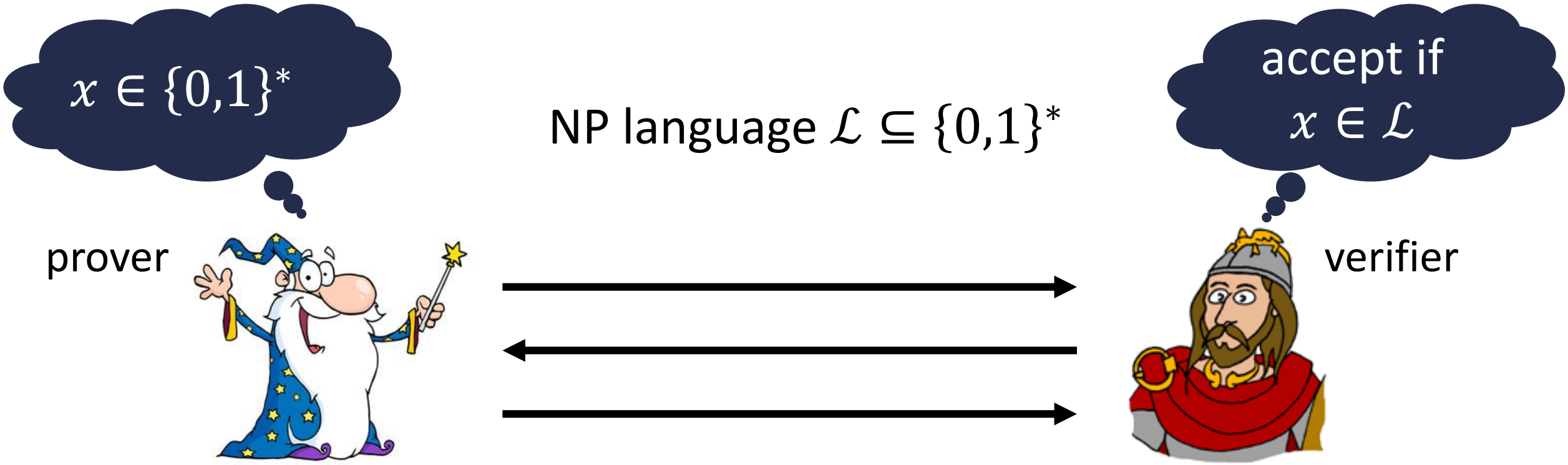
New Constructions of Reusable Designated-Verifier NIZKs

Alex Lombardi, Willy Quach, Ron D. Rothblum,
Daniel Wichs, and David J. Wu

Proof Systems and Argument Systems

[GMR85]



Completeness: $\forall x \in \mathcal{L} : \Pr[\langle P, V \rangle(x) = \text{accept}] = 1$

"Honest prover convinces honest verifier of true statements"

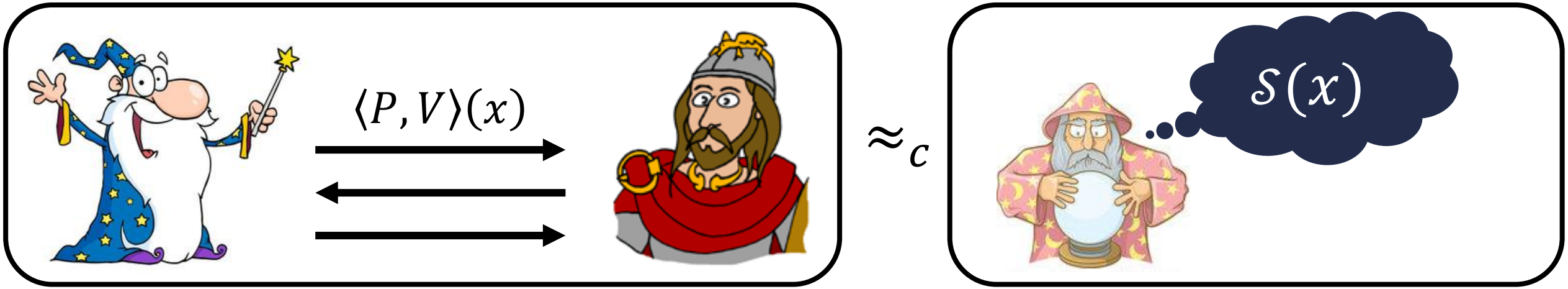
(Computational) Soundness: $\forall x \notin \mathcal{L},\ \forall \text{ efficient } P^* : \Pr[\langle P^*, V \rangle(x) = \text{accept}] \leq \varepsilon$

"No efficient prover can convince honest verifier of false statement"

Zero-Knowledge Arguments for NP

[GMR85]

NP language $\mathcal{L}$



real distribution

ideal distribution

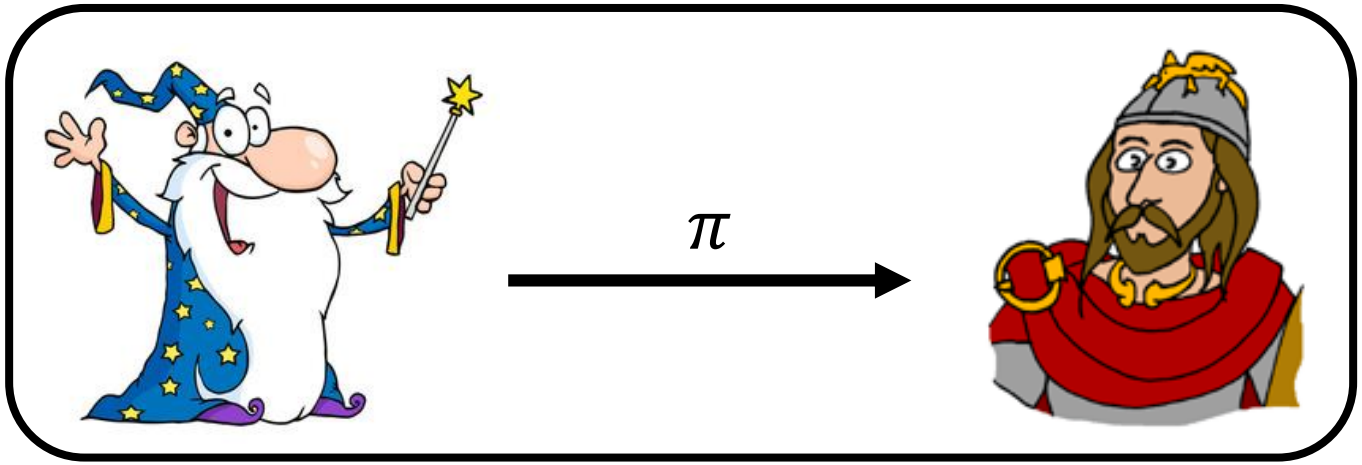
Zero-Knowledge: for all efficient verifiers V^*, there exists an efficient simulator $\mathcal{S}$ such that:

$$\forall x \in \mathcal{L} : \langle P, V^* \rangle(x) \approx_c \mathcal{S}(x)$$

Non-Interactive Zero-Knowledge (NIZK)

[BFM88]

NP language $\mathcal{L}$



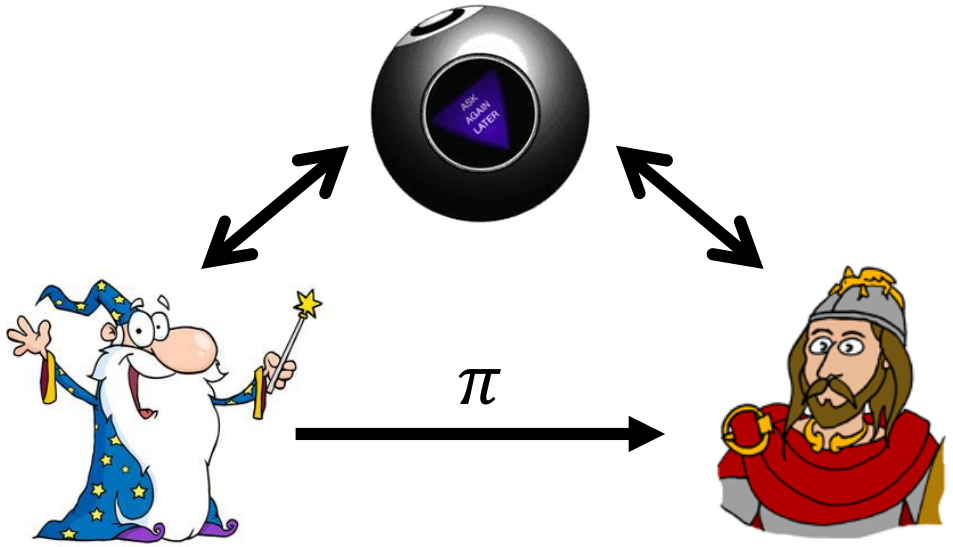
$\approx_c$

real distribution

ideal distribution

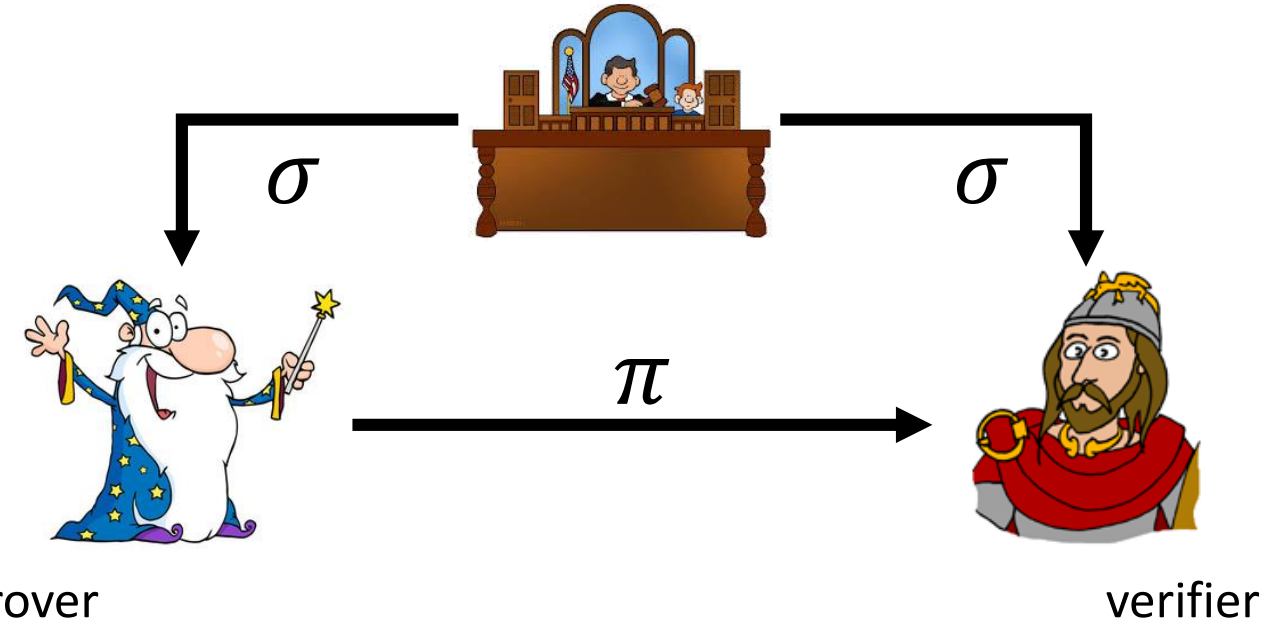
In the standard model, this is only achievable for languages $\mathcal{L} \in \mathrm{BPP}$

Which Assumptions give NIZKs for NP?



