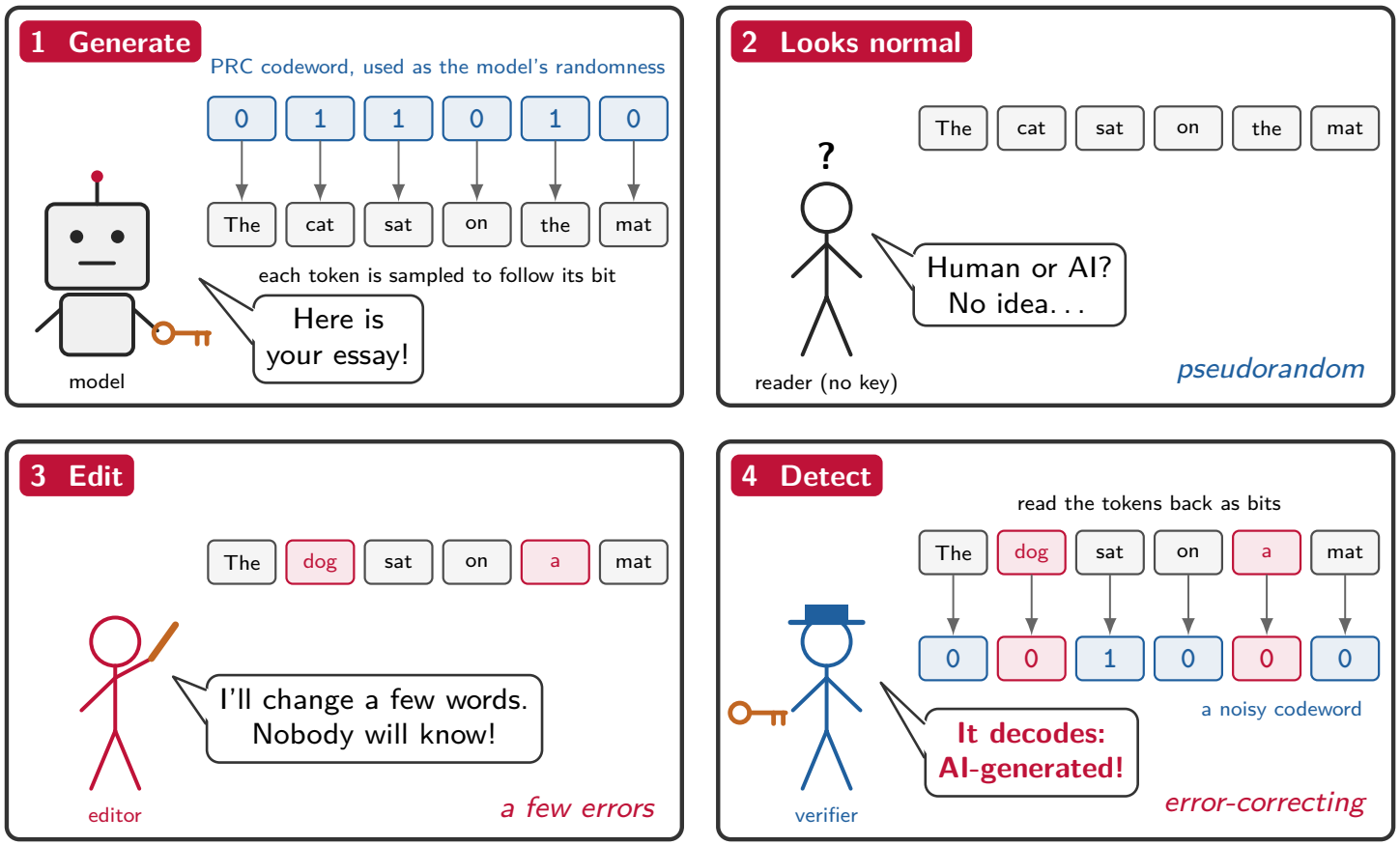
# Two Basic Properties of PRC

(Pseudorandomness)




cannot tell which one it talks to
(even given ek for PK)

# Application: Watermarking LLMs




Watermarking from PRCs [CG24][2]

[2] M. Christ and S. Gunn. "Pseudorandom Error-Correcting Codes". *CRYPTO 2024, Part VI*. vol. 14925. LNCS. 2024. DOI: 10.1007/978-3-031-68391-6_10

# Standalone Security of PRC

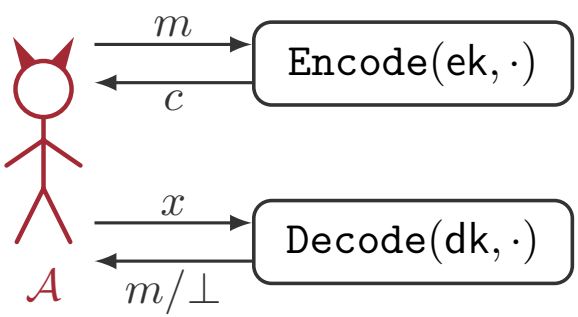
Ideal Security for SK-PRC and CCA Security for PK-PRC [AIr+25][3]:

[3] O. Alrabiah, P. Ananth, M. Christ, Y. Dodis, and S. Gunn. "Ideal Pseudorandom Codes". *57th ACM STOC*. 2025. DOI: 10.1145/3717823.3718309

# Standalone Security of PRC

Ideal Security for SK-PRC and CCA Security for PK-PRC [AIr+25][3]:

**Real ($b = 0$)**

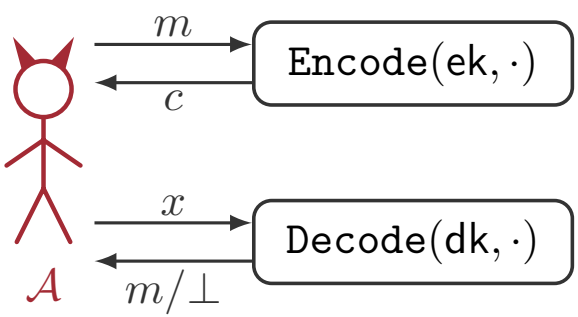



[3] O. Alrabiah, P. Ananth, M. Christ, Y. Dodis, and S. Gunn. "Ideal Pseudorandom Codes". *57th ACM STOC*. 2025. DOI: 10.1145/3717823.3718309

# Standalone Security of PRC

Ideal Security for SK-PRC and CCA Security for PK-PRC [AIr+25][3]:

**Real ($b = 0$)**




**Ideal ($b = 1$)**

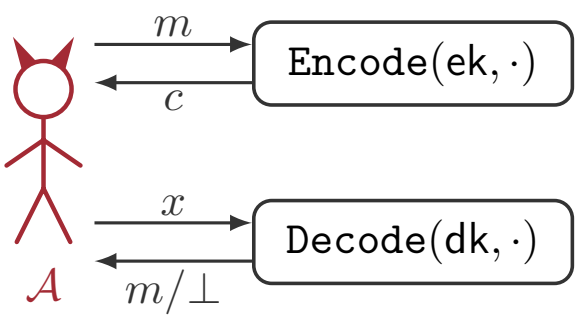



[3] O. Alrabiah, P. Ananth, M. Christ, Y. Dodis, and S. Gunn. "Ideal Pseudorandom Codes". *57th ACM STOC*. 2025. DOI: 10.1145/3717823.3718309

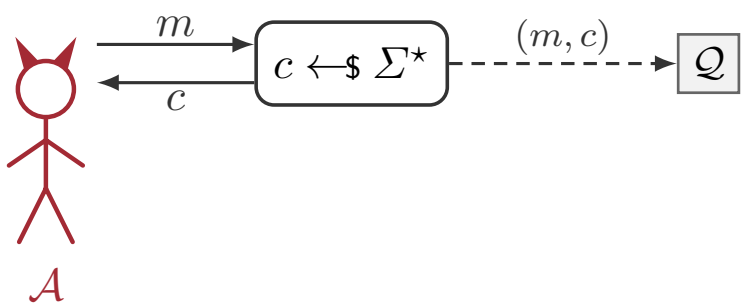
# Standalone Security of PRC

Ideal Security for SK-PRC and CCA Security for PK-PRC [AIr+25][3]:

**Real ($b = 0$)**



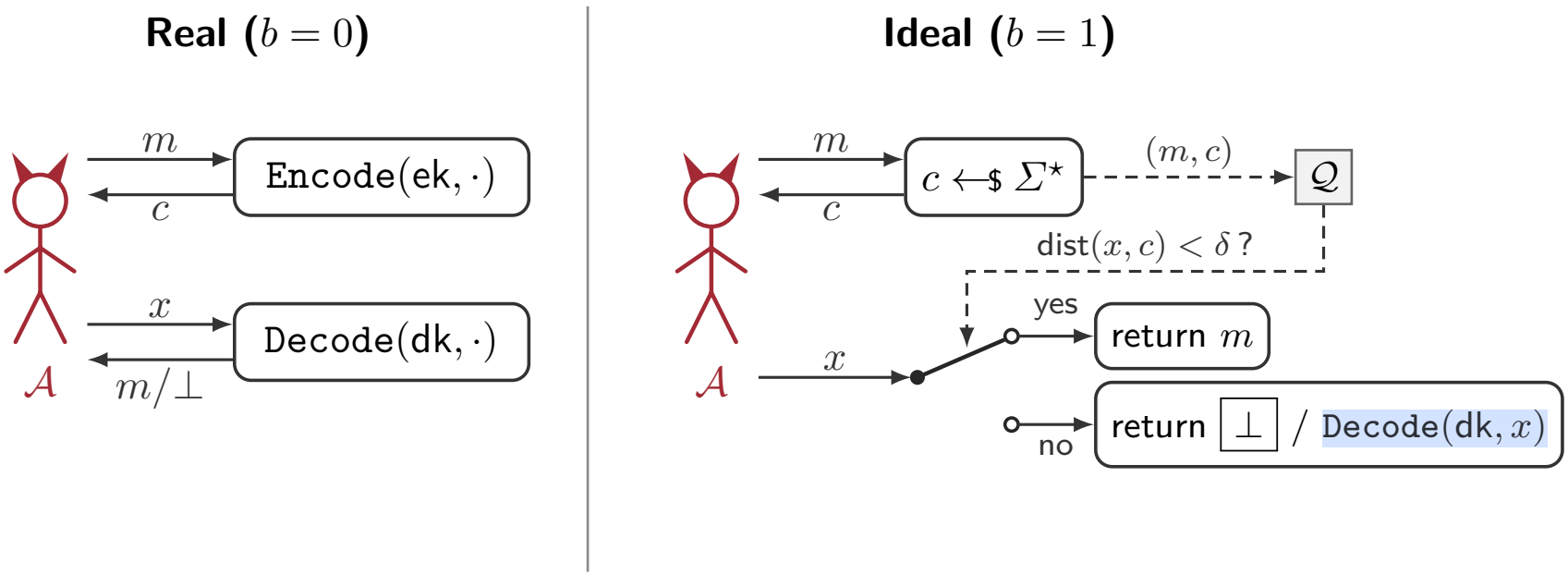

**Ideal ($b = 1$)**




[3] O. Alrabiah, P. Ananth, M. Christ, Y. Dodis, and S. Gunn. "Ideal Pseudorandom Codes". *57th ACM STOC*. 2025. DOI: 10.1145/3717823.3718309

# Standalone Security of PRC

Ideal Security for SK-PRC and CCA Security for PK-PRC [AIr+25][3]:




[3] O. Alrabiah, P. Ananth, M. Christ, Y. Dodis, and S. Gunn. "Ideal Pseudorandom Codes". *57th ACM STOC*. 2025. DOI: 10.1145/3717823.3718309