Random Oracle Model

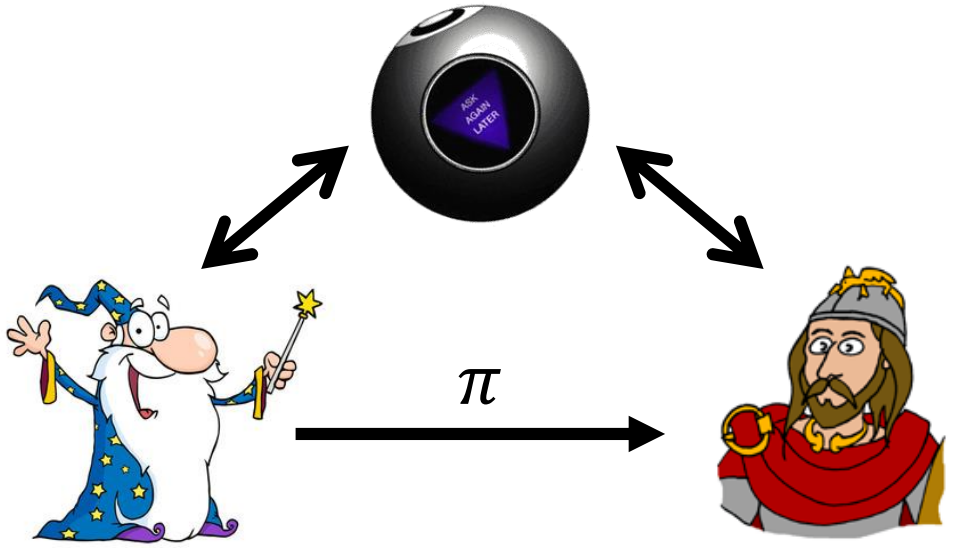
[FS86, PS96]



Common Reference String (CRS) Model

- Quadratic Residuosity [BFM88, DMP87, BDMP91]
- Trapdoor Permutations [FLS90, DDO+01, Gro10]
- Pairings [GOS06]
- Learning with Errors [PS19]

Which Assumptions give NIZKs for NP?



prover verifier

Random Oracle Model
[FS86, PS96]

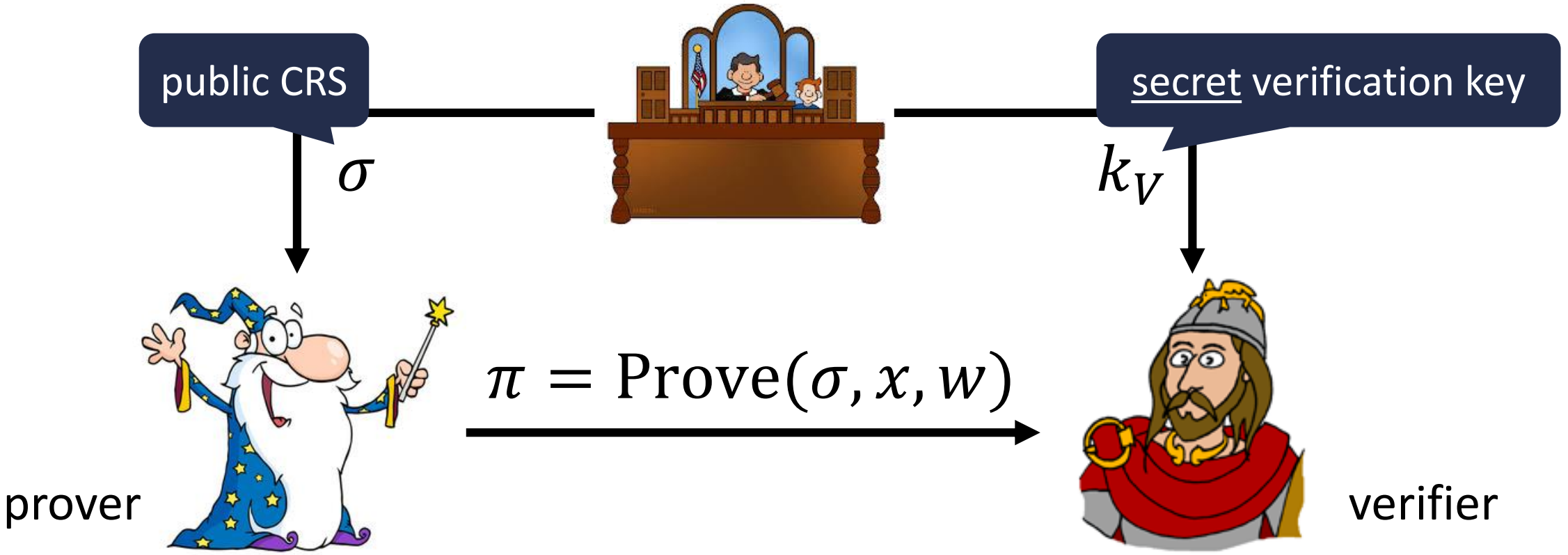
- **Many different approaches:** hidden-bits model, correlation-intractability, homomorphic commitments
- **Some assumptions still missing:** Diffie-Hellman assumptions, learning parity with noise (LPN)

Common Reference String (CRS) Model

- Quadratic Residuosity [BFM88, DMP87, BDMP91]
- Trapdoor Permutations [FLS90, DDO+01, Gro10]
- Pairings [GOS06]
- Learning with Errors [PS19]

Designated-Verifier NIZKs

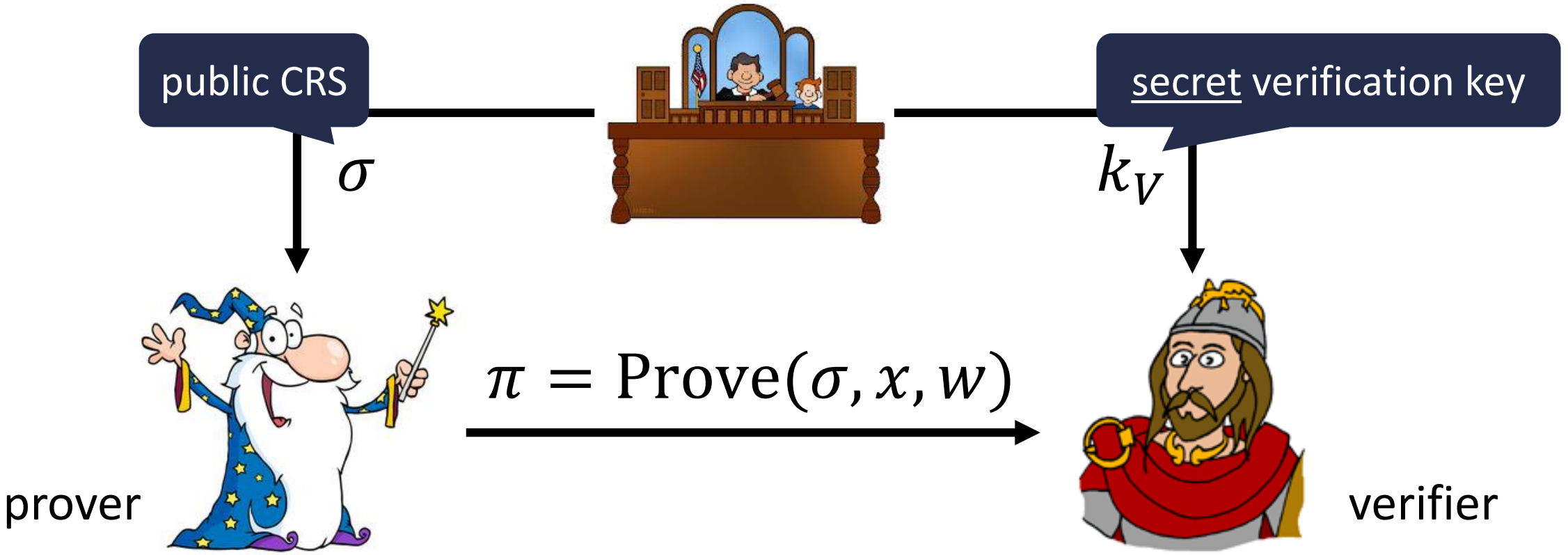
Is there a general framework for constructing NIZKs?



This work: focus on the designated-verifier model

Designated-Verifier NIZKs

Is there a general framework for constructing NIZKs?



Requirement: soundness should hold even if the prover has access to the verification oracle

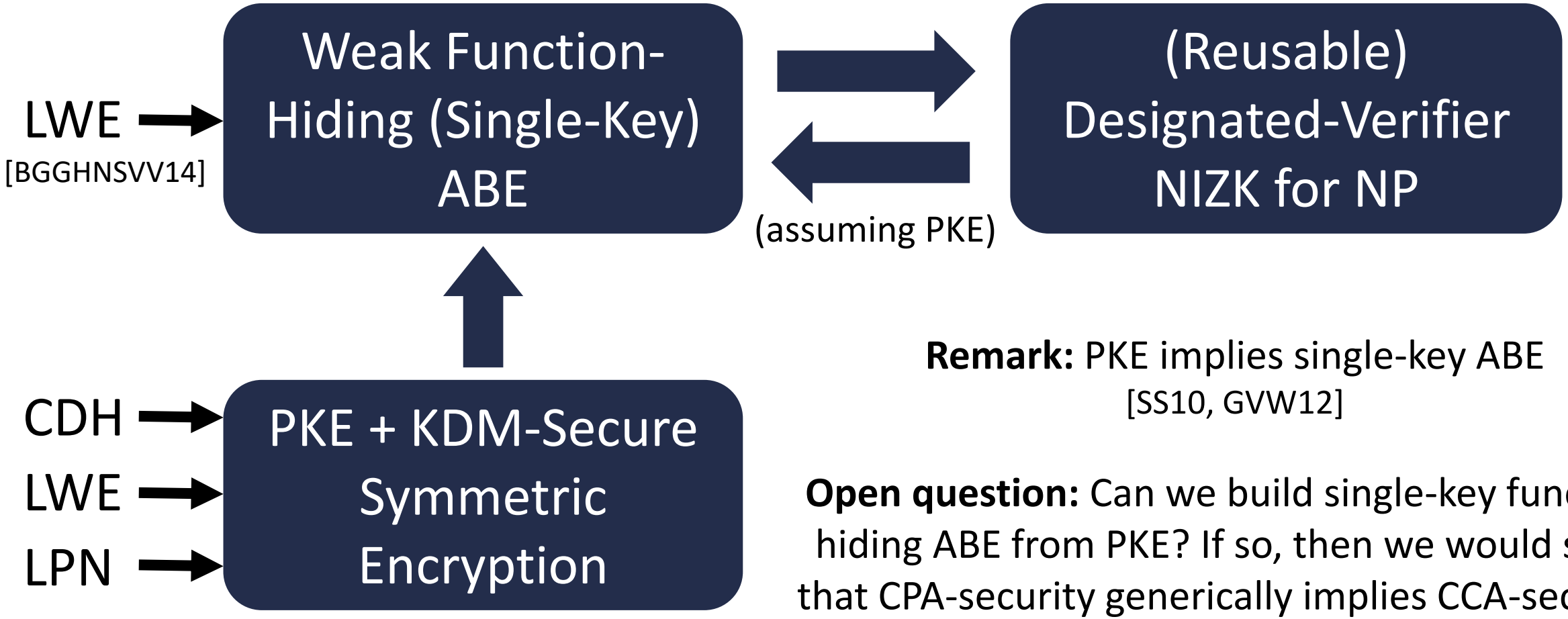
Why Designated-Verifier NIZKs?

Is there a general framework for constructing DV-NIZKs?

- Sufficient to instantiate many classic applications of NIZKs (e.g., CPA-security to CCA-security, boosting security of MPC protocols, etc.)
- Non-trivial relaxation of standard NIZKs:
  - **Until very recently:** instantiations of reusable DV-NIZKs are all NIZK constructions in standard CRS model
  - **Recently:** DV-NIZKs based on CDH/DDH (specific algebraic instantiation of the hidden-bits model) [CH19, KNYY19, QRW19]
  - **Non-reusable DV-NIZKs:** known from homomorphic encryption [DFN06], public-key encryption [PsV06]

Our Results

Is there a general framework for constructing DV-NIZKs?

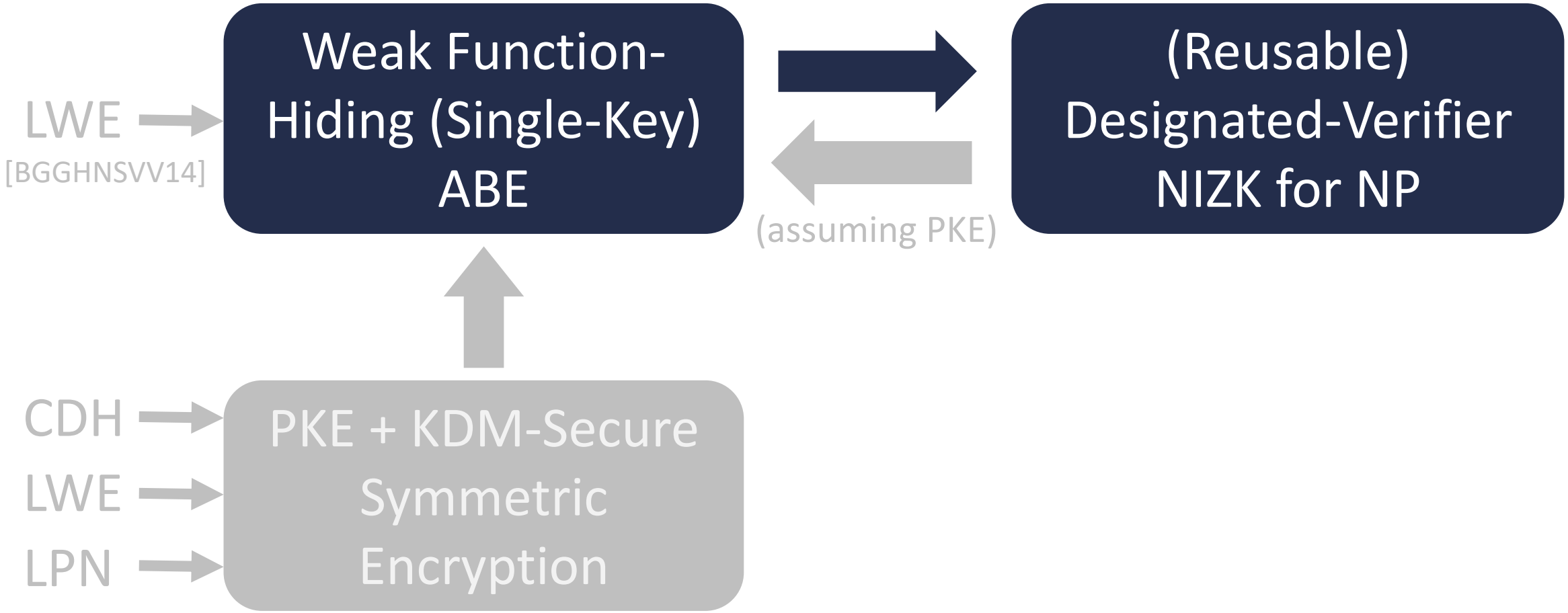


Remark: PKE implies single-key ABE
[SS10, GVW12]

Open question: Can we build single-key function-hiding ABE from PKE? If so, then we would show that CPA-security generically implies CCA-security!

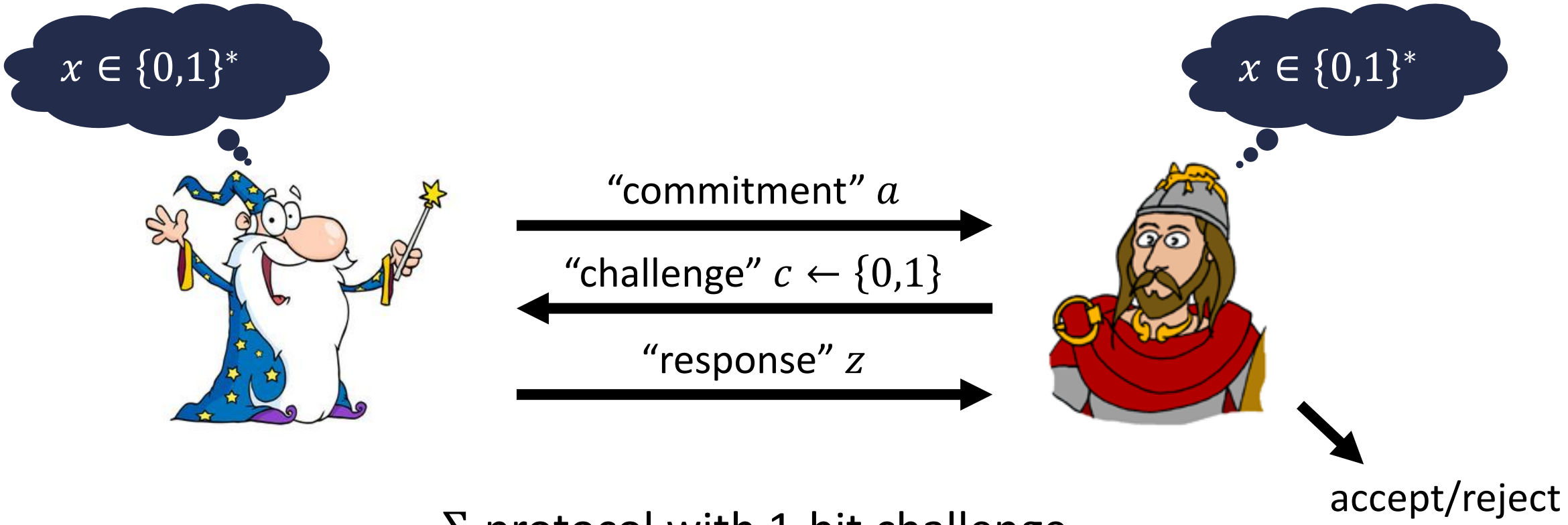
Part I: From ABE to DV-NIZK

Is there a general framework for constructing DV-NIZKs?



Starting Point: A Non-Reusable DV-NIZK

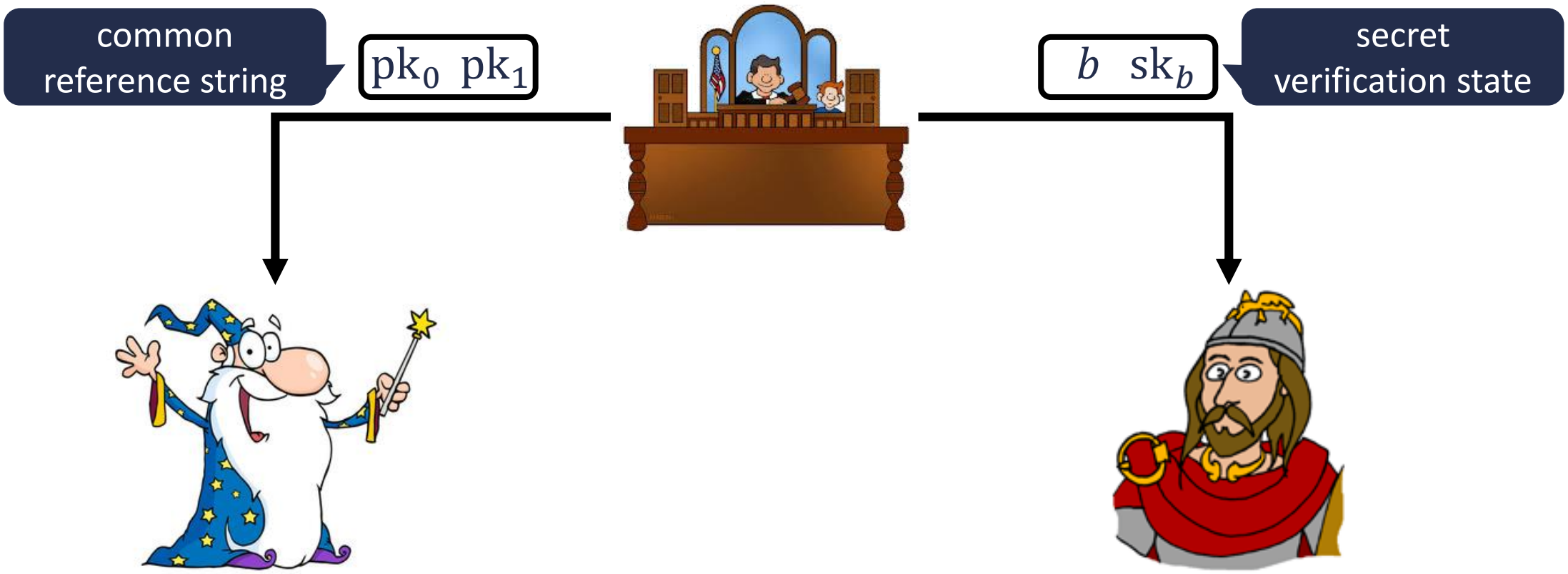
[PsV06]



Σ-protocol with 1-bit challenge
(e.g., Blum's protocol for graph Hamiltonicity [Blu86])

Starting Point: A Non-Reusable DV-NIZK

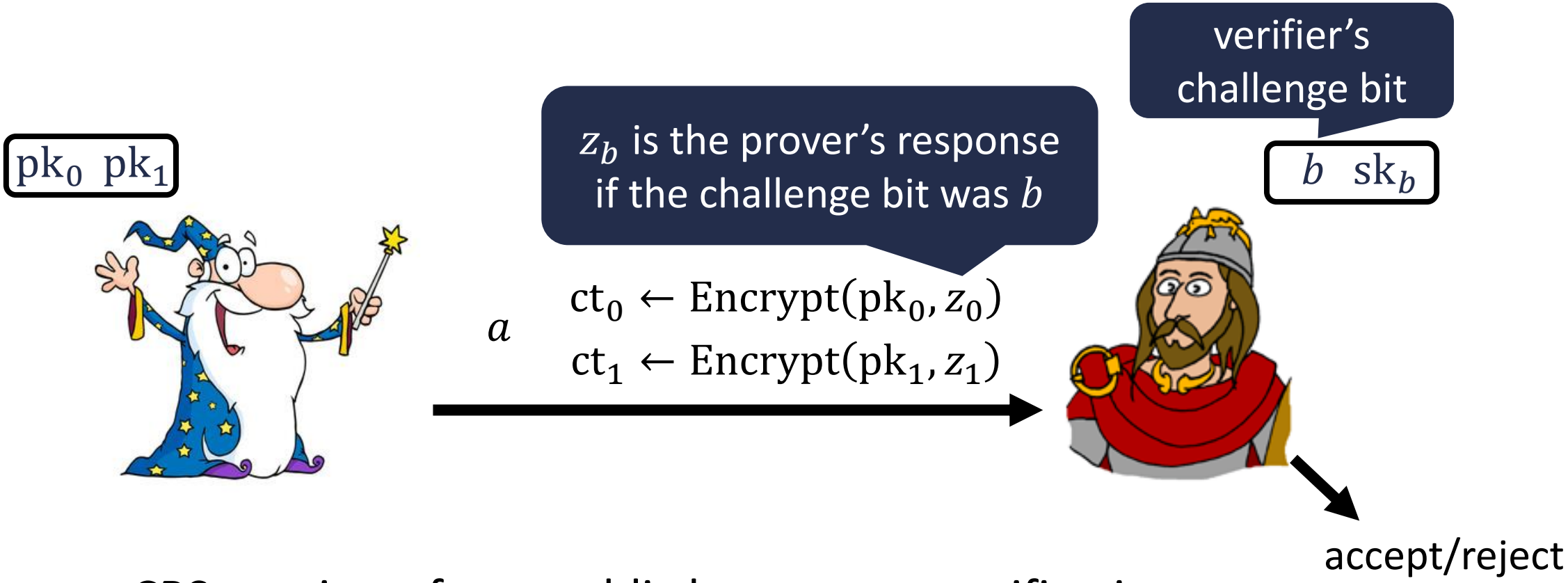
[PsV06]



CRS consists of two public keys, secret verification state consists of one of the secret keys (unknown to the prover)

Starting Point: A Non-Reusable DV-NIZK

[PsV06]



CRS consists of two public keys, secret verification state consists of one of the secret keys (unknown to the prover)

Starting Point: A Non-Reusable DV-NIZK

[PsV06]

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

$a_1, \ldots, a_n$

$$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$$
$$\vdots \quad \vdots$$
$$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$$

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

Repeat the protocol n times in parallel to amplify soundness

Starting Point: A Non-Reusable DV-NIZK

[PsV06]

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

$a_1, \ldots, a_n$

$$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$$
$$\vdots \quad \vdots$$
$$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$$

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

Zero-knowledge: follows by semantic-security of PKE scheme and special zero-knowledge of the Σ-protocol

Starting Point: A Non-Reusable DV-NIZK

[PsV06]

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

$$a_1, \ldots, a_n$$

$$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$$
$$\vdots \quad \vdots$$
$$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$$

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

(One-time) soundness: verifier's challenges are uniformly random and perfectly hidden from the prover, so soundness reduces to that of the Σ-protocol

Starting Point: A Non-Reusable DV-NIZK

[PsV06]

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

$a_1, \ldots, a_n$

$$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$$
$$\vdots \quad \vdots$$
$$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$$

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

Does <u>not</u> provide reusable soundness!

Starting Point: A Non-Reusable DV-NIZK

[PsV06]

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

Verifier rejection attack: take valid proof and perturb it

If verifier accepts, then $b_1 = 1$, and else, $b_1 = 0$

$$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$$
$$\vdots \quad \vdots$$
$$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$$

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

Recover verifier's challenge bit-by-bit!

Does not provide reusable soundness!

Starting Point: A Non-Reusable DV-NIZK

[PsV06]

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

Verifier rejection attack: take valid proof and perturb it

If verifier accepts, then $b_1 = 1$, and else, $b_1 = 0$

$$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$$
$$\vdots \quad \vdots$$
$$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$$

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

Recover verifier's challenge bit-by-bit!