# Outline

A Background on Pseudorandom Code

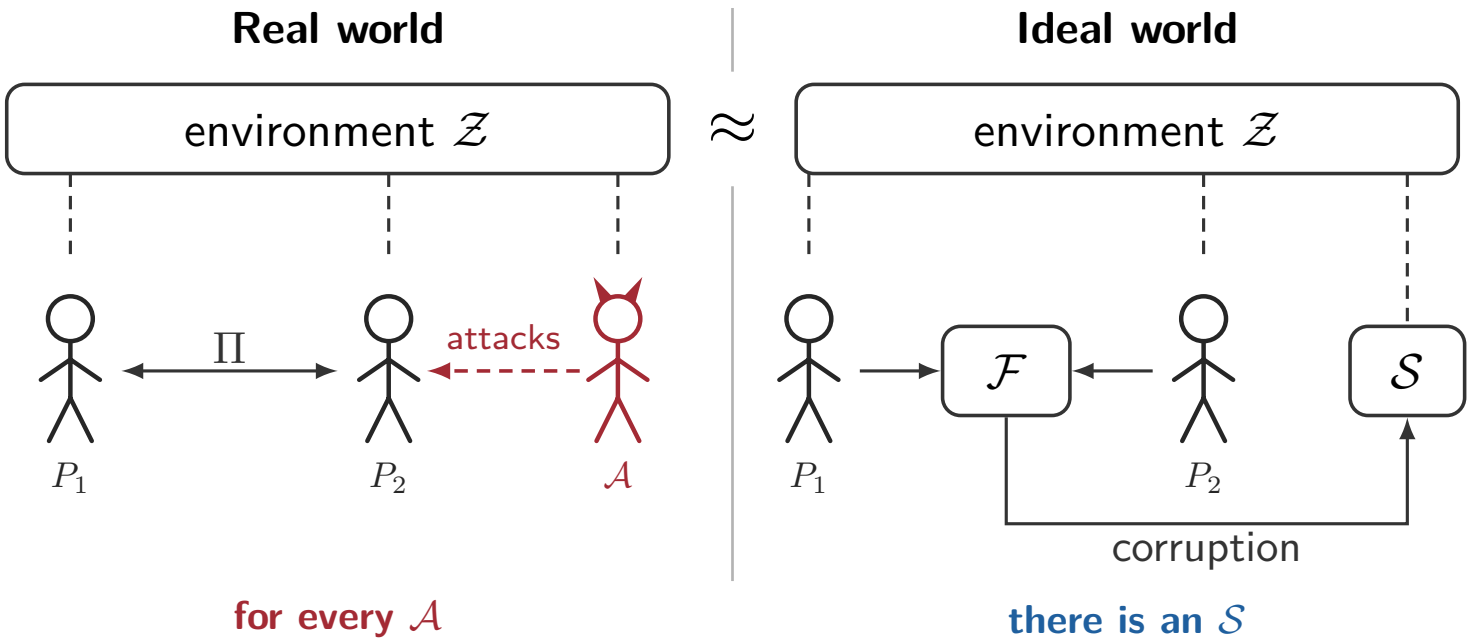
UC Security of PRC

(Decoder) Non-Committing Pseudorandom Code

A Gap on Code Structure

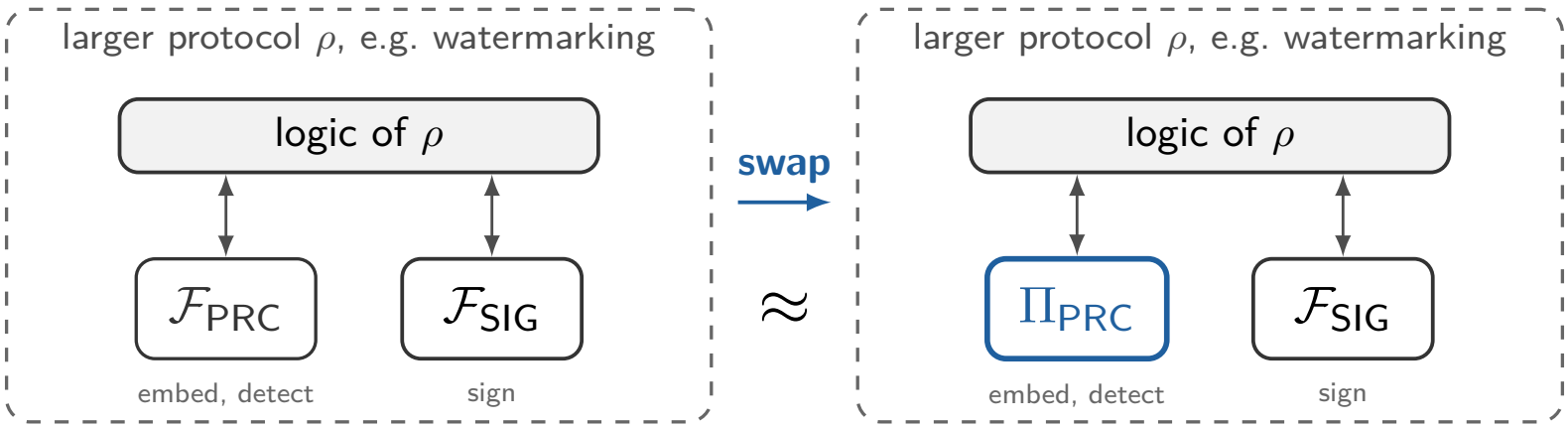
Construction for NC-PRCs

# UC Security in a Nutshell




$\Pi$ *UC-realizes* $\mathcal{F}$: no environment $\mathcal{Z}$ tells the two worlds apart

# UC Security in a Nutshell

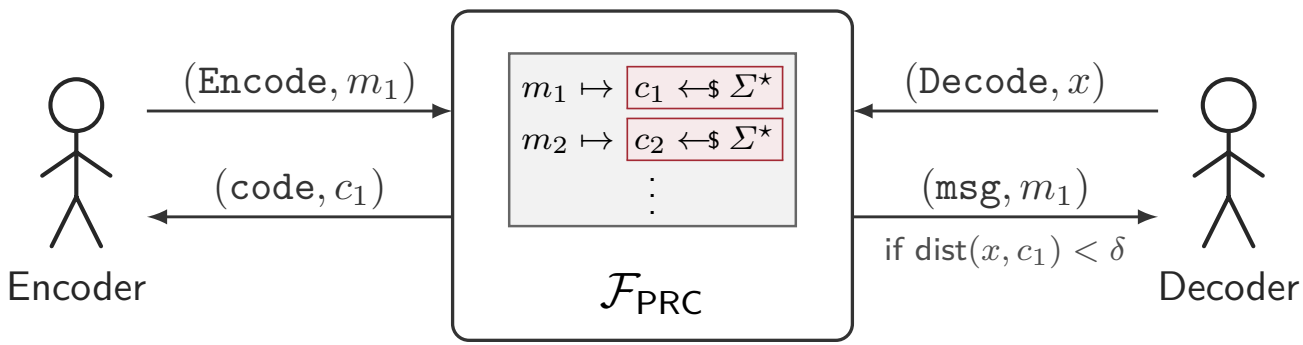



**for every larger protocol $\rho$, whatever else it uses**

composition: swap $\mathcal{F}_{\text{PRC}}$ for $\Pi_{\text{PRC}}$ inside $\rho$, the rest is untouched

# UC Security of PRC

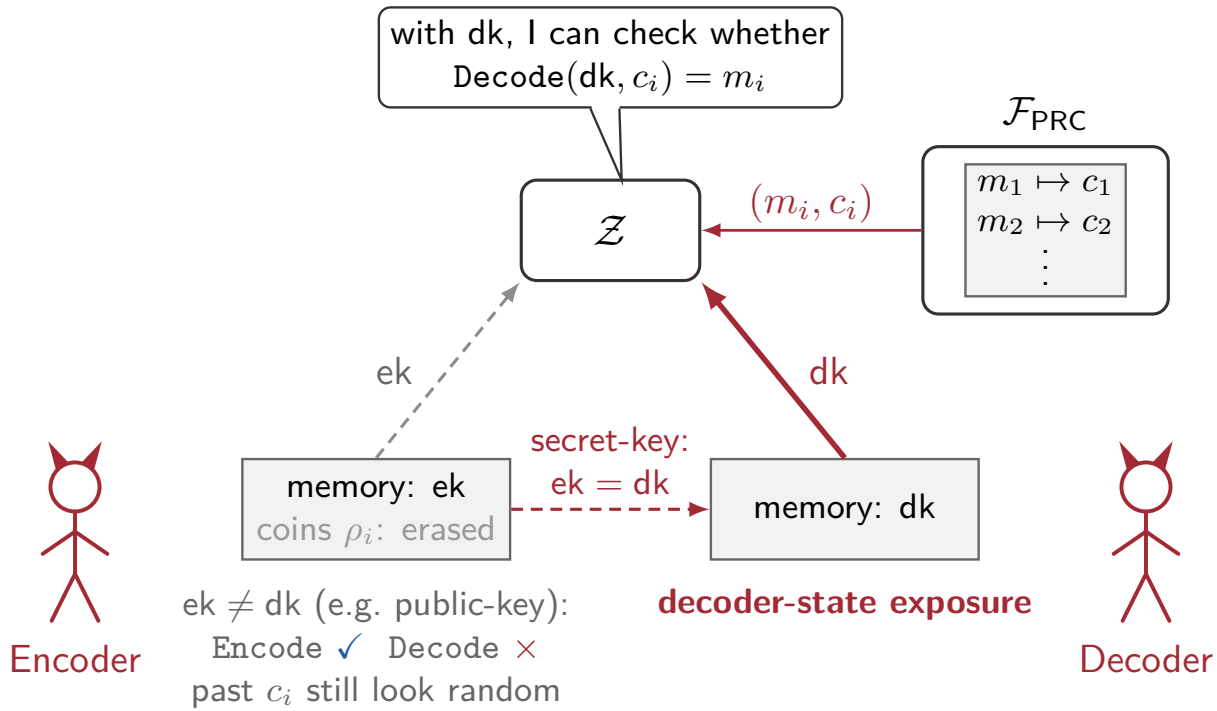
For PRCs, the ideal world is $\mathcal{F}_{\mathsf{PRC}}$ (both parties honest).




random codewords, kept in a table; decoding by lookup

# UC Security of PRC

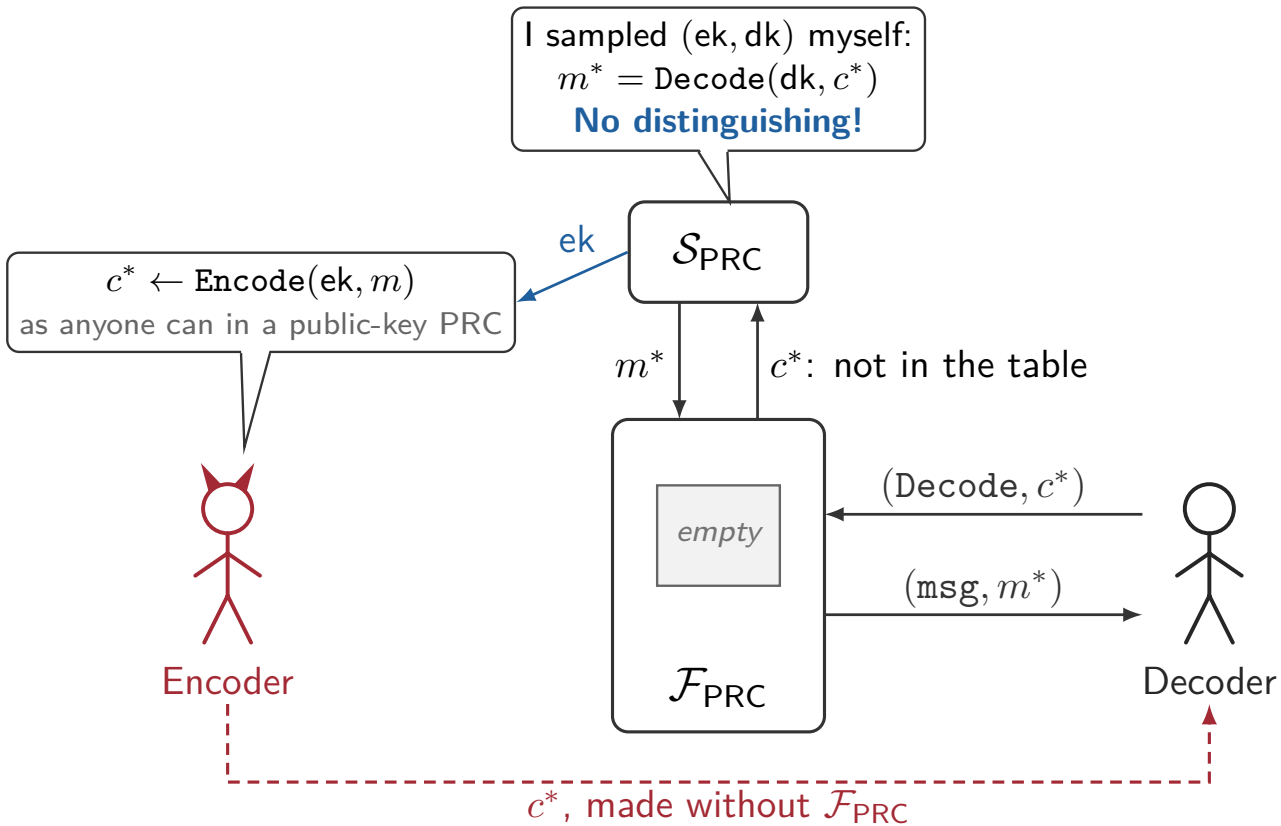
Corruption hands the key to the adversary.