Problem: Verifier uses same randomness to verify all proofs

Our Compiler: ABE ⇒ DV-NIZK

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

Idea: Derive $b_1, \ldots, b_n$ from the statement x

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

$a_1, \ldots, a_n$

$$\begin{matrix} \mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)} \\ \vdots \quad \vdots \\ \mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)} \end{matrix}$$

Key idea: Use <u>independent</u> randomness to verify each statement x

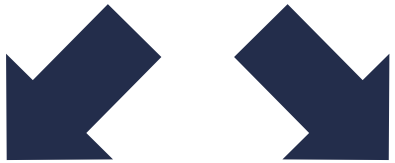
Core Ingredient: Attribute-Based Encryption (ABE)

[SW05, GPSW06]

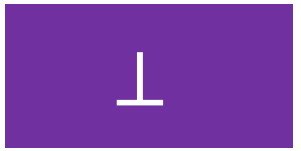
Public-key attribute-based encryption (ABE):

Ciphertexts associated with **public** attribute $x \in \mathcal{X}$ and message m

Secret keys associated with function $f : \mathcal{X} \to \{0,1\}$



if $f(x) = 1$



if $f(x) = 0$

Semantic security: ct hides message m if $f(x) = 0$

Our Compiler: ABE ⇒ DV-NIZK

$$\begin{matrix} \mathrm{pk}_0^{(1)} & \mathrm{pk}_1^{(1)} \\ \vdots & \vdots \\ \mathrm{pk}_0^{(n)} & \mathrm{pk}_1^{(n)} \end{matrix}$$

Idea: Derive $b_1, \ldots, b_n$ from the statement x

$$\begin{matrix} b_1 & \mathrm{sk}_{b_1}^{(1)} \\ \vdots & \vdots \\ b_n & \mathrm{sk}_{b_n}^{(n)} \end{matrix}$$

$a_1, \ldots, a_n$

$$\begin{matrix} \mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)} \\ \vdots \quad \vdots \\ \mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)} \end{matrix}$$

Key idea: Use <u>independent</u> randomness to verify each statement x

Our Compiler: ABE ⇒ DV-NIZK

mpk

$a_1, \ldots, a_n$

$\mathrm{ct}_0^{(1)}, \mathrm{ct}_1^{(1)}$

$\vdots \quad \vdots$

$\mathrm{ct}_0^{(n)}, \mathrm{ct}_1^{(n)}$

$b_1 \quad \mathrm{sk}_{b_1}^{(1)}$

$\vdots \quad \vdots$

$b_n \quad \mathrm{sk}_{b_n}^{(n)}$

Key idea: Use independent randomness to verify each statement x

Our Compiler: ABE ⇒ DV-NIZK

mpk

$a_1, \ldots, a_n$

$(1,0)$ $z_0^{(1)}$ $\mathrm{ct}_0^{(1)}$ $(1,1)$ $z_1^{(1)}$ $\mathrm{ct}_1^{(1)}$

$(n,0)$ $z_0^{(n)}$ $\mathrm{ct}_0^{(n)}$ $(n,1)$ $z_1^{(n)}$ $\mathrm{ct}_1^{(n)}$

b_1 $\mathrm{sk}_{b_1}^{(1)}$

b_n $\mathrm{sk}_{b_n}^{(n)}$

Encrypt responses with attribute (index, bit)

Our Compiler: ABE ⇒ DV-NIZK

$a_1, \ldots, a_n$

mpk

$(1,0)$ $z_0^{(1)}$ $\mathrm{ct}_0^{(1)}$ $(1,1)$ $z_1^{(1)}$ $\mathrm{ct}_1^{(1)}$

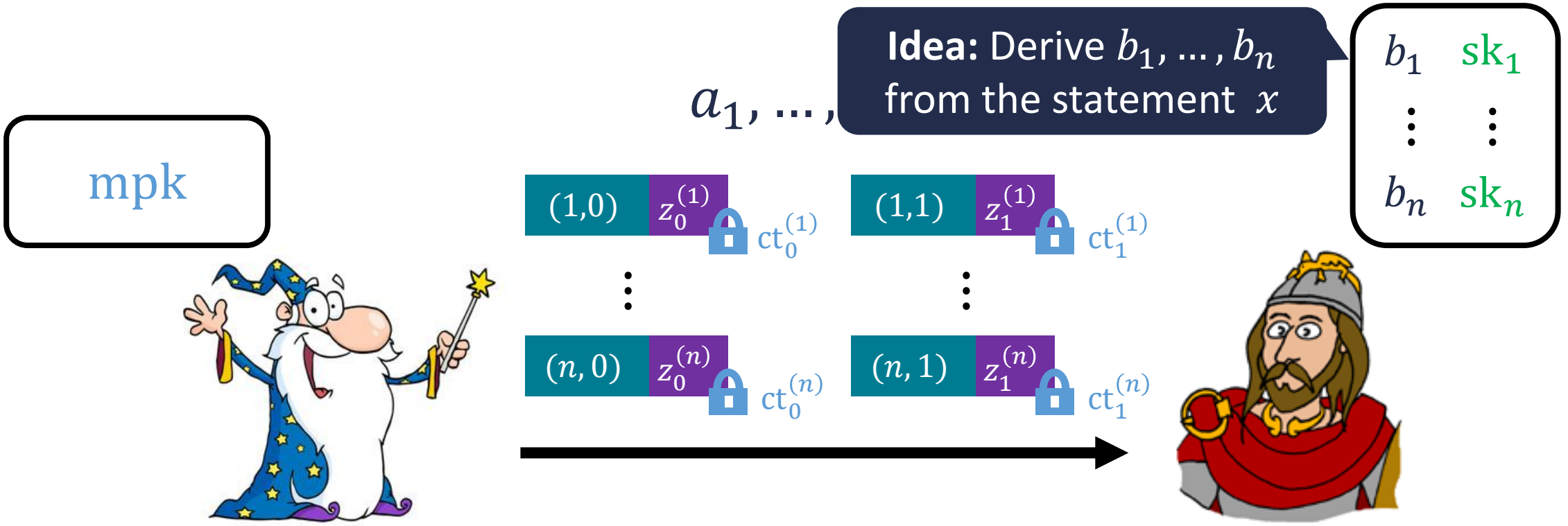
$(n,0)$ $z_0^{(n)}$ $\mathrm{ct}_0^{(n)}$ $(n,1)$ $z_1^{(n)}$ $\mathrm{ct}_1^{(n)}$

b_1	sk_1
⋮	⋮
b_n	sk_n

Decrypts ciphertext with attribute (j, b_j)

sk_j: key for function $f_j(i,b) = \begin{cases} 1 & \text{if } i = j \text{ and } b = b_j \\ 0 & \text{otherwise} \end{cases}$

Our Compiler: ABE ⇒ DV-NIZK



Key idea: Use <u>independent</u> randomness to verify each statement x

Our Compiler: ABE ⇒ DV-NIZK

Derive $b_i^{(x)}$ by computing $\mathrm{PRF}\big(k,(x,i)\big)$

mpk

$(x,1,0)$ $z_0^{(1)}$ $\mathrm{ct}_0^{(1)}$ $\quad$ $(x,1,1)$ $z_1^{(1)}$ $\mathrm{ct}_1^{(1)}$

$(x,n,0)$ $z_0^{(n)}$ $\mathrm{ct}_0^{(n)}$ $\quad$ $(x,n,1)$ $z_1^{(n)}$ $\mathrm{ct}_1^{(n)}$

$b_1^{(x)}$ sk_1

$b_n^{(x)}$ sk_n

sk_j: key for function $f_j(x,i,b) = \begin{cases} 1 & \text{if } i=j \text{ and } b = \mathrm{PRF}\big(k,(x,i)\big) \\ 0 & \text{otherwise} \end{cases}$

Our Compiler: ABE ⇒ DV-NIZK

Derive $b_i^{(x)}$ by computing $\mathrm{PRF}(k,(x,i))$

k | sk_1 ⋮ sk_n

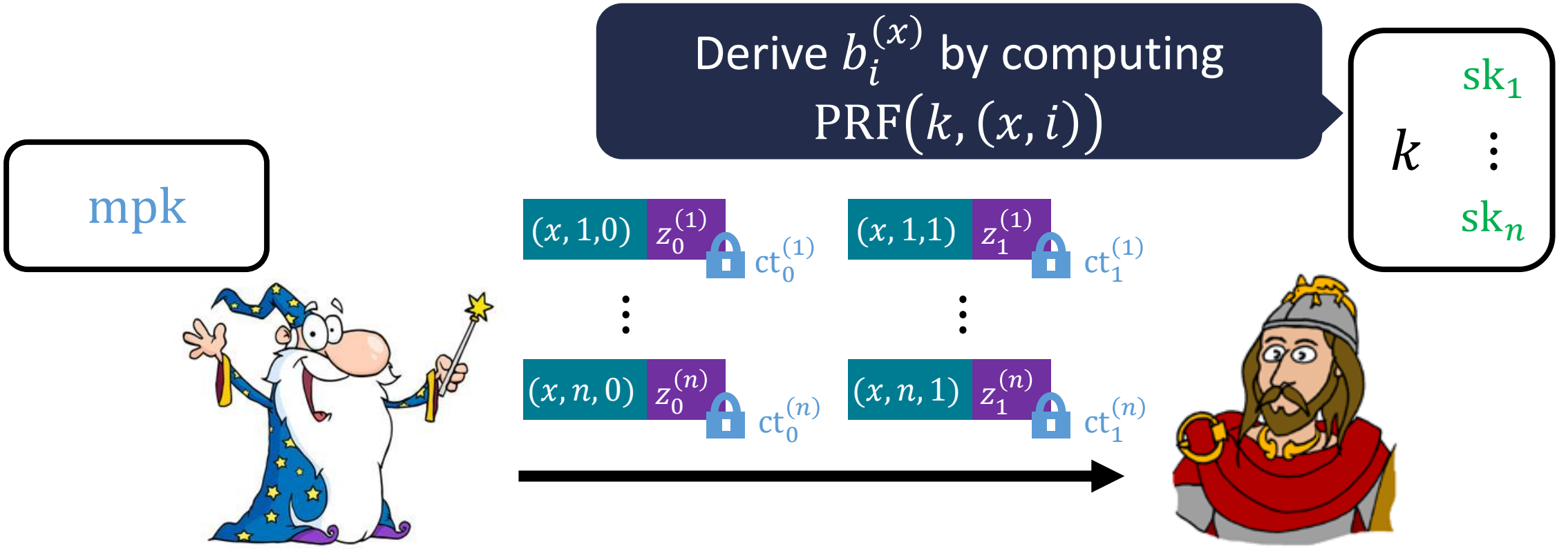
mpk

$(x,1,0)$ $z_0^{(1)}$ $\mathrm{ct}_0^{(1)}$ ⋮ $(x,n,0)$ $z_0^{(n)}$ $\mathrm{ct}_0^{(n)}$

$(x,1,1)$ $z_1^{(1)}$ $\mathrm{ct}_1^{(1)}$ ⋮ $(x,n,1)$ $z_1^{(n)}$ $\mathrm{ct}_1^{(n)}$

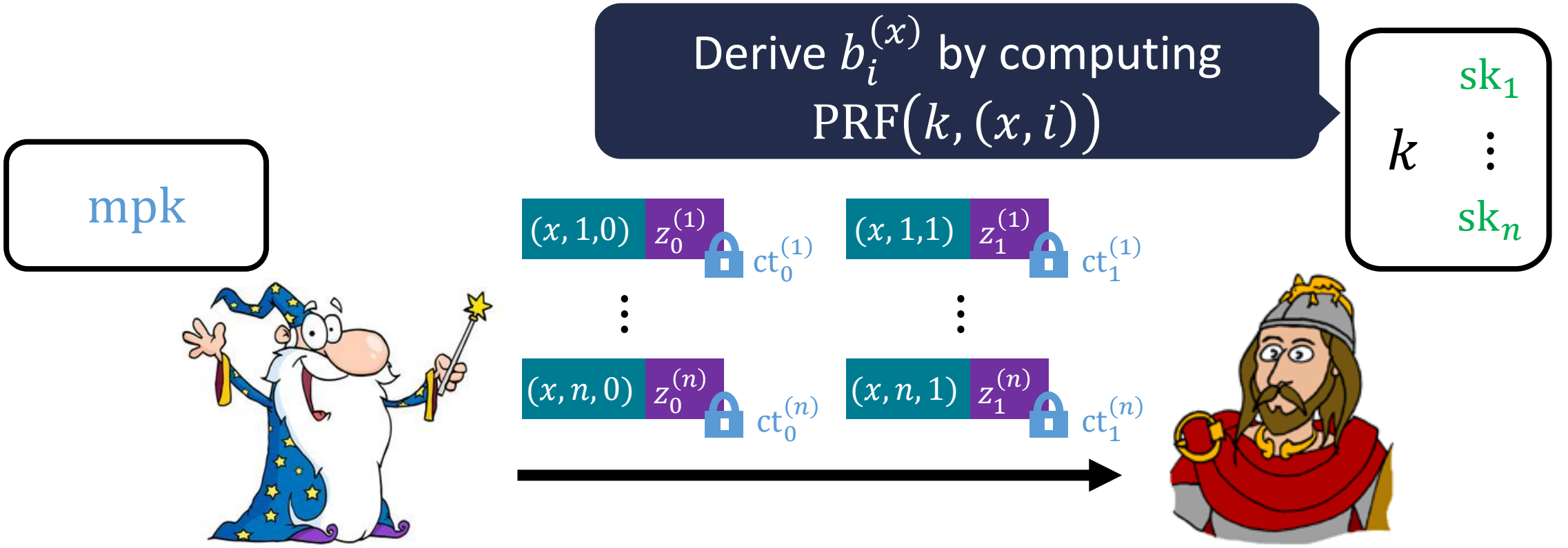
sk_j: key for function $f_j(x,i,b) = \begin{cases} 1 & \text{if } i=j \text{ and } b = \mathrm{PRF}(k,(x,i)) \\ 0 & \text{otherwise} \end{cases}$

Our Compiler: ABE ⇒ DV-NIZK



Zero-knowledge: Follows from semantic security of the ABE scheme and (special) ZK of underlying Sigma protocol: for all x, sk_i can decrypt exactly one of $\mathrm{ct}_0^{(i)}$ and $\mathrm{ct}_1^{(i)}$

Our Compiler: ABE ⇒ DV-NIZK



Soundness: Follows if verifier's queries are uniformly random and unknown to the adversary – intuitively should follow from security of the PRF (since prover does not see k)

Our Compiler: ABE ⇒ DV-NIZK

sk_1

k ⋮

$a_1, \ldots, a_n$

mpk

Problem: Prover gets to query the verifier (who would evaluate the ABE decryption function); ABE decryption could leak information about the secret decryption key which depends on the PRF key k

Soundness: Follows if verifier's queries are uniformly random and unknown to the adversary – intuitively should follow from security of the PRF (since prover does not see k)

Our Compiler: ABE ⇒ DV-NIZK

mpk

ct

$\text{Decrypt}(\text{sk}_f, \text{ct})$

$(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$

$\text{sk}_f \leftarrow \text{KeyGen}(\text{msk}, f)$

mpk

ct

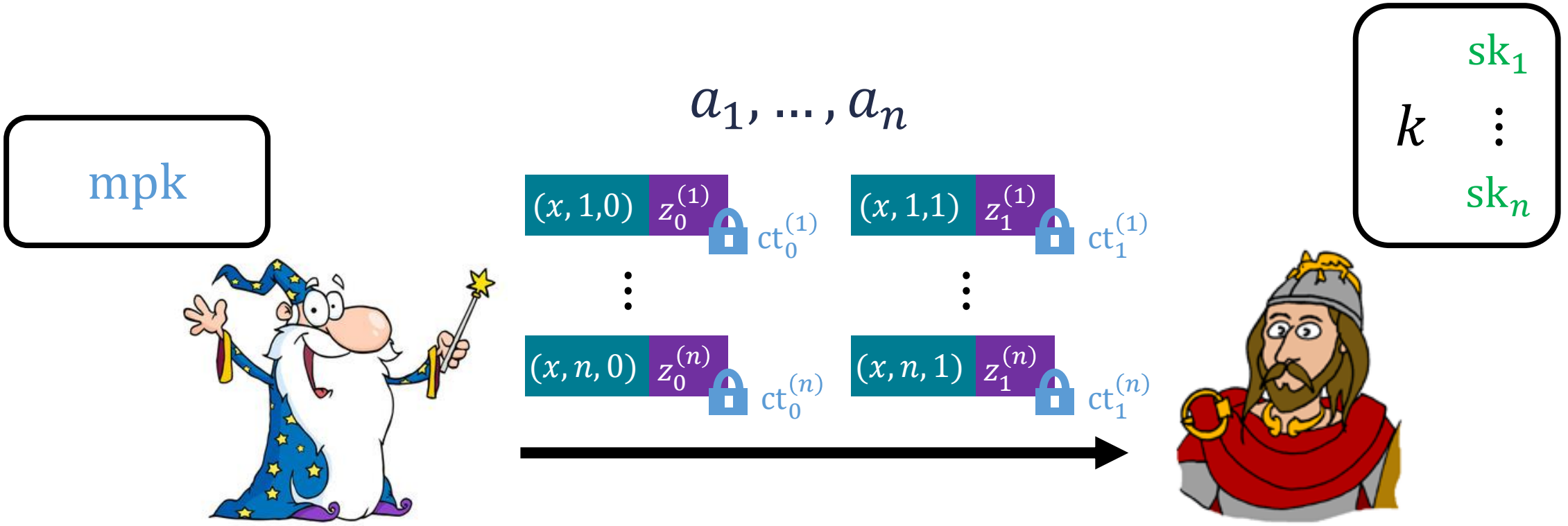
$\mathcal{S}^{f(\cdot)}(\text{msk}, \text{ct})$

$(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$

Decryption queries can be simulated given msk and oracle access to f (simulator can query f once per decryption query)

(Weak) Function-Privacy: Output of ABE decryption should hide the associated function

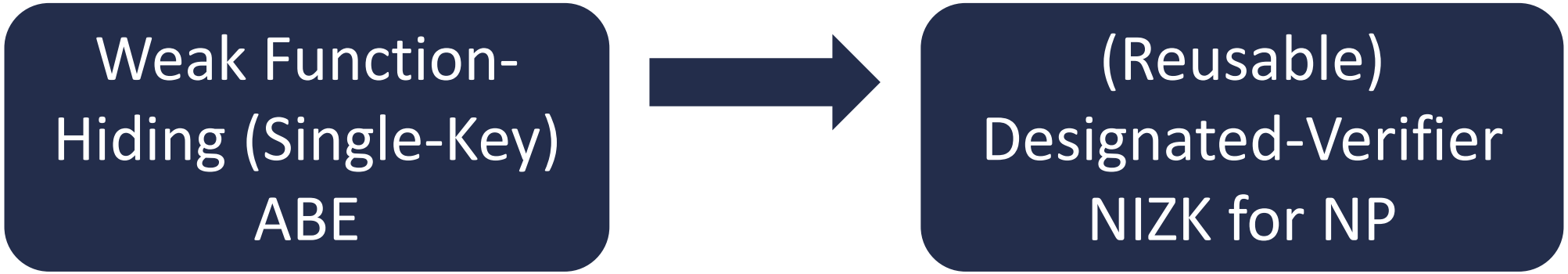
Our Compiler: ABE ⇒ DV-NIZK



If ABE scheme satisfies weak function-privacy, then decryption queries can be simulated just given oracle access to the PRF; soundness follows from PRF security and soundness of the Σ-protocol

Our Compiler: ABE ⇒ DV-NIZK

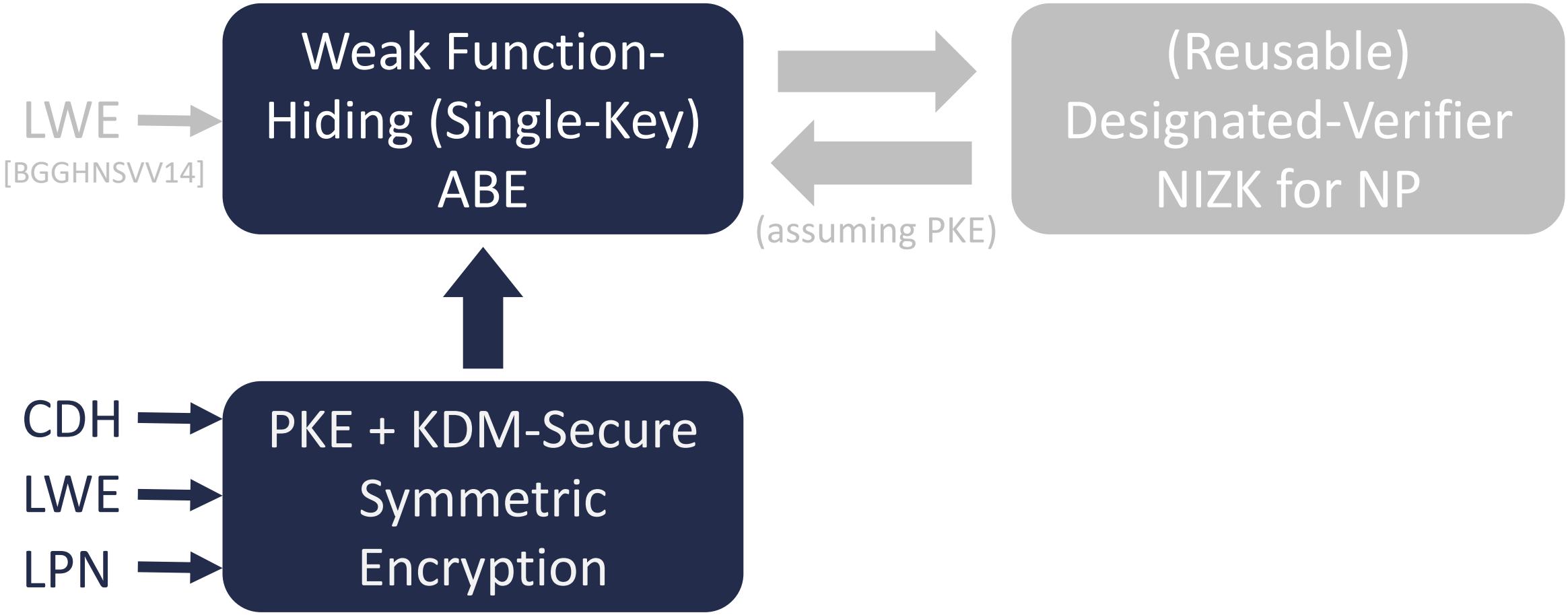
Is there a general framework for constructing DV-NIZKs?



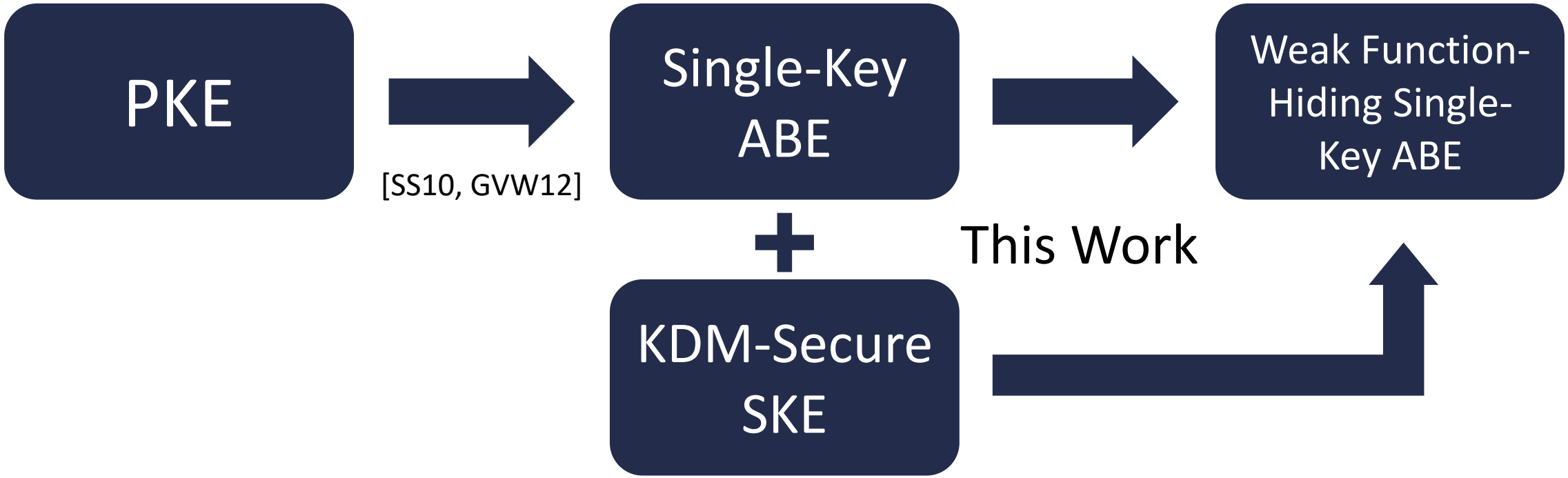
Simple variant of lattice-based ABE scheme [BGGHNSVV14] satisfies this notion

Part II: From PKE to Function-Hiding ABE

Is there a general framework for constructing DV-NIZKs?