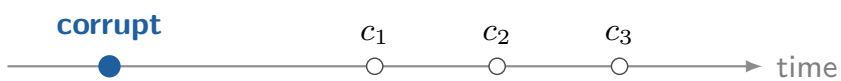

no key decodes a dummy, so everything depends on *when* the key leaks

# UC Security of PRC

**Non-adaptive**: the key leaks *before* the codewords ⇒ no dummies.

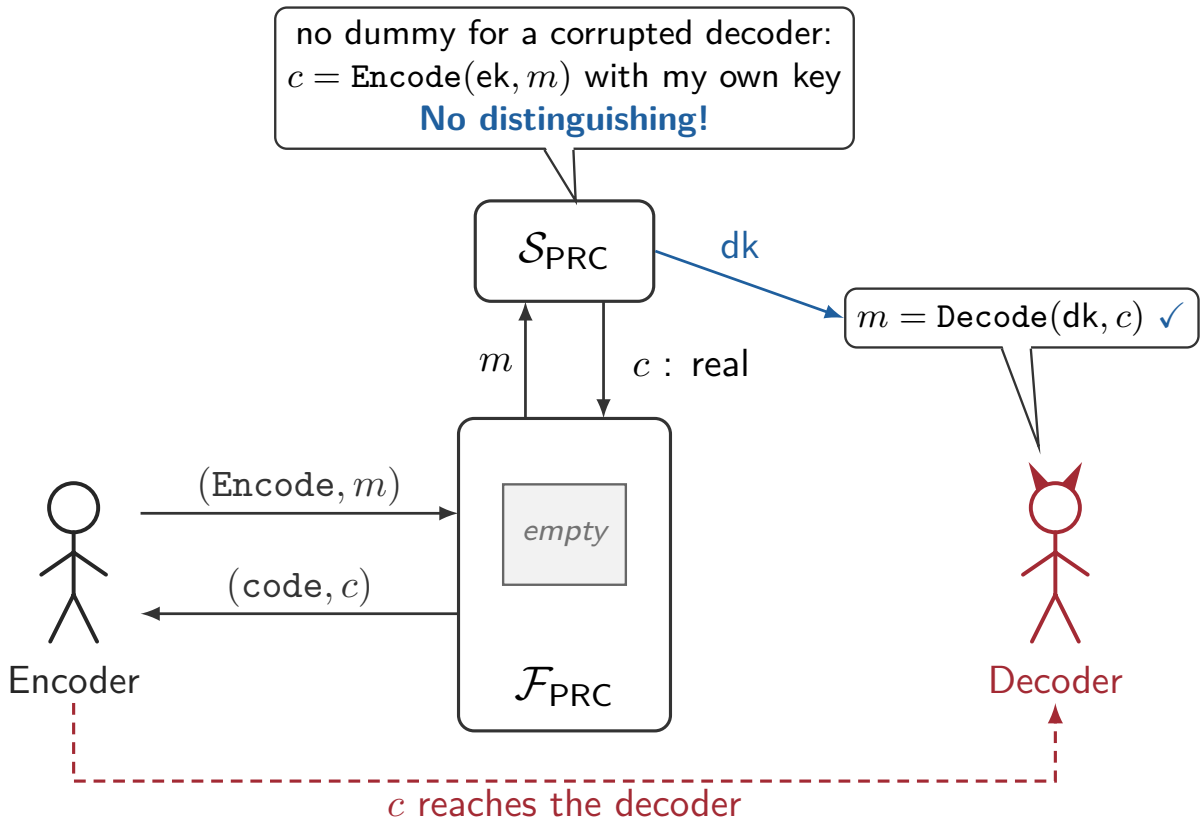



the corrupted party uses the leaked key itself; $\mathcal{S}_{\text{PRC}}$ serves only the honest party

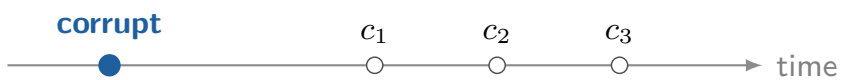


# UC Security of PRC

**Non-adaptive**: the key leaks *before* the codewords ⇒ no dummies.

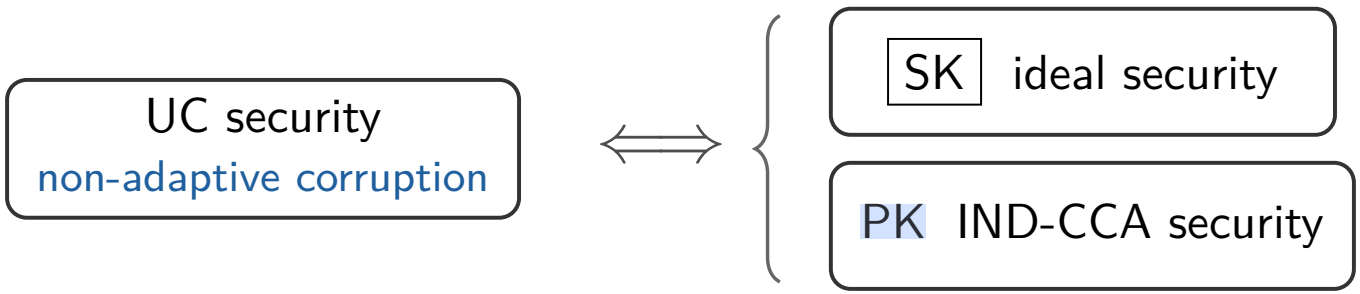



the corrupted party uses the leaked key itself; $\mathcal{S}_{\mathsf{PRC}}$ serves only the honest party



# Equivalence under Non-Adaptive Corruption

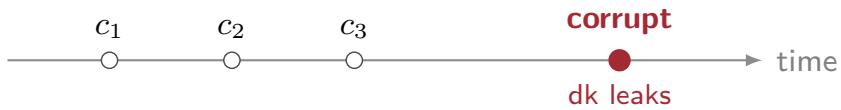
> For **non-adaptive corruption**, UC security asks for *nothing more* than the standalone notions.

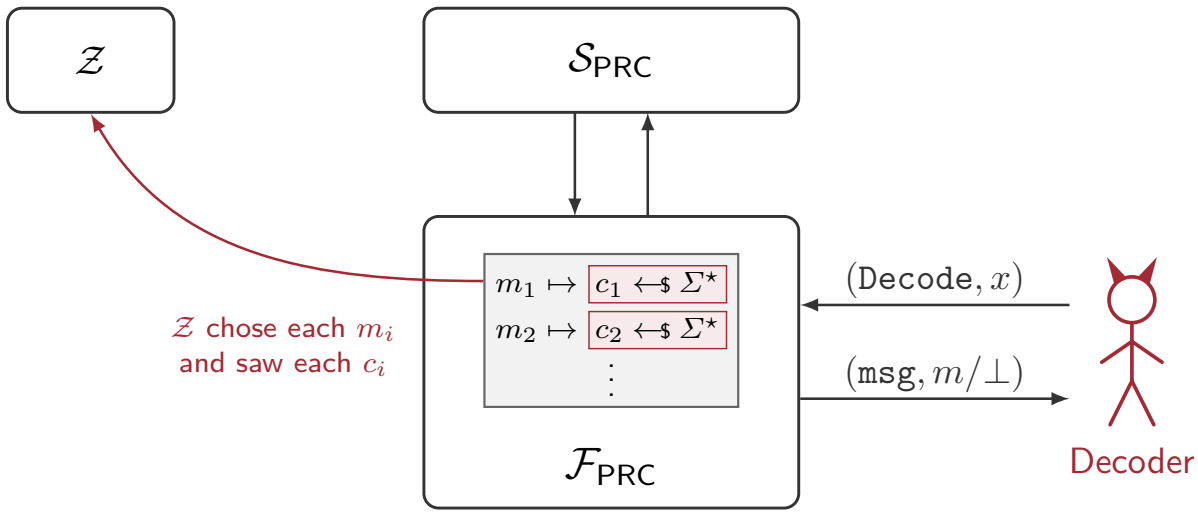



corruption adds no assumption: $\mathcal{S}_{\mathsf{PRC}}$ just holds the key

# Separation under Adaptive Corruption

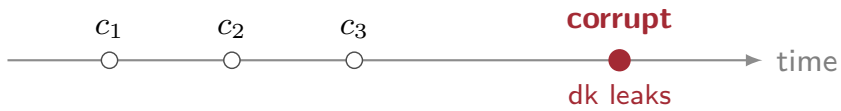
**Adaptive**: the key leaks *after* the codewords ⇒ dummies are out.

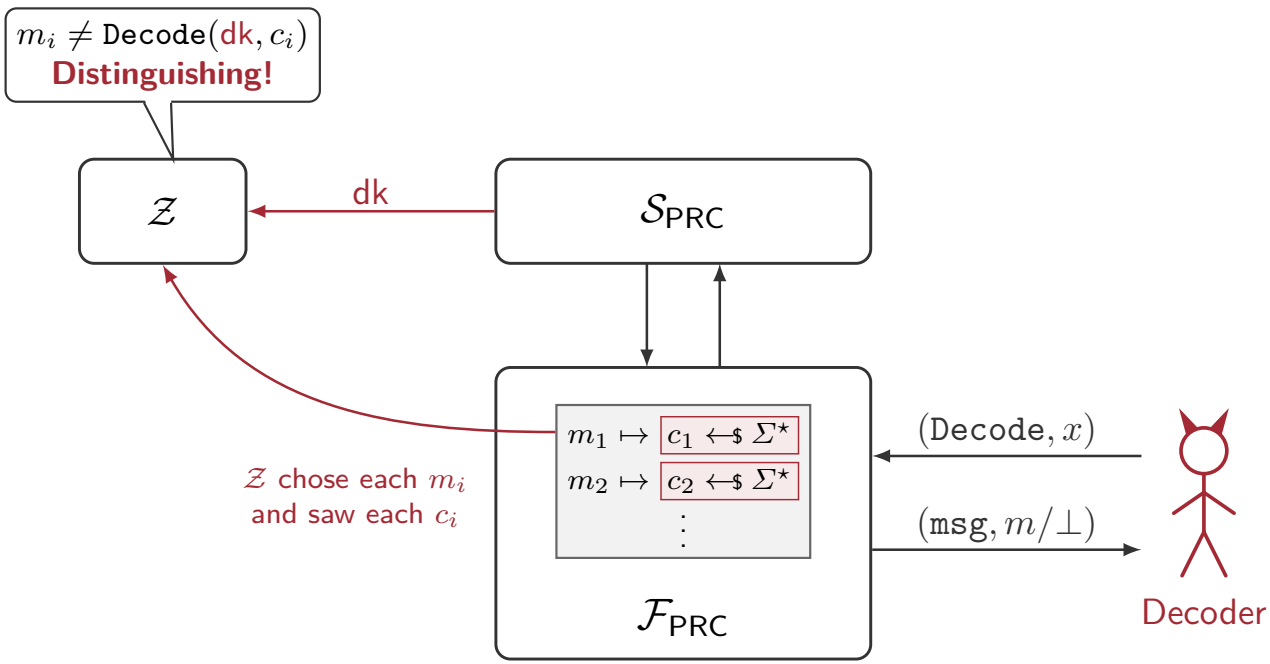


# Separation under Adaptive Corruption

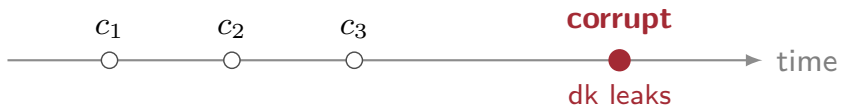
**Adaptive**: the key leaks *after* the codewords ⇒ dummies are out.