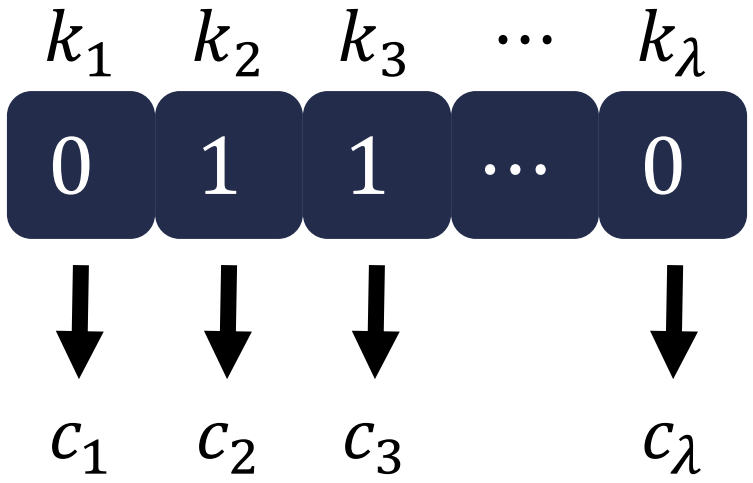


Part II: From PKE to Function-Hiding ABE



Will rely on “mirroring” technique that has been developed for constructing TDFs/CCA-security [GRM18, KW18, KMT19]

Constructing Function-Hiding ABE



Sample symmetric encryption key $k \leftarrow \{0,1\}^\lambda$

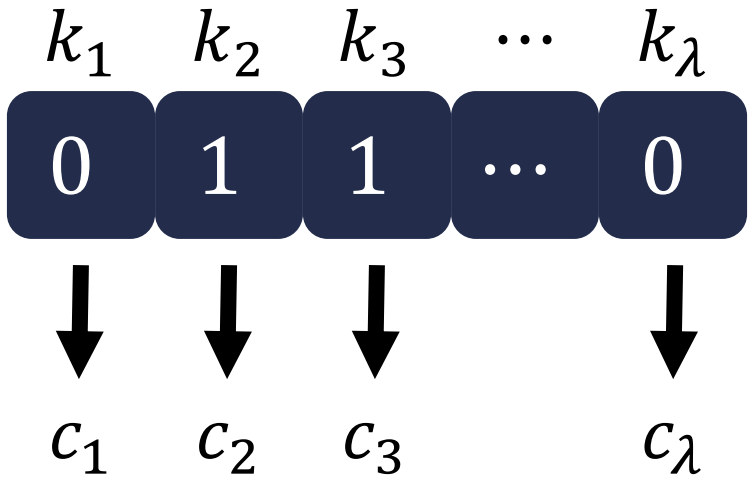
Commit to each bit of the key k

$$c_i \leftarrow \mathrm{Commit}(k_i\ ; \rho_i)$$

computational-hiding, statistically-biding, equivocable (in CRS model)

commitment randomness

Constructing Function-Hiding ABE



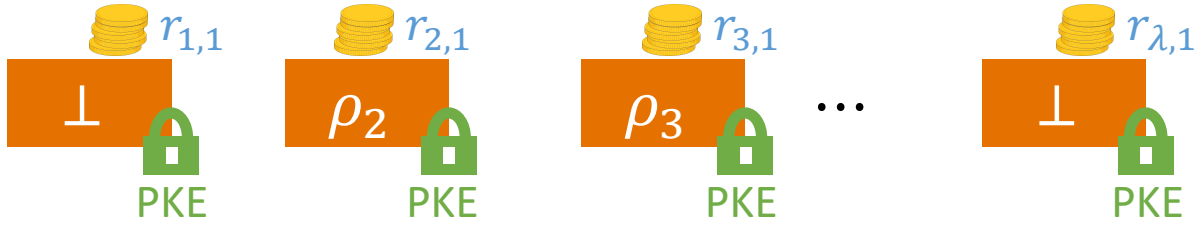
Sample symmetric encryption key $k \leftarrow \{0,1\}^\lambda$

Commit to each bit of the key k

$$c_i \leftarrow \text{Commit}(k_i\,;\rho_i)$$

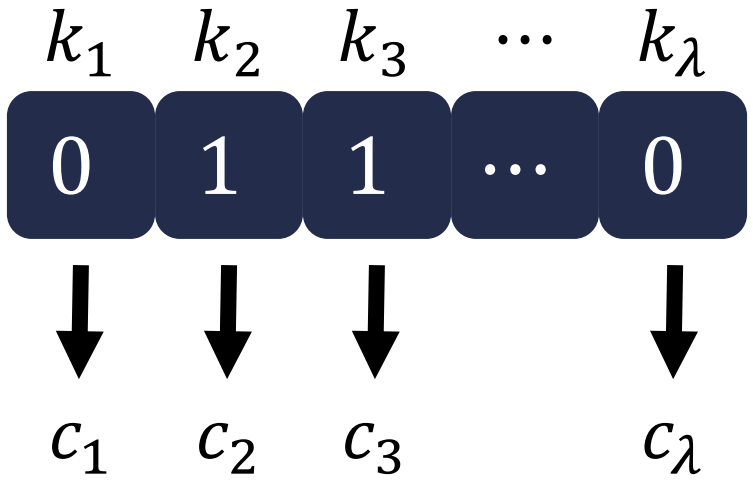
encryption randomness

if $k_i = 0$: encrypt ρ_i using ABE scheme
if $k_i = 1$: encrypt $\perp$ using ABE scheme



if $k_i = 0$: encrypt $\perp$ using PKE scheme
if $k_i = 1$: encrypt ρ_i using PKE scheme

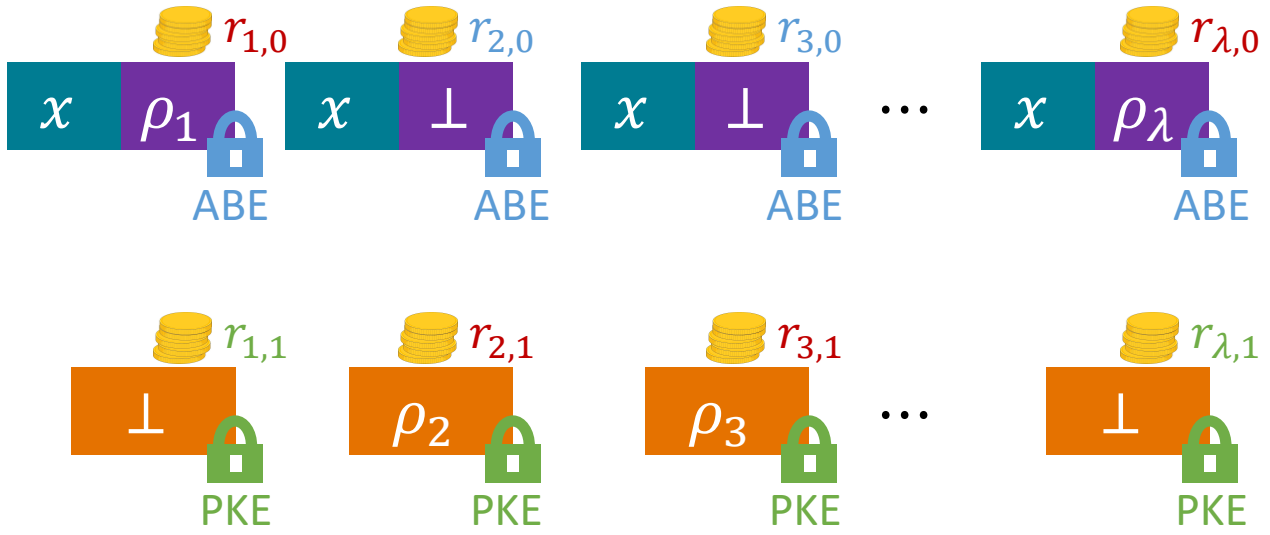
Constructing Function-Hiding ABE



Sample symmetric encryption key $k \leftarrow \{0,1\}^\lambda$

Commit to each bit of the key k

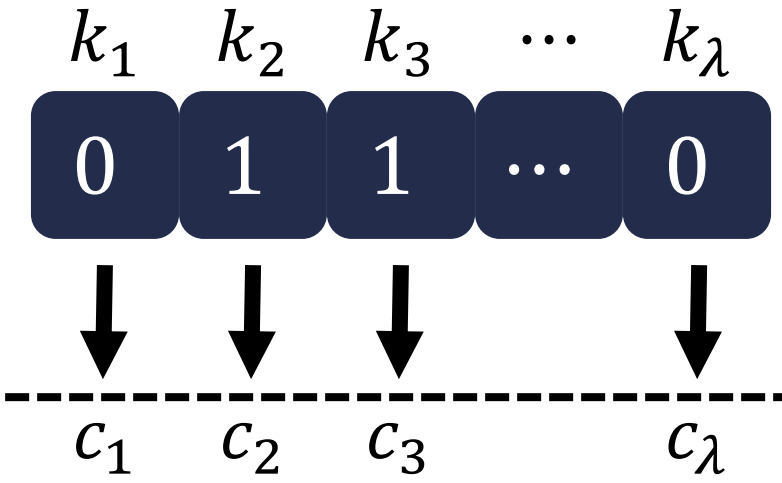
$$c_i \leftarrow \text{Commit}(k_i\,;\rho_i)$$



Encrypt message m and <u>subset</u> of encryption randomness using SKE scheme

$$m, r_{1,k_1}, \ldots, r_{\lambda,k_\lambda}$$ SKE

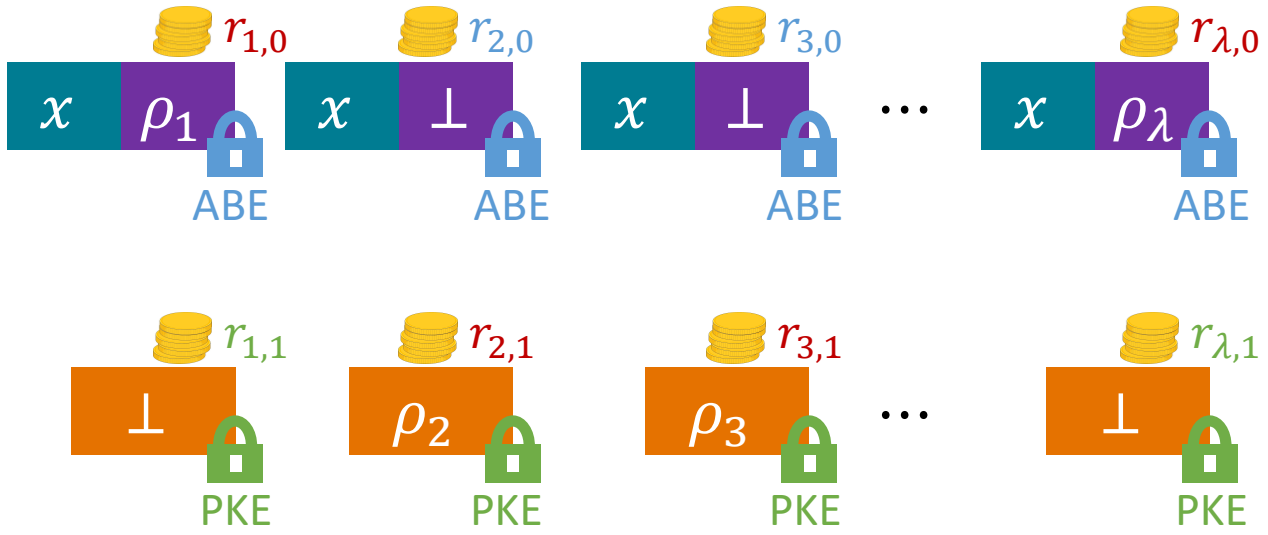
Constructing Function-Hiding ABE



Sample symmetric encryption key $k \leftarrow \{0,1\}^\lambda$

Commit to each bit of the key k

$$c_i \leftarrow \text{Commit}(k_i\,;\rho_i)$$



Encrypt message m and <u>subset</u> of encryption randomness using SKE scheme

$$m, r_{1,k_1}, \ldots, r_{\lambda,k_\lambda}$$

SKE

Ciphertext

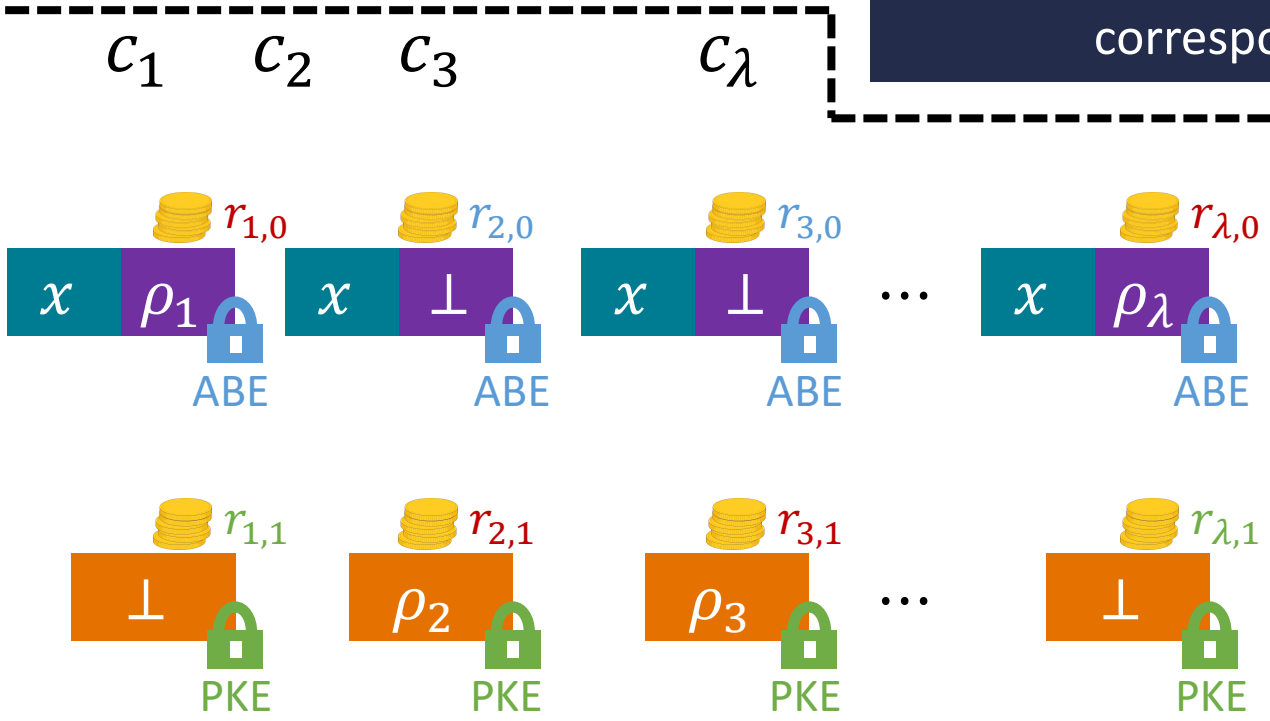
Constructing Function-Hiding ABE

Decryption key: ABE decryption key for f

Decryption algorithm:

1. Decrypt ABE ciphertexts using secret key to obtain messages z_i
2. If $c_i = \mathrm{Commit}(0\,; z_i)$, set $k_i = 0$, else $k_i = 1$
3. Decrypt symmetric ciphertext with recovered key k
4. **Validity check:** check that ABE/PKE ciphertexts corresponding to bits of k_i are well formed

Note: Validity check requires that we can recover message given the encryption randomness (follows without loss of generality)



$m, r_{1,k_1}, \ldots, r_{\lambda,k_\lambda}$ SKE

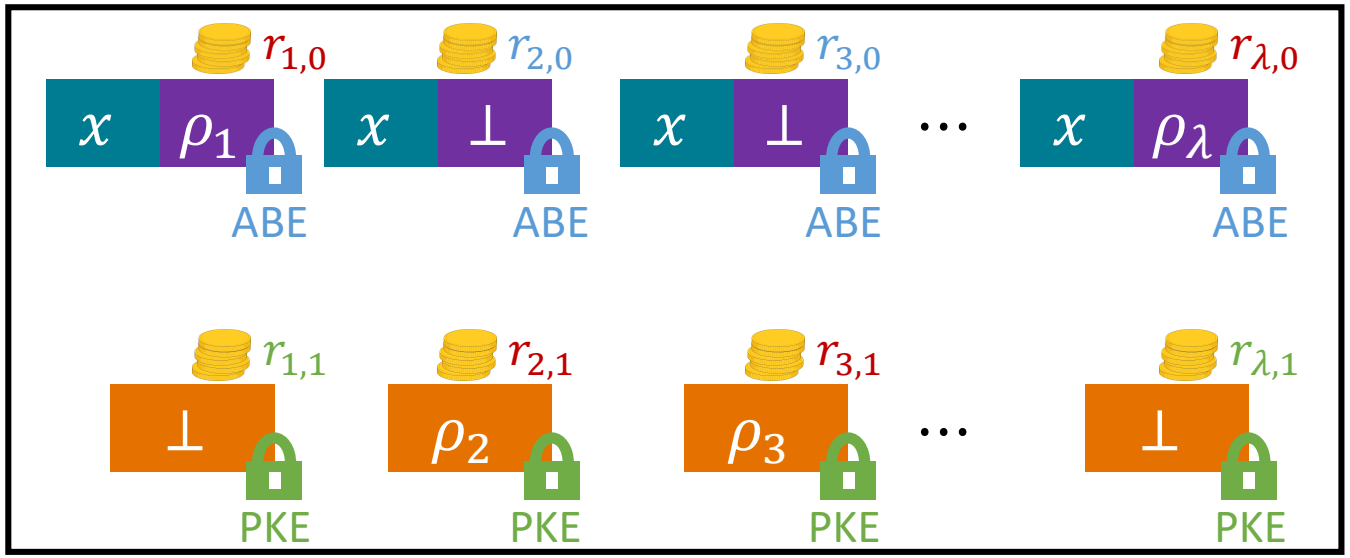
Ciphertext

Constructing Function-Hiding ABE

Function-Hiding: Decryption oracle (with sk_f) hides f (up to what is revealed by $f(x)$)

Commitments are *statistically-binding* so adversary cannot signal both 0 and 1 for a bit

Any collection of ciphertexts can only bind to a single possible k (which is checked by validity test)

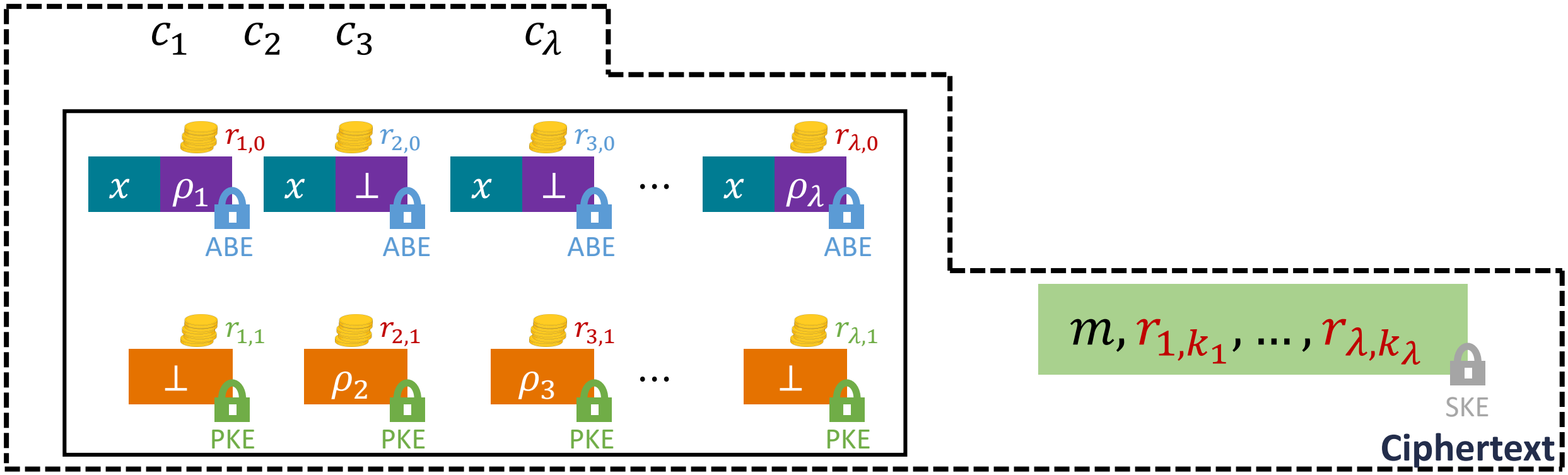


Instead of decrypting using the ABE secret key, we can decrypt using the secret key for the PKE scheme (consistency check ensures that behavior is identical)

"Ideal" decryption function is independent of f

Constructing Function-Hiding ABE

Semantic Security: If $f(x) = 0$, then message is hidden

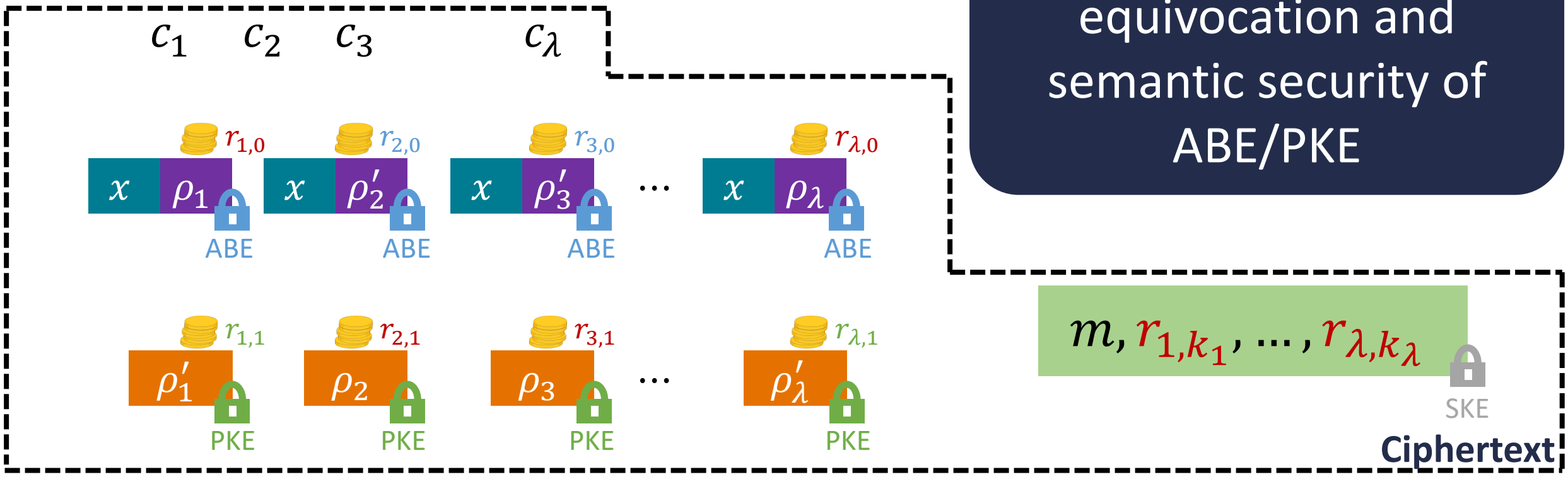


Constructing Function-Hiding ABE

Semantic Security: If $f(x) = 0$, then message is hidden

Switch commitments to equivocable mode

$$c_i = \text{Commit}(k_i\,;\rho_i) = \text{Commit}(1 - k_i\,;\rho_i')$$



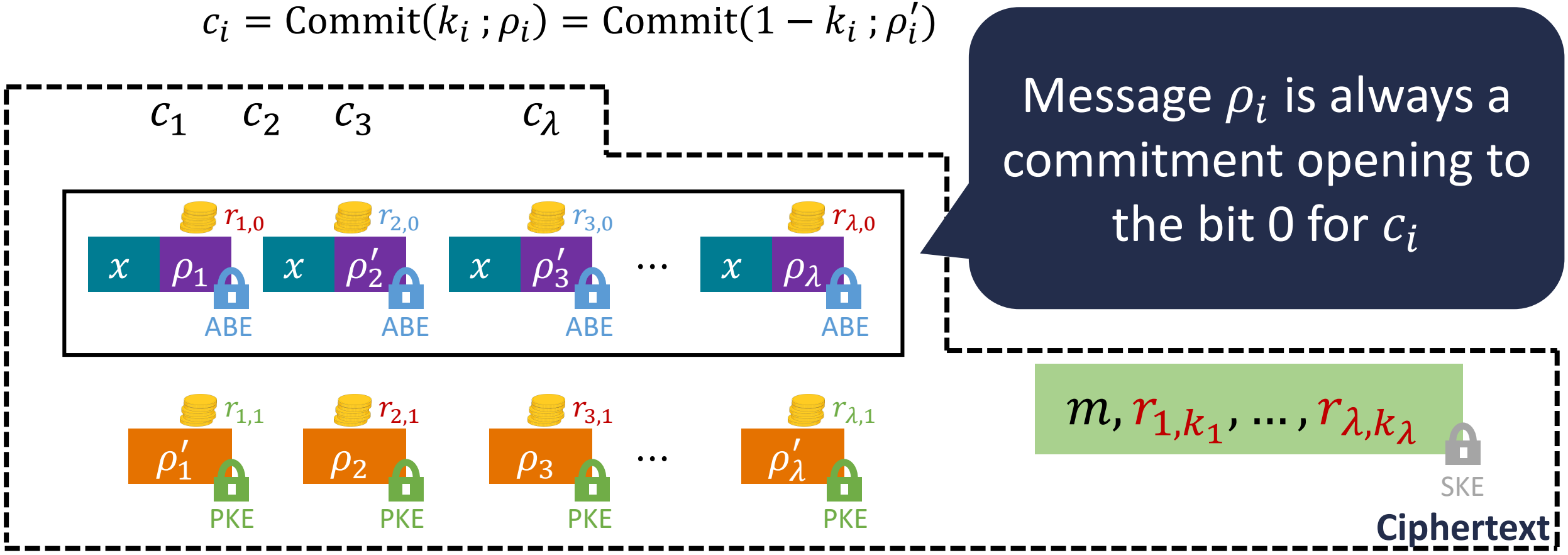
Indistinguishable by equivocation and semantic security of ABE/PKE