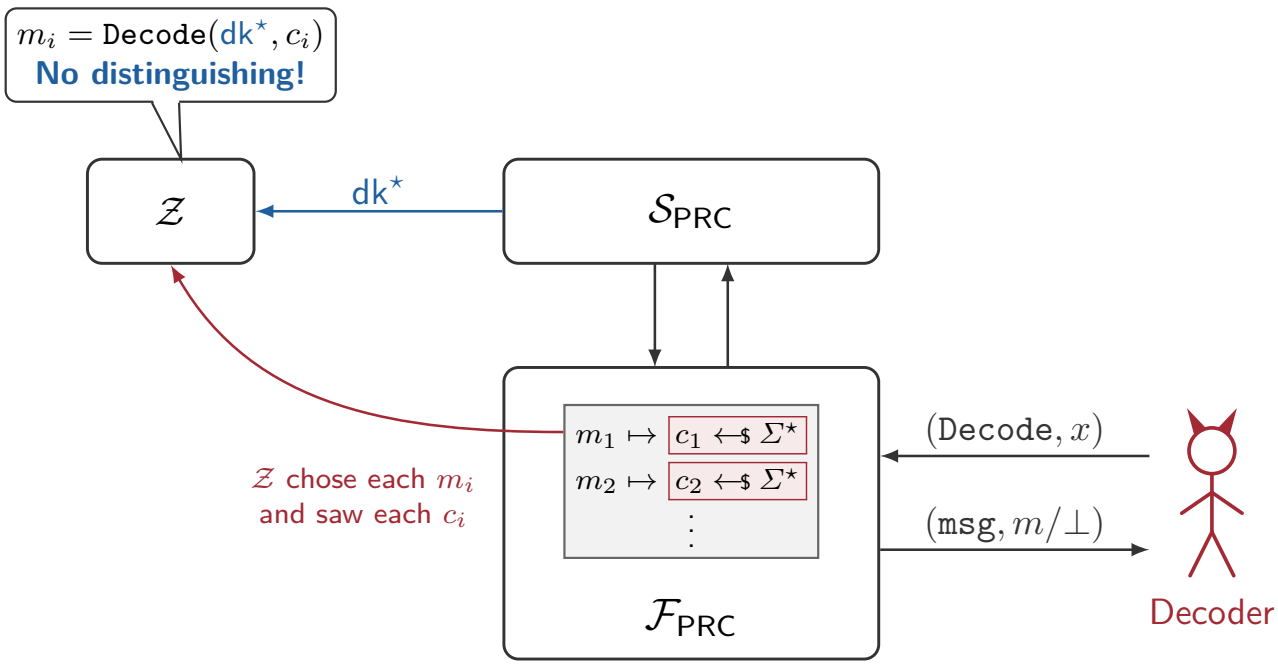


# Separation under Adaptive Corruption

**Adaptive**: the key leaks *after* the codewords ⇒ dummies are out.



# Outline

A Background on Pseudorandom Code

UC Security of PRC

(Decoder) Non-Committing Pseudorandom Code

A Gap on Code Structure

Construction for NC-PRCs

# (Decoder) Non-Committing Pseudorandom Code (NC-PRC)

$(\mathsf{pp}, \mathsf{st}) \leftarrow \mathtt{SimKGen}(1^\lambda)$ simulated keys, with a trapdoor st

# (Decoder) Non-Committing Pseudorandom Code (NC-PRC)

$(\mathsf{pp}, \mathsf{st}) \leftarrow \mathtt{SimKGen}(1^\lambda)$ simulated keys, with a trapdoor st

$c \leftarrow \mathtt{SimCode}(\mathsf{pp})$ a dummy codeword

# (Decoder) Non-Committing Pseudorandom Code (NC-PRC)

$(\mathsf{pp}, \mathsf{st}) \leftarrow \mathtt{SimKGen}(1^\lambda)$ simulated keys, with a trapdoor st

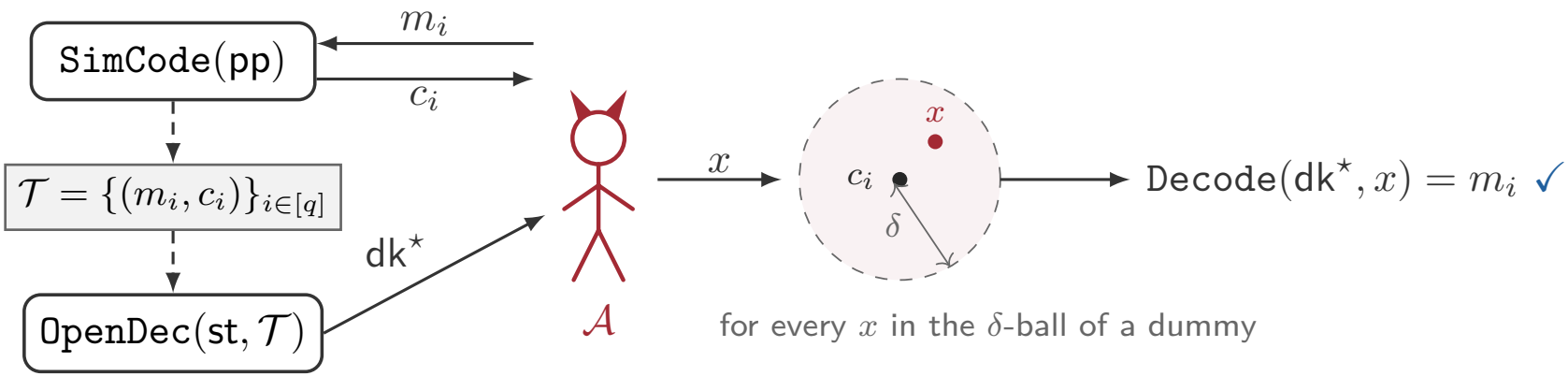
$c \leftarrow \mathtt{SimCode}(\mathsf{pp})$ a dummy codeword

$\mathsf{dk}^\star \leftarrow \mathtt{OpenDec}(\mathsf{st}, \mathcal{T})$ a decoder state explaining $\mathcal{T} = \{(m_i, c_i)\}_i$

# Properties of NC-PRC

## ($\delta$-opening robustness)

after seeing the dummy codewords and the opened $\mathsf{dk}^\star$, $\mathcal{A}$ cannot find an $x$ near a dummy that decodes wrongly



# Properties of NC-PRC

(Decoder-opening indistinguishability)

# Properties of NC-PRC

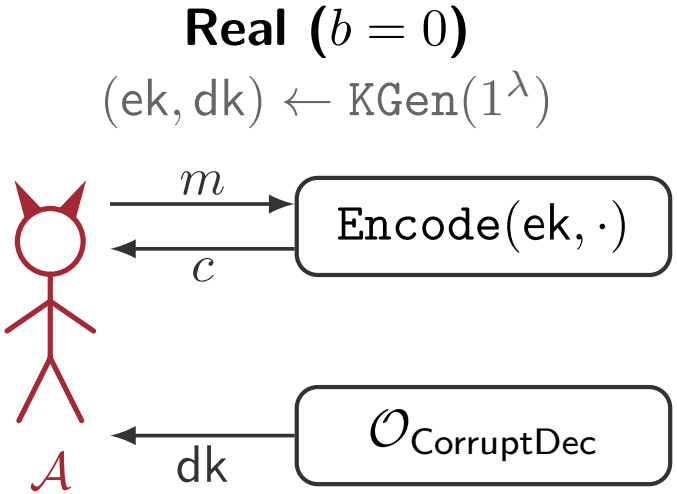
(Decoder-opening indistinguishability)

**Real ($b = 0$)**



# Properties of NC-PRC

(Decoder-opening indistinguishability)

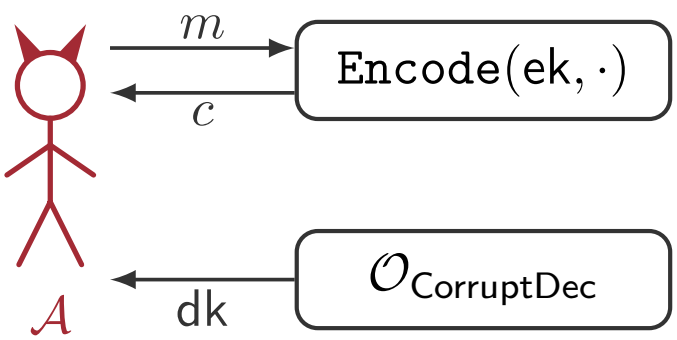


# Properties of NC-PRC

(Decoder-opening indistinguishability)

**Real** ($b = 0$)

$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathtt{KGen}(1^\lambda)$

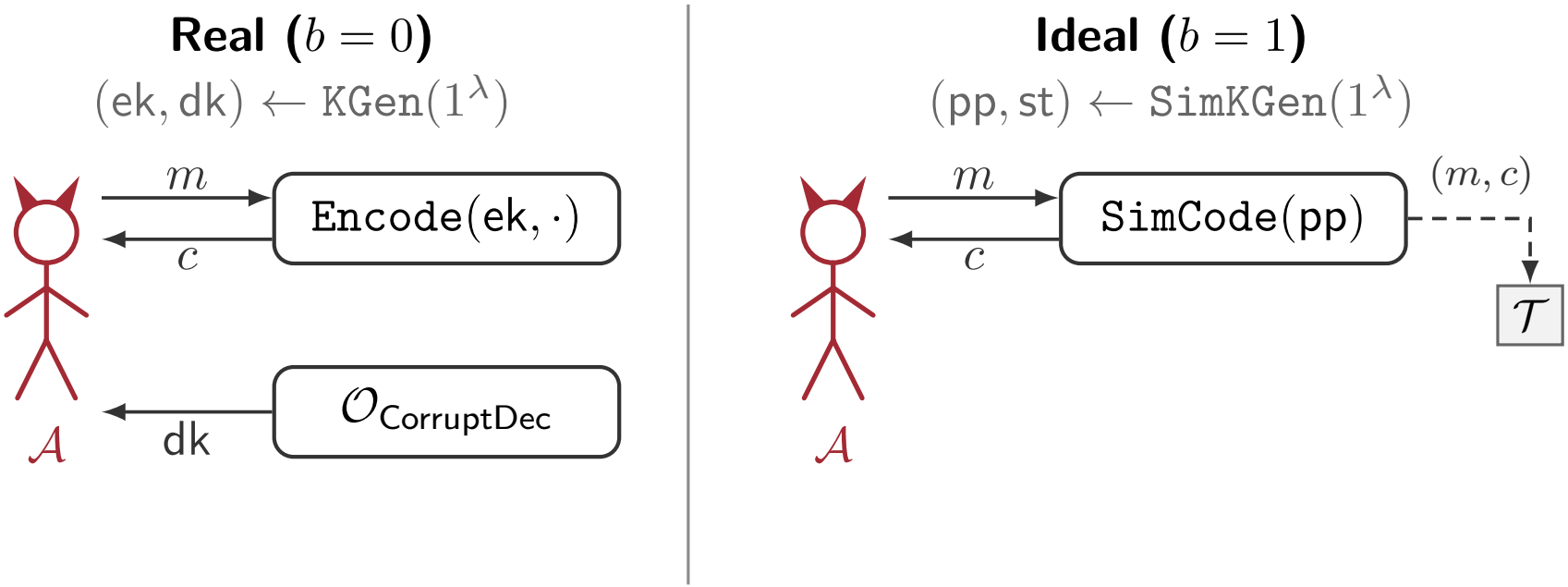



**Ideal** ($b = 1$)



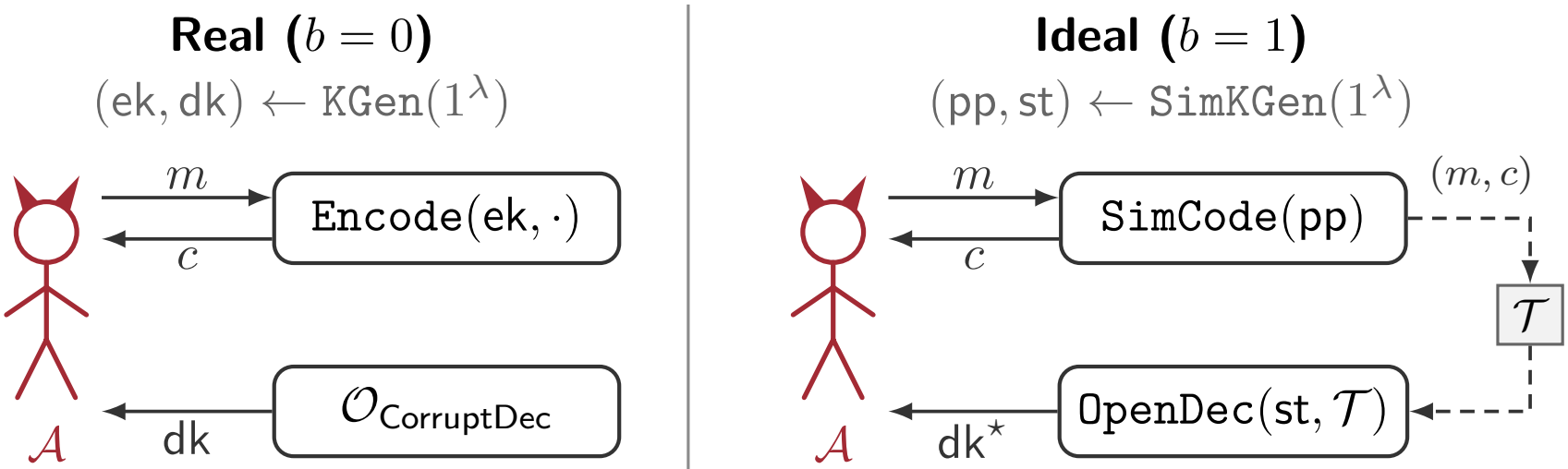
# Properties of NC-PRC

(Decoder-opening indistinguishability)



# Properties of NC-PRC

(Decoder-opening indistinguishability)




no post-corruption encoding: $\mathcal{O}_{\mathsf{Enc}}$ returns $\perp$ once corrupted

# NC-PRC is Necessary

> To be UC-secure against **adaptive corruption**, a PRC *must* admit a (decoder) non-committing code.

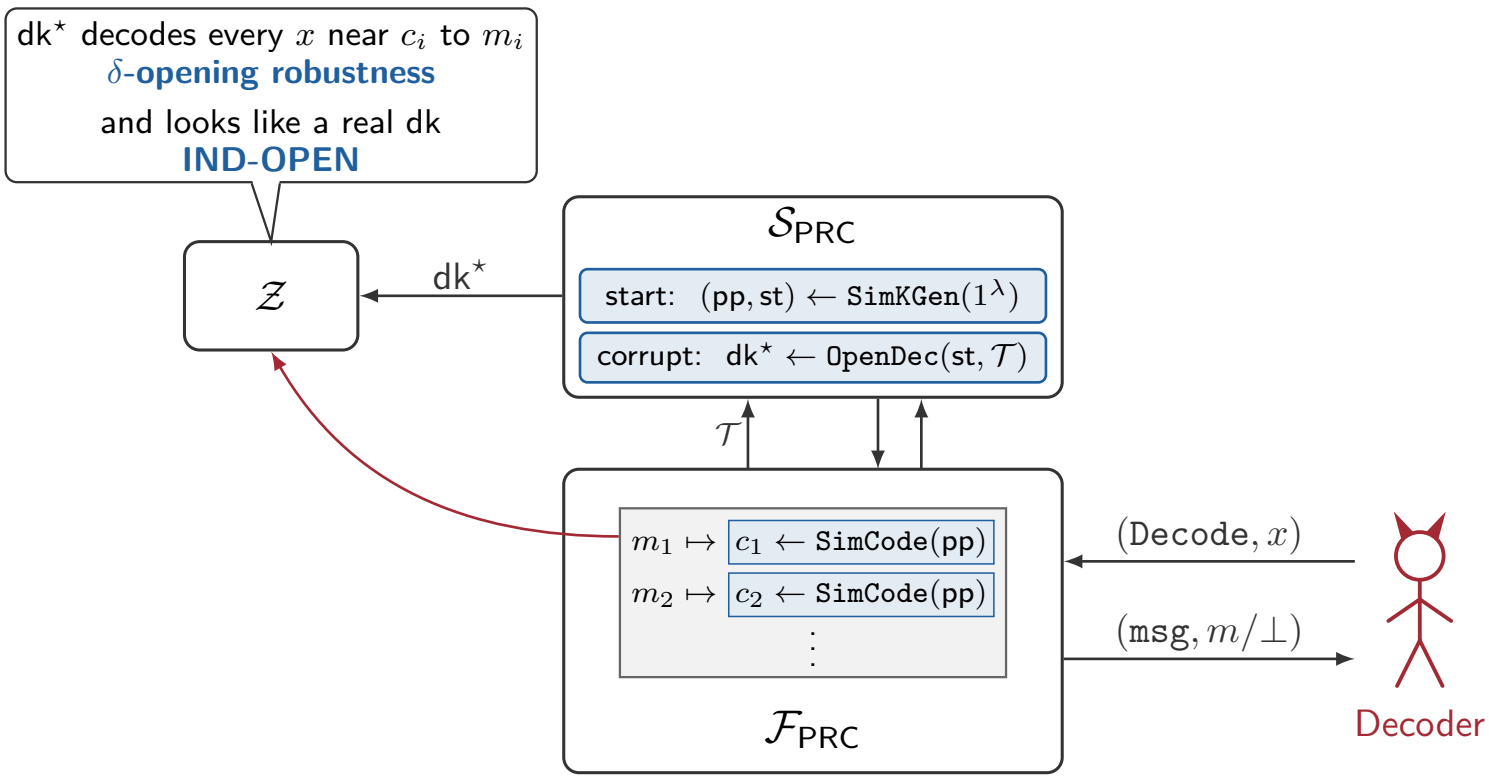
UC security
adaptive corruption

$\Longrightarrow$

an NC-PRC exists
`SimKGen`, `SimCode`, `OpenDec`
$\delta$-opening robustness + IND-OPEN

# NC-PRC is Necessary

Read $\mathcal{S}_{\mathsf{PRC}}$ of an adaptive UC realization as an NC-PRC.



# Outline

A Background on Pseudorandom Code

UC Security of PRC

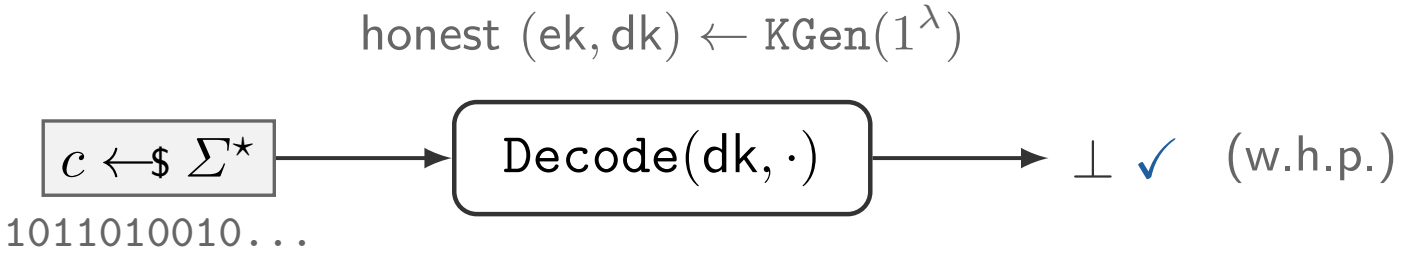
(Decoder) Non-Committing Pseudorandom Code

A Gap on Code Structure

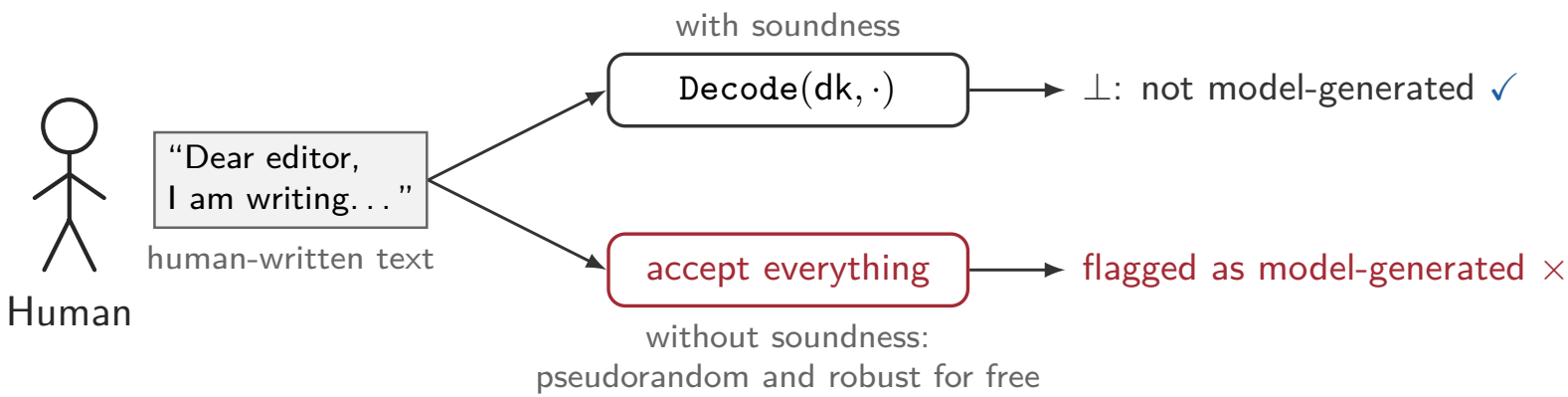
Construction for NC-PRCs

# A Gap on Code Structure

(Soundness)

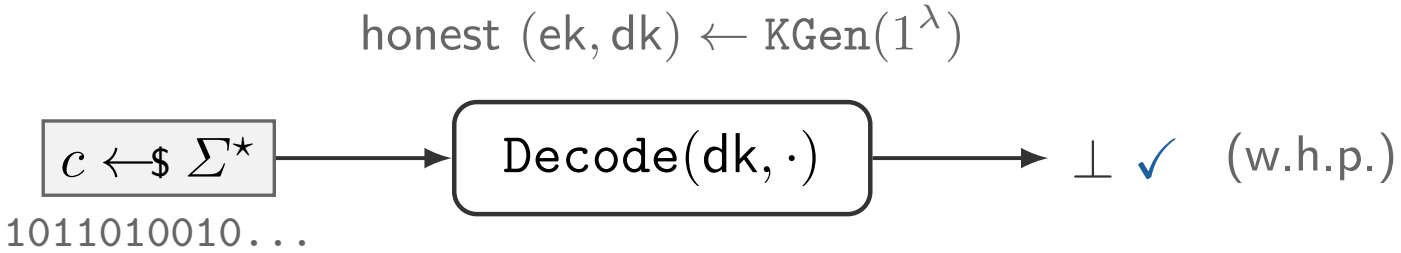



(Why Soundness?) it rules out accepting everything

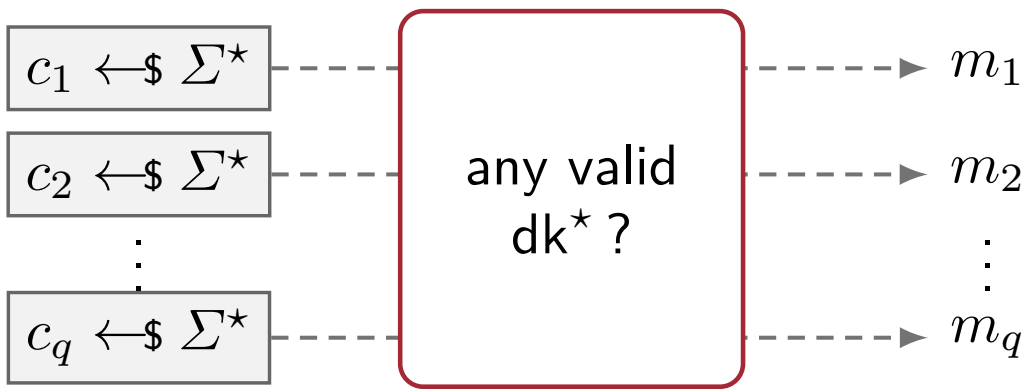


# A Gap on Code Structure

(Soundness)