Constructing Function-Hiding ABE

Semantic Security: If $f(x) = 0$, then message is hidden

Switch commitments to equivocable mode

$$c_i = \text{Commit}(k_i\,; \rho_i) = \text{Commit}(1 - k_i\,; \rho_i')$$

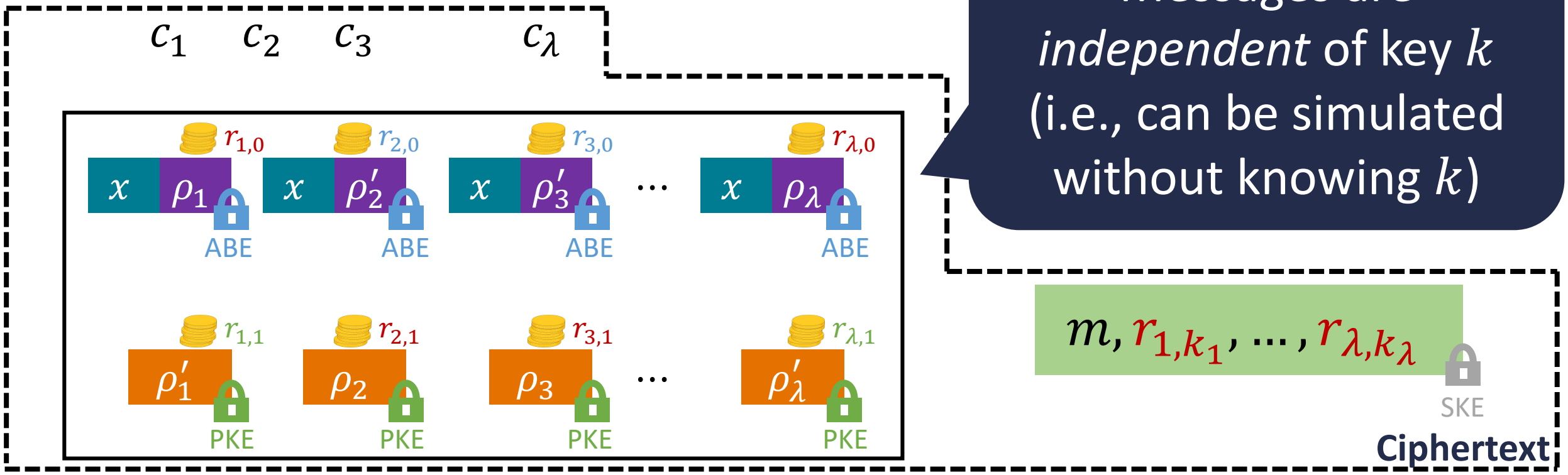


Constructing Function-Hiding ABE

Semantic Security: If $f(x) = 0$, then message is hidden

Switch commitments to equivocable mode

$$c_i = \text{Commit}(k_i\,;\rho_i) = \text{Commit}(1 - k_i\,;\rho_i')$$

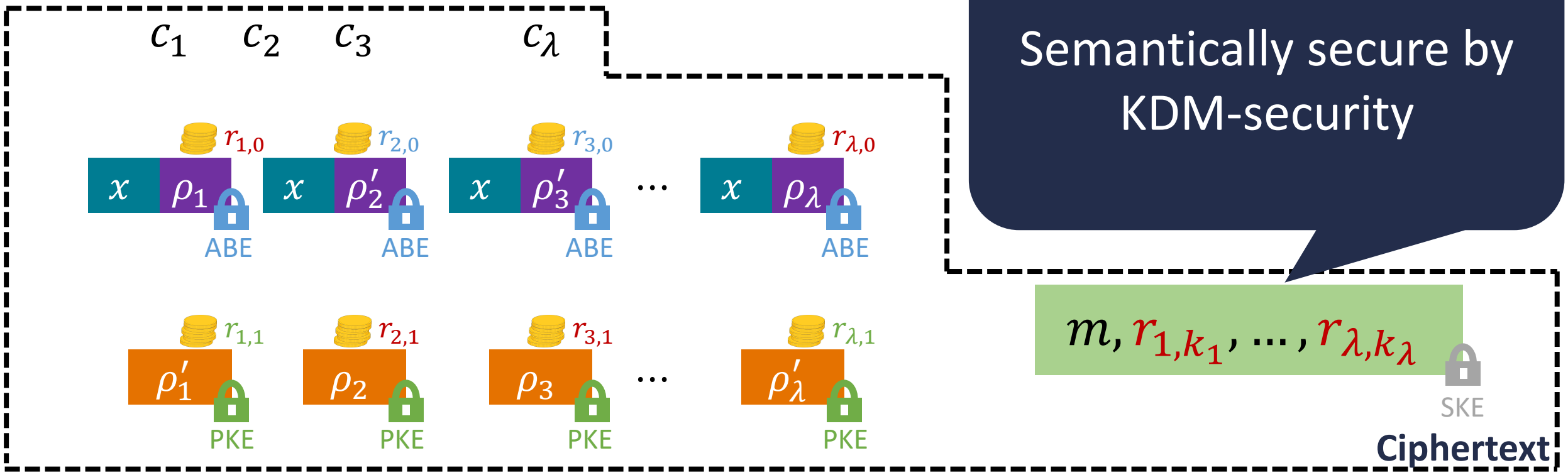


Constructing Function-Hiding ABE

Semantic Security: If $f(x) = 0$, then message is hidden

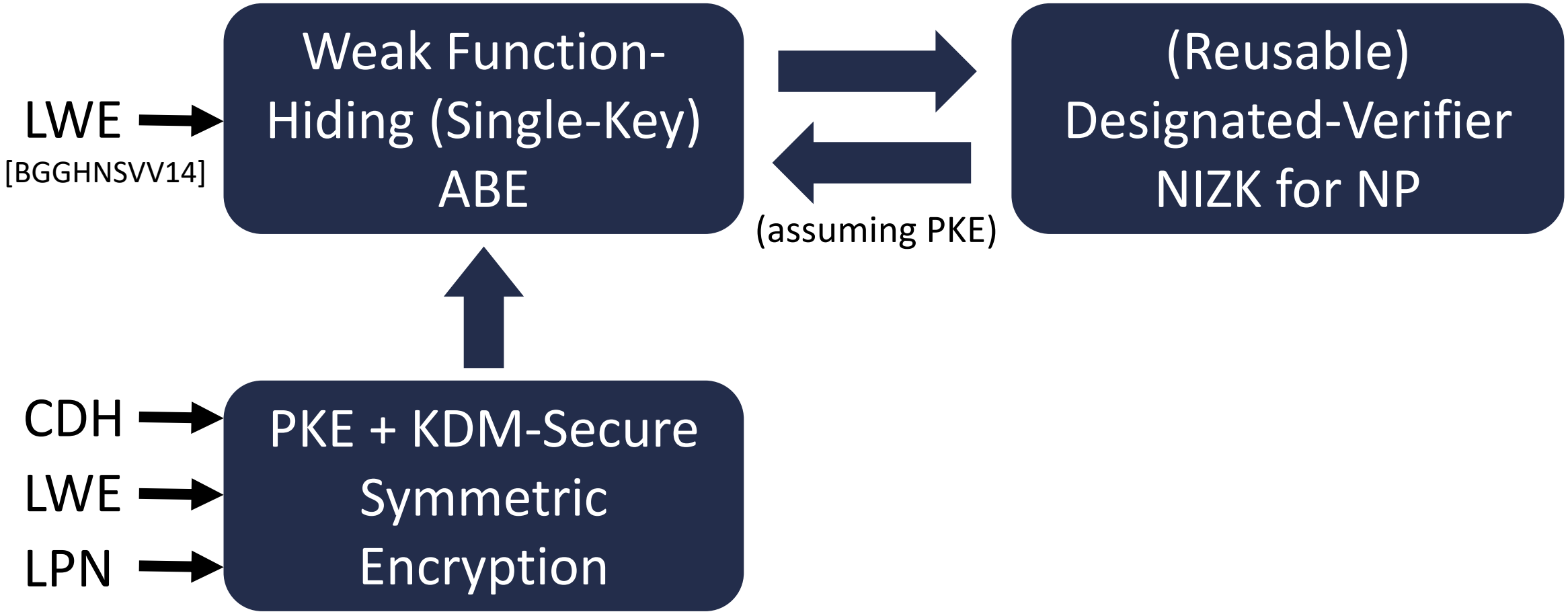
Switch commitments to equivocable mode

$$c_i = \text{Commit}(k_i\,; \rho_i) = \text{Commit}(1 - k_i\,; \rho_i')$$



Our Results

Is there a general framework for constructing DV-NIZKs?



Malicious DV-NIZKs

σ

k_V

prover

$\pi = \text{Prove}(\sigma, x, w)$

verifier

Standard DV-NIZK: CRS and verification state needs to be generated by a <u>trusted</u> party

Malicious DV-NIZKs

common <u>random</u> string

`11101001101111100110110000001`

prover

$\pi = \text{Prove}(\sigma, \text{pk}, x, w)$

verifier

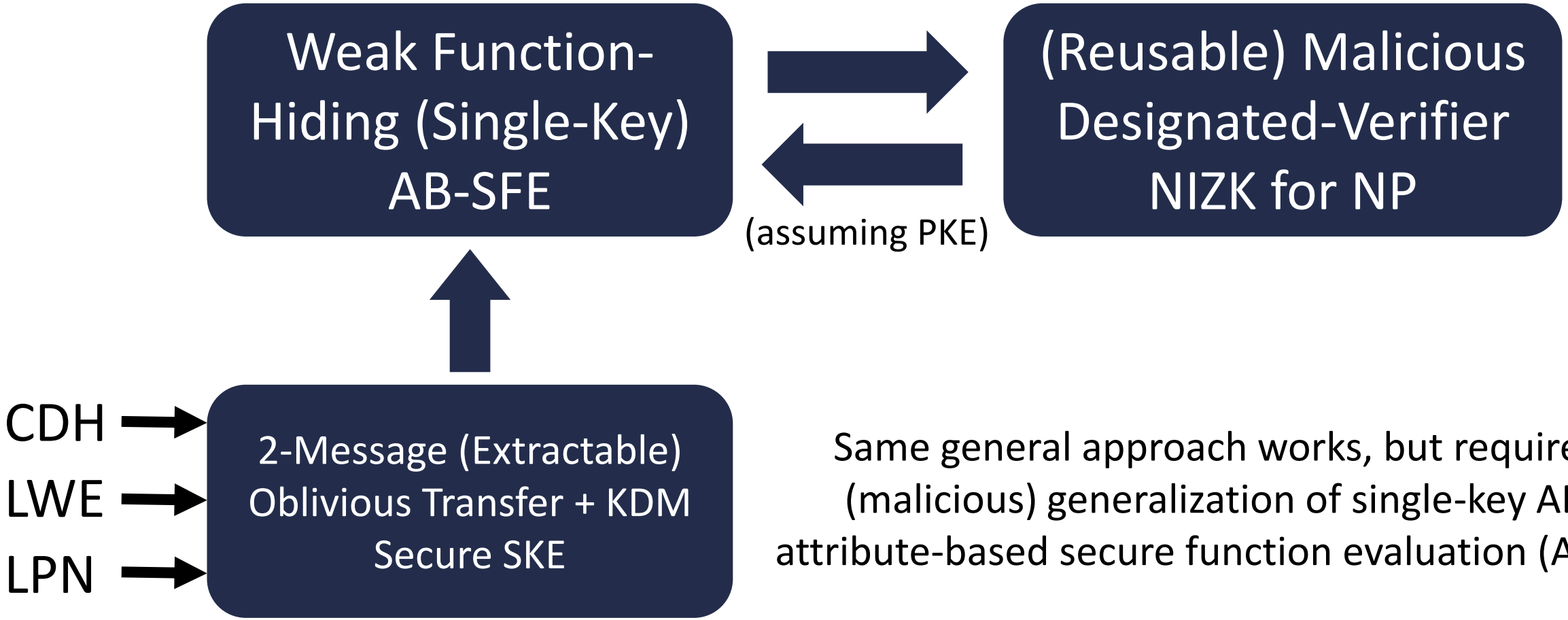
pk

sk

Malicious DV-NIZK [QRW19]: only trusted setup needed is common random string, verifier publishes its own public/secret key-pair

Our Results: Malicious DV-NIZKs

Is there a general framework for constructing DV-NIZKs?



Same general approach works, but requires a (malicious) generalization of single-key ABE: attribute-based secure function evaluation (AB-SFE)

Open Questions

Is there a general framework for constructing DV-NIZKs?

Can we construct weak function-hiding single-key ABE from PKE?
[Would mean that CPA-security generically implies CCA-security!]

Open Questions

Is there a general framework for constructing DV-NIZKs?

Can we construct weak function-hiding single-key ABE from PKE?

Can we construct weak function-hiding single-key ABE from CCA-secure PKE?
[Converse of Naor-Yung]

Open Questions

Is there a general framework for constructing DV-NIZKs?

Can we construct weak function-hiding single-key ABE from PKE?

Can we construct weak function-hiding single-key ABE from CCA-secure PKE?

Can we get reusable preprocessing NIZKs from OWFs?

Open Questions

Is there a general framework for constructing DV-NIZKs?

Thank you!

https://eprint.iacr.org/2019/242.pdf