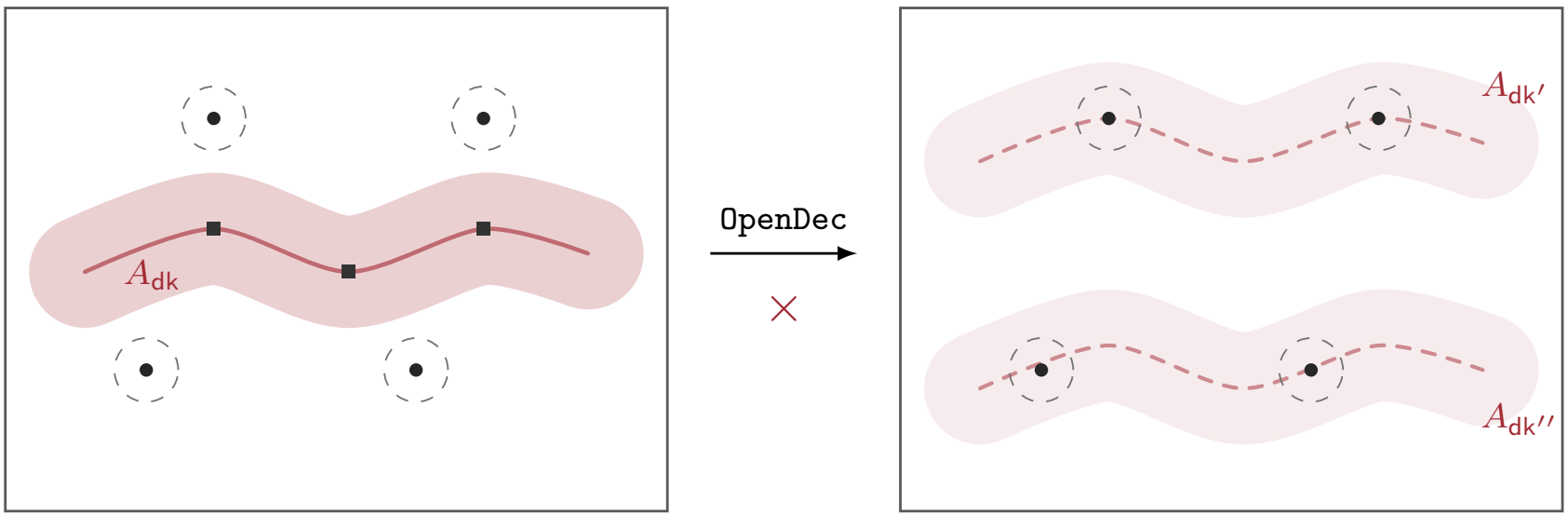

(Low Opening Capacity) for a target $\mathbf{m} = (m_1, \ldots, m_q) \in \Sigma^q$




$\nexists\, \mathsf{dk}^\star$ **(w.h.p.)**: no single key decodes every $c_i$ to its target $m_i$

# A Gap on Code Structure

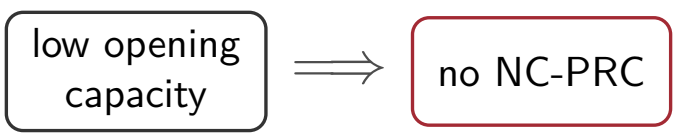
(Low Opening Capacity) valid keys are too *rigid*




valid keys only move the band, never reshape it: none covers all the $\delta$-balls

# A Gap on Code Structure

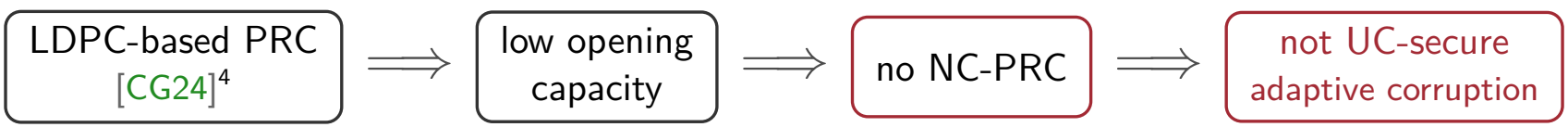
> Low opening capacity rules out non-committing codes.



# A Gap on Code Structure

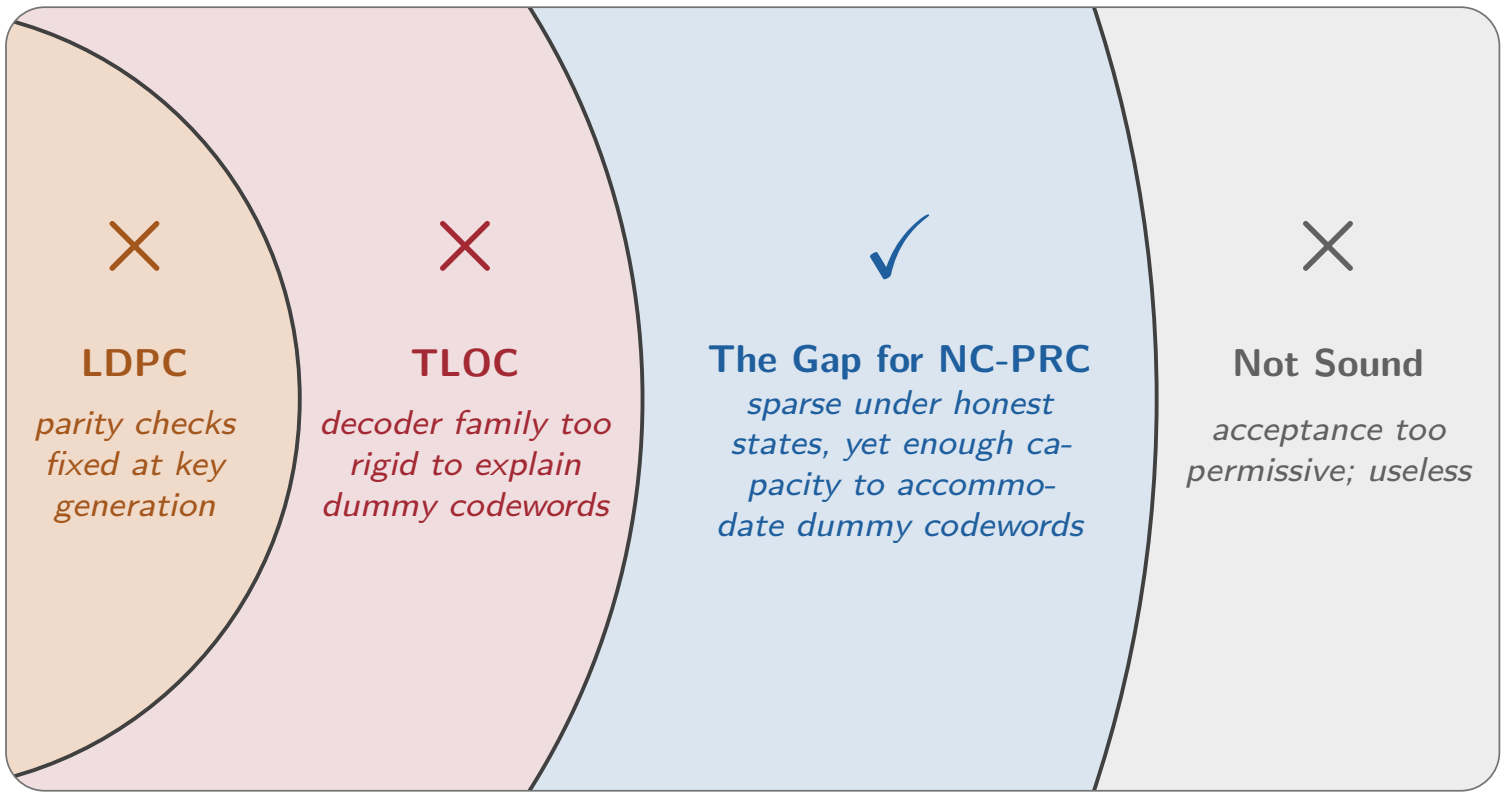
> Low opening capacity rules out non-committing codes.

> LDPC-based PRCs have low opening capacity, so they are *not* UC-secure against **adaptive corruption**.




[4] M. Christ and S. Gunn. "Pseudorandom Error-Correcting Codes". *CRYPTO 2024, Part VI*. vol. 14925. LNCS. 2024. DOI: 10.1007/978-3-031-68391-6_10

# A Gap on Code Structure



# Outline

A Background on Pseudorandom Code

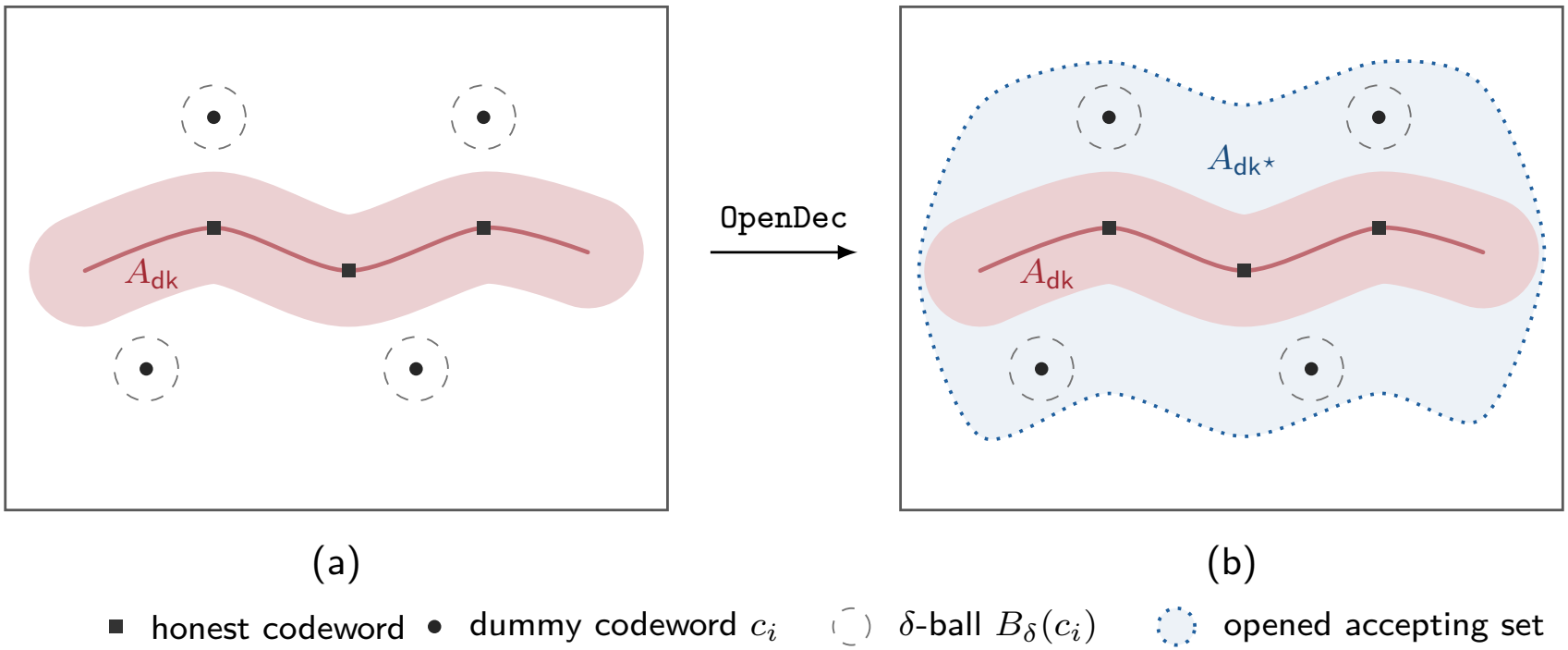
UC Security of PRC

(Decoder) Non-Committing Pseudorandom Code

A Gap on Code Structure

Construction for NC-PRCs

# Idea: Programmable Acceptance



# A Heuristic Secret-key NC-PRC

Error-Correcting Code

robustness

# A Heuristic Secret-key NC-PRC

Error-Correcting Code

robustness

+

# A Heuristic Secret-key NC-PRC

Error-Correcting Code

robustness

+

Puncturable PRF

pseudorandomness

# A Heuristic Secret-key NC-PRC

Error-Correcting Code

robustness

+

Puncturable PRF

pseudorandomness

+

# A Heuristic Secret-key NC-PRC

Error-Correcting Code

robustness

+

Puncturable PRF

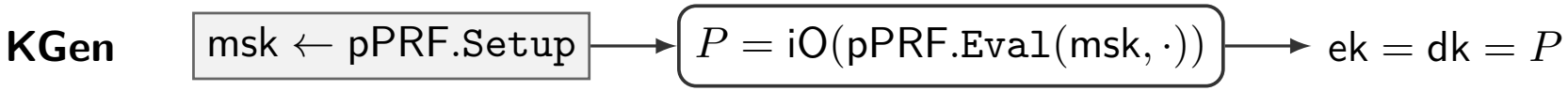
pseudorandomness

+

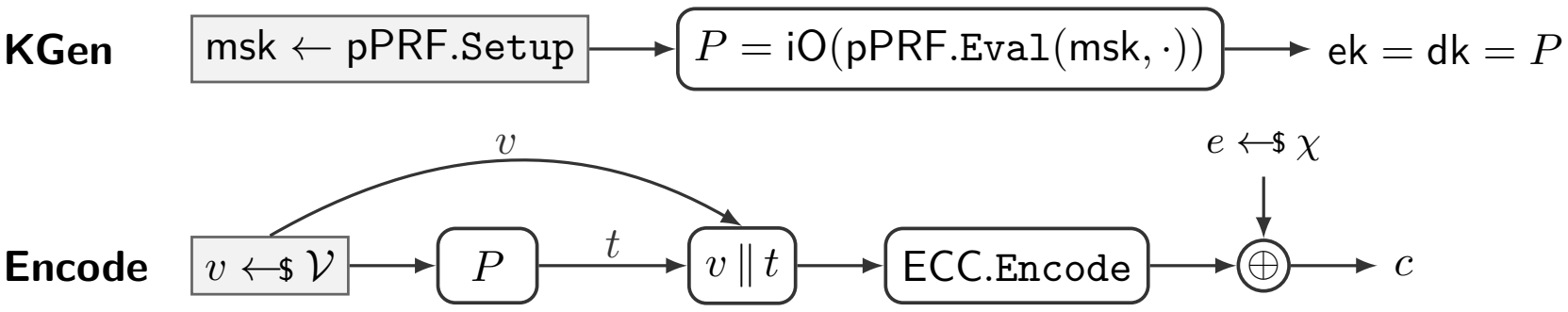
Indistinguishability Obfuscator (iO)

programmability

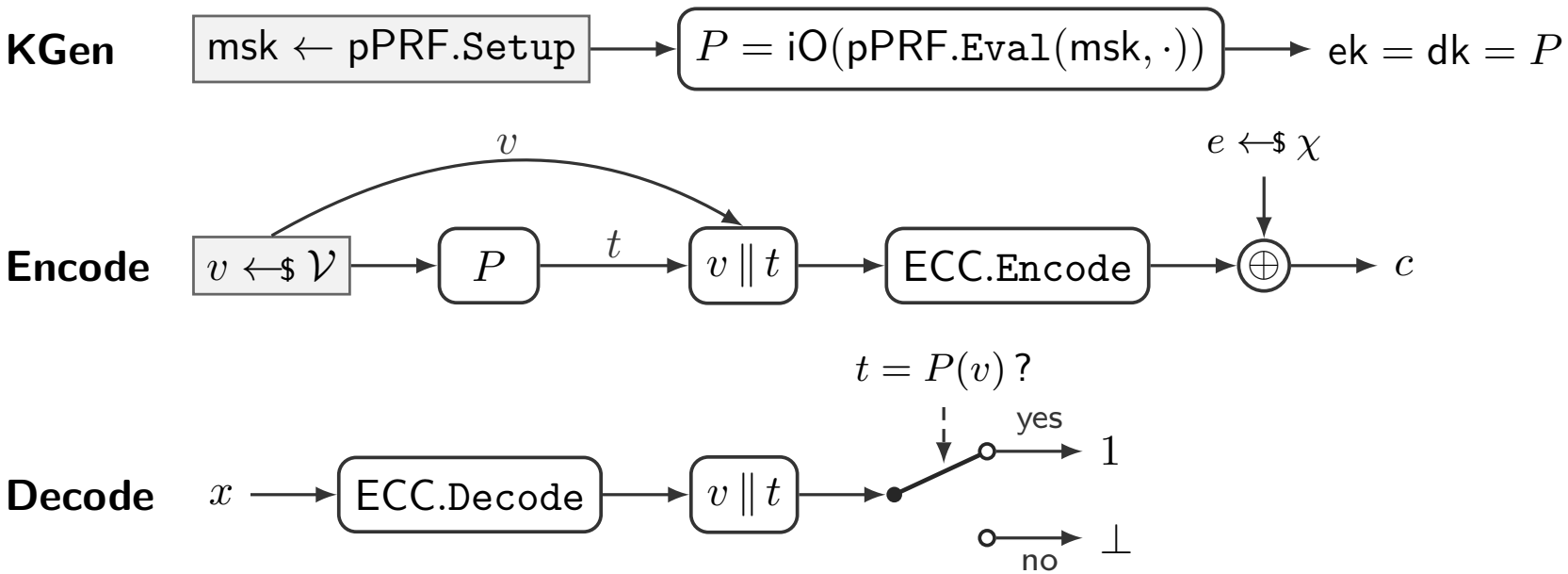
# A Heuristic Secret-Key NC-PRC



# A Heuristic Secret-Key NC-PRC



# A Heuristic Secret-Key NC-PRC



# A Heuristic NC-PRC: Opening the Decoder

dummy codewords

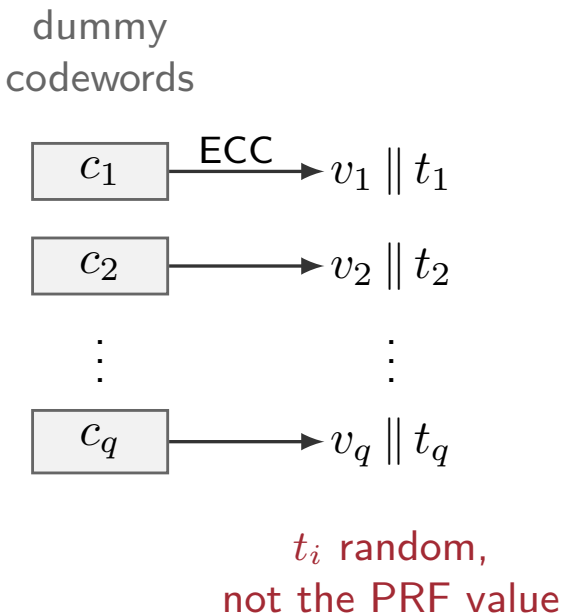
$c_1$

$c_2$

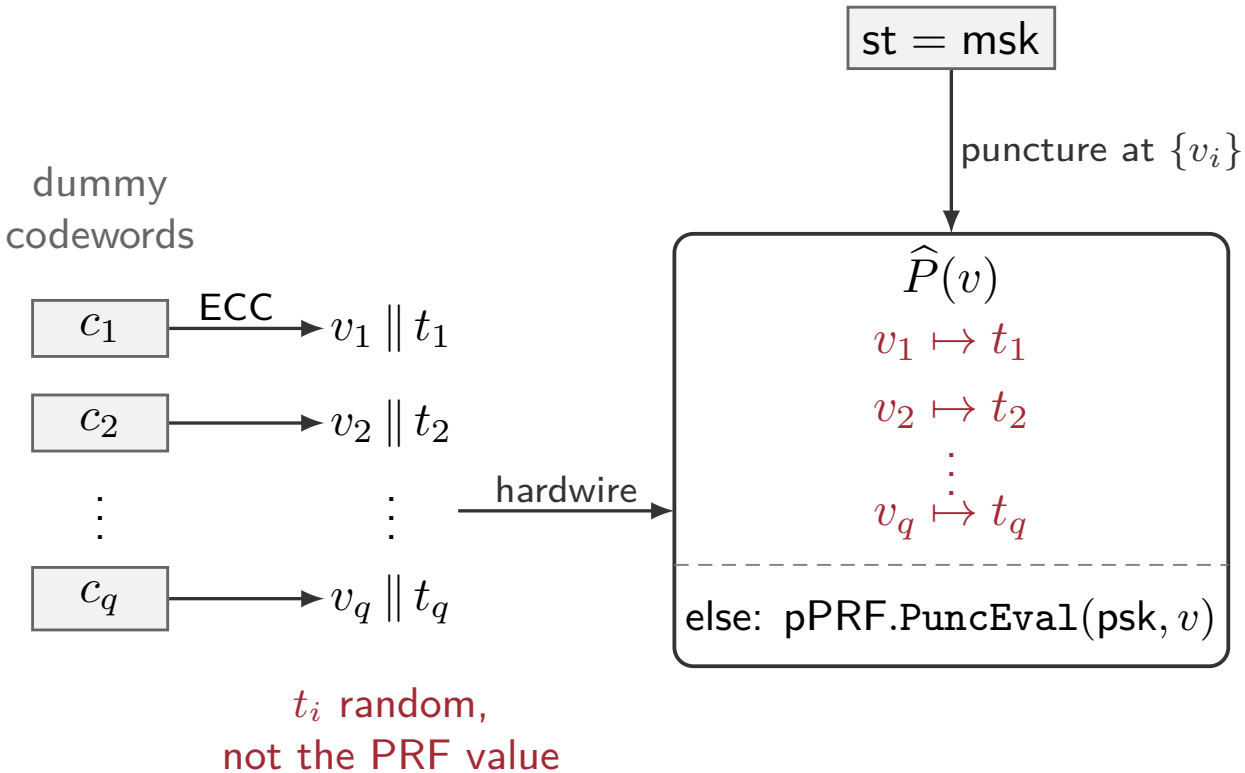
$\vdots$

$c_q$

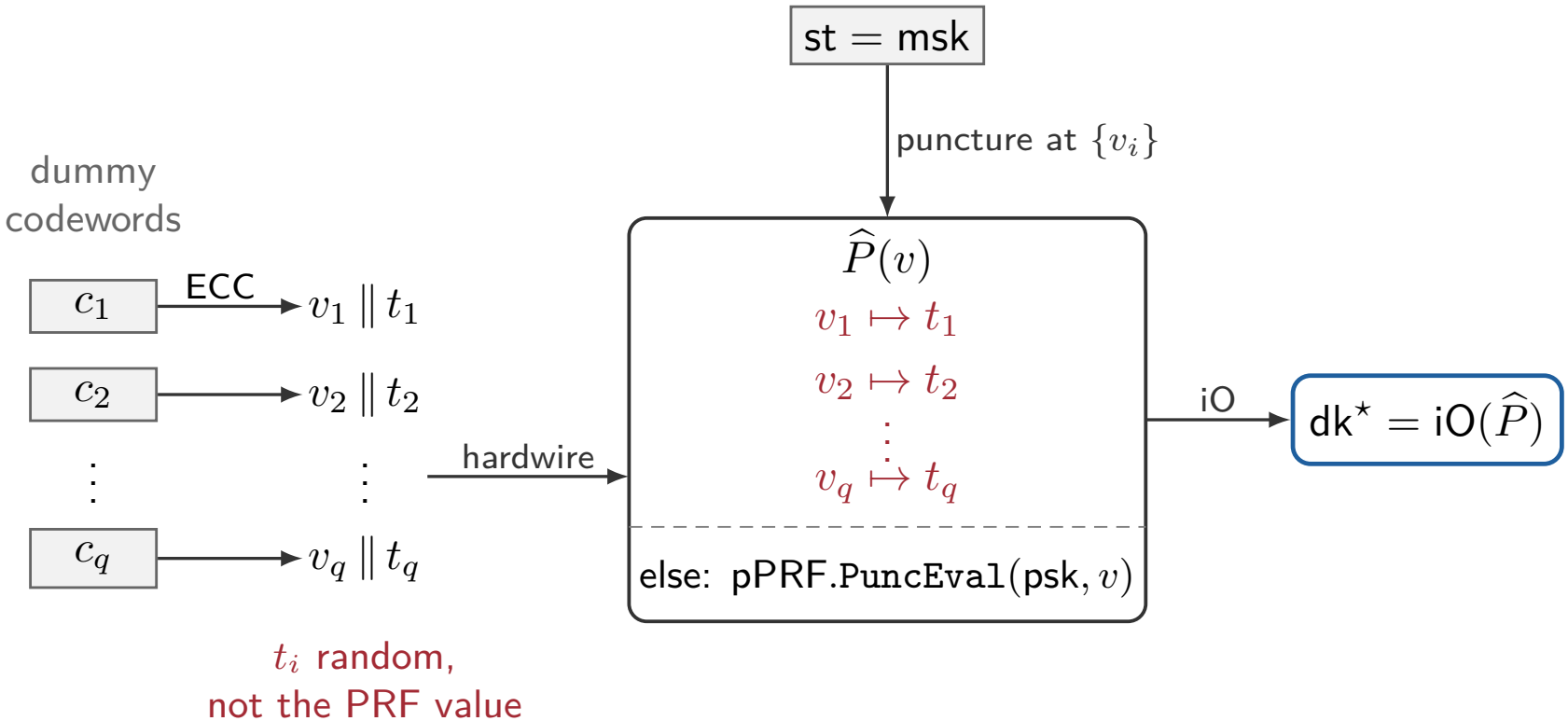
# A Heuristic NC-PRC: Opening the Decoder



# A Heuristic NC-PRC: Opening the Decoder



# A Heuristic NC-PRC: Opening the Decoder






$c_i$ and its $\delta$-ball

ECC.Decode → $v_i \| t_i$ → $\widehat{P}(v_i) = t_i$ → ✓ **accepted**

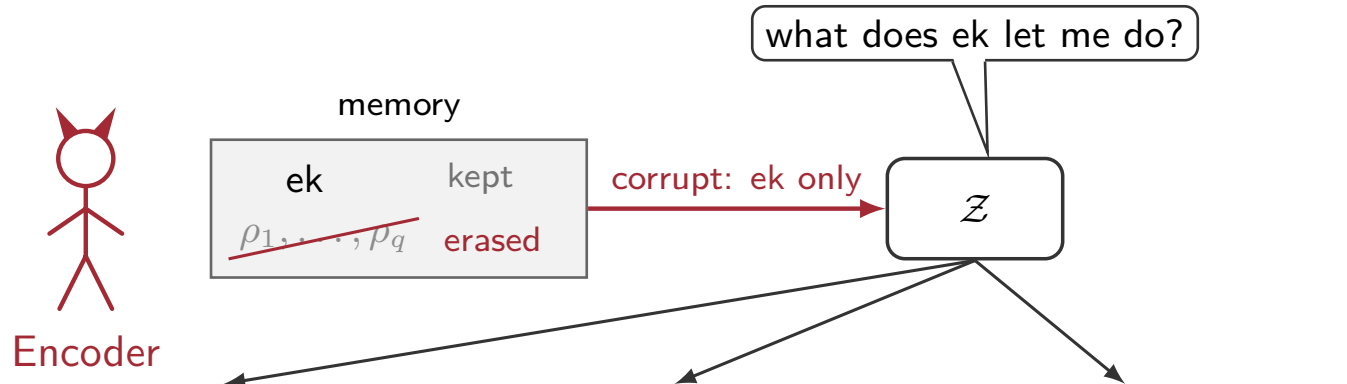
$\mathsf{dk}^\star = \mathsf{iO}(\widehat{P})$ ≈ $\mathsf{dk} = P$

iO + pPRF security at the punctured $\{v_i\}$

# Quick Take-Away

1. **Non-adaptive corruption**: UC $\Longleftrightarrow$ Standalone Security
2. **Adaptive corruption**: non-committing PRC is necessary.
3. LDPC-based PRCs do NOT admit a non-committing PRC.
4. Non-committing PRCs exist (heuristically).

# Encoder Corruption and Erasure




| **SK,** ek $=$ dk (this work) | **SK,** ek $\neq$ dk (LDPC [CG24][5]) | **PK** (ek is public) |
|---|---|---|
| `Decode` ✓ | `Encode` ✓ `Decode` × | $\mathcal{Z}$ had ek already |
| decoder exposure, explained by `OpenDec` | past $c_i$ still look random | nothing new |

**Without erasure:**

$\rho_i$ kept $\xrightarrow{\mathcal{Z}}$ $\texttt{Encode}(\text{ek}, m_i; \rho_i) \stackrel{?}{=} c_i$ × for a dummy

**all three cases** need opened coins: $\rho_i^*$ with $c_i = \texttt{Encode}(\text{ek}, m_i; \rho_i^*)$

our impossibility uses decoder corruption only, so it holds either way

[5] M. Christ and S. Gunn. "Pseudorandom Error-Correcting Codes". *CRYPTO 2024, Part VI*. vol. 14925. LNCS. 2024. DOI: 10.1007/978-3-031-68391-6_10

# A Secret-Key NC-PRC (Algorithms)

- $\mathtt{KGen}(1^\lambda)$:
  1. $\mathsf{msk} \leftarrow \mathsf{pPRF}.\mathtt{Setup}(1^\lambda)$, $P := \mathsf{iO}\big(\mathsf{pPRF}.\mathtt{Eval}(\mathsf{msk}, \cdot)\big)$.
  2. $\mathsf{ek} := \mathsf{dk} := P$.
- $\mathtt{Encode}(\mathsf{ek}, 1)$ (zero-bit):
  1. Sample $v \leftarrow\!\!\$\ \mathcal{V}$ and noise $e \leftarrow\!\!\$\ \chi$.
  2. Set $t \leftarrow P(v)$ and output $c \leftarrow \mathsf{ECC}.\mathtt{Encode}(v \| t) \oplus e$.
- $\mathtt{Decode}(\mathsf{dk}, x)$:
  1. $v \| t \leftarrow \mathsf{ECC}.\mathtt{Decode}(x)$.
  2. If ECC fails, output $\perp$.
  3. If $t = P(v)$, output $1$; otherwise output $\perp$.

# A Secret-Key NC-PRC (Opening Algorithm)

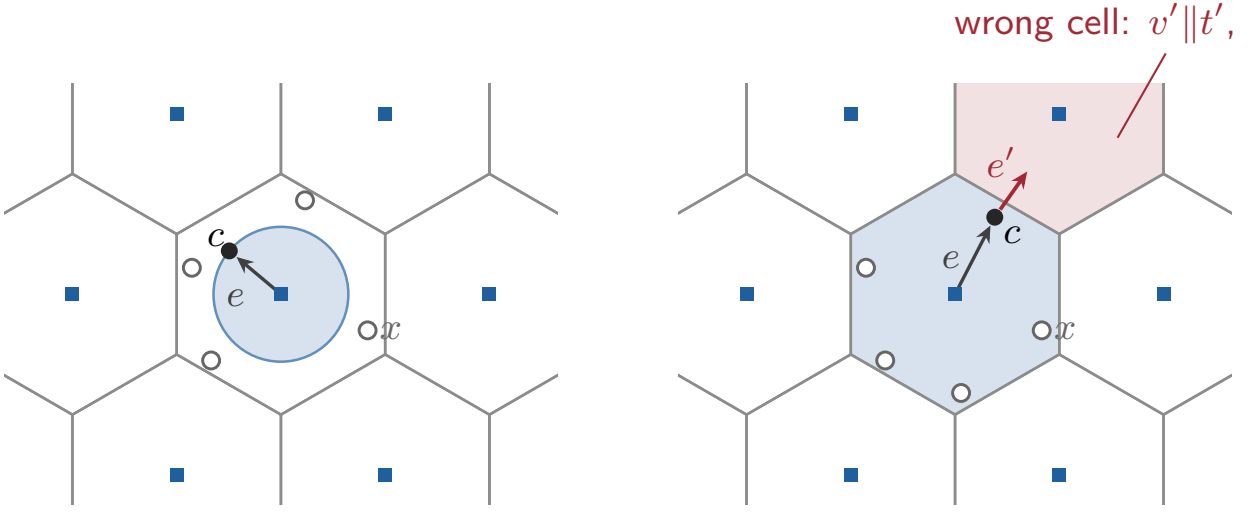
– $\mathtt{OpenDec}(\mathsf{st} := \mathsf{msk}, \mathcal{T} := \{(m_i, c_i)\}_{i=1}^{q})$:

1. For $i \in [q]$, $v_i \| t_i \leftarrow \mathsf{ECC}.\mathtt{Decode}(c_i)$, skipping $c_i$ if ECC fails.
2. $\mathsf{psk} \leftarrow \mathsf{pPRF}.\mathtt{Puncture}(\mathsf{msk}, \{v_i\}_{i\in[q]})$, $P^\star := \mathsf{iO}(\widehat{P})$ where

$$\widehat{P}(v) := \begin{cases} t_i & \text{if } v = v_i \text{ for some } i \in [q], \\ \mathsf{pPRF}.\mathtt{PuncEval}(\mathsf{psk}, v) & \text{otherwise.} \end{cases}$$

3. Output $\mathsf{dk}^\star := P^\star$.

# Why the Coding Layer Is Idealized




**(a) small noise**
robust ✓ pseudorandom ×
$c$ stays near the center; a uniform $x$ almost never does

**(b) noise fills the cell**
pseudorandom ✓ robust ×
$c$ sits near an edge, like a uniform $x$; one more error $e'$ crosses into the next cell

The code is *public*: anyone can measure the distance to the nearest center.
Way out: a *secret* code (LPN; a PRP per symbol over $2^\lambda$ alphabets), or an idealized code (this work).

# Decoder Openings for Known PRCs

| | key | openable dummies |
|---|---|---|
| **Acceptance fixed by the public key** a uniform dummy codeword is rejected under every key | | |
| Distorted McEliece [DFV26] | PK | $\times$ no $Q \geq 1$ |
| Noisy McEliece [DJK+26] | PK | $\times$ no $Q \geq 1$ |
| Planted hyperloop [GG25] | PK | $\times$ no $Q \geq 1$ |
| **Short or sparse secret** too few keys to explain many dummy codewords | | |
| LDPC [CG24], PRC$^\sharp$ [AAC+25] | PK/SK | $\times$ large $Q$ (TLOC, this work) |
| Space-bounded SSR [GG25] | SK | $\times$ large $Q$ (sparse checks) |
| Permuted codes, binary [CGG+25] | SK | $\times$ $Q = \omega(\log n)$ (only $n!$ keys) |
| SKE + RS + symbol-wise PRP [DMR25] | SK | $\times$ no $Q \geq 1$ ($\lambda$-bit PRP key) |
| **Programmable tables in the key** equivocation without a coding hypothesis | | |
| Local weak PRF [GM24] | SK | ✓ $Q = 1$ (program the mask $z$) |
| Permuted codes, large $\Sigma$ [CGG+25] | SK | ✓ $Q^2 n \ll \lvert\Sigma\rvert$ (program each $\pi_i$) |
| SKE + RS + iO-programmed PRP [DMR25] | SK | ✓ any bounded $Q$ (heuristic, as ours) |

our sketches, not theorems

# A Public-Key NC-PRC

Error-Correcting Code

$+$

Smooth Projective Hash Function (SPHF)

# A Public-Key NC-PRC

Let $L := \{u \in U : \exists\, w \in W : (u, w) \in \mathcal{R}\}$

- Sparsity

$$\Pr_{u \leftarrow\$ U}[u \in L] \leq \mathsf{negl}(\lambda)$$

- Subset Membership Indistinguishability

$$\left| \Pr_{(u,w) \leftarrow\$ \mathcal{R}}\left[\mathcal{A}(1^\lambda, u) = 1\right] - \Pr_{u \leftarrow\$ U}\left[\mathcal{A}(1^\lambda, u) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

- Correctness

$$\Pr[\mathsf{SPHF.Hash}(\mathsf{hk}, u) \neq \mathsf{SPHF.Proj}(\mathsf{hp}, u, w)] \leq \mathsf{negl}(\lambda)$$

# A Public-Key NC-PRC

- $\texttt{KGen}(1^\lambda)$:
  1. $\mathsf{hk} \leftarrow \mathsf{SPHF}.\texttt{HashKG}(1^\lambda)$, $\mathsf{hp} \leftarrow \mathsf{SPHF}.\texttt{ProjKG}(\mathsf{hk})$.
  2. Output $\mathsf{ek} := \mathsf{hp}$ and $\mathsf{dk} := \mathsf{hk}$.
- $\texttt{Encode}(\mathsf{ek}, 1)$ (zero-bit):
  1. Sample $(u, w) \leftarrow\!\!\$\, U \times W$ such that $(u, w) \in \mathcal{R}$ and noise $e \leftarrow\!\!\$\, \chi$.
  2. Set $h \leftarrow \mathsf{SPHF}.\texttt{Proj}(\mathsf{hp}, u, w)$
  3. Output $c \leftarrow \mathsf{ECC}.\texttt{Encode}(u \| h) \oplus e$.
- $\texttt{Decode}(\mathsf{dk}, x)$:
  1. Compute $u \| h \leftarrow \mathsf{ECC}.\texttt{Decode}(x)$.
  2. If ECC fails, output $\perp$.
  3. If $h = \mathsf{SPHF}.\texttt{Hash}(\mathsf{hk}, u)$, output $1$; otherwise output $\perp$.

# A Public-Key NC-PRC

– $\texttt{OpenDec}(\mathsf{st} := (\mathsf{hk}, \mathsf{hp}), \mathcal{T} := \{(m_i, c_i)\}_{i=1}^{q})$:

1. For $i \in [q]$, recover $u_i \| h_i \leftarrow \mathsf{ECC.Decode}(c_i)$, skipping $c_i$ if ECC fails or $u_i \in L$.
2. Sample $\mathsf{hk}^\star$ uniformly from

$$\mathcal{K}^\star := \left\{ \mathsf{hk}' : \begin{array}{l} \mathsf{SPHF.ProjKG}(\mathsf{hk}') = \mathsf{hp} \wedge \\ \forall i \in [q],\ \mathsf{SPHF.Hash}(\mathsf{hk}', u_i) = h_i \end{array} \right\}.$$

3. Output $\mathsf{dk}^\star := \mathsf{hk}^\star$; if $\mathcal{K}^\star = \varnothing$, output $\mathsf{dk}^\star := \mathsf{hk}$.