Designated-Verifier Pseudorandom Generators, and their Applications

Geoffroy Couteau, Dennis Hofheinz

Karlsruher Institut für Technologie

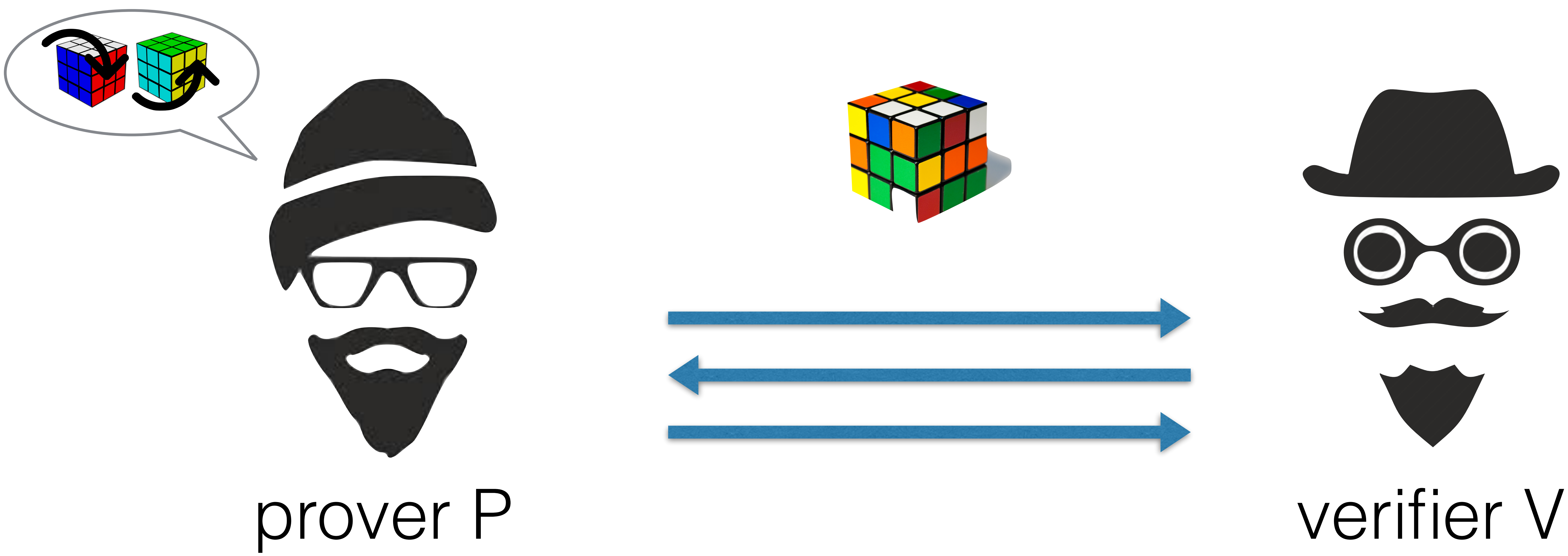
Reusable Designated-Verifier NIZKs for all NP from CDH

Willy Quach, Ron D. Rothblum, and Daniel Wichs

Designated Verifier/Prover and Preprocessing NIZKs from Diffie-Hellman Assumptions

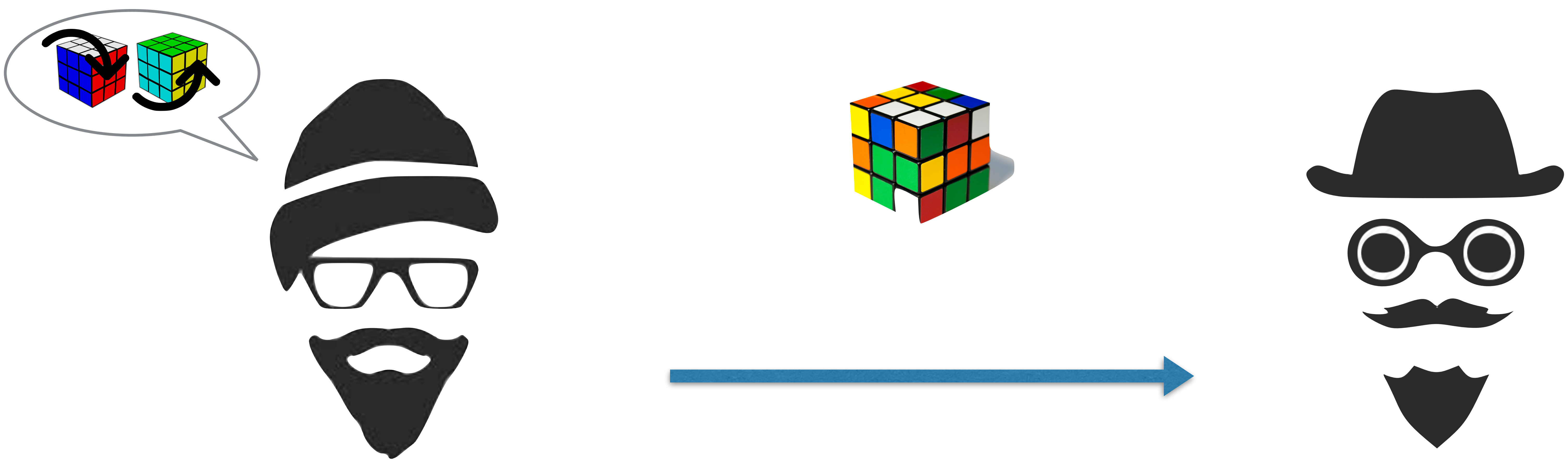
Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa

Zero-Knowledge Proof



- Complete: if P knows a solution, V accepts
- Sound: if there is no solution, P cannot convince V
- Zero-Knowledge: V does not learn the solution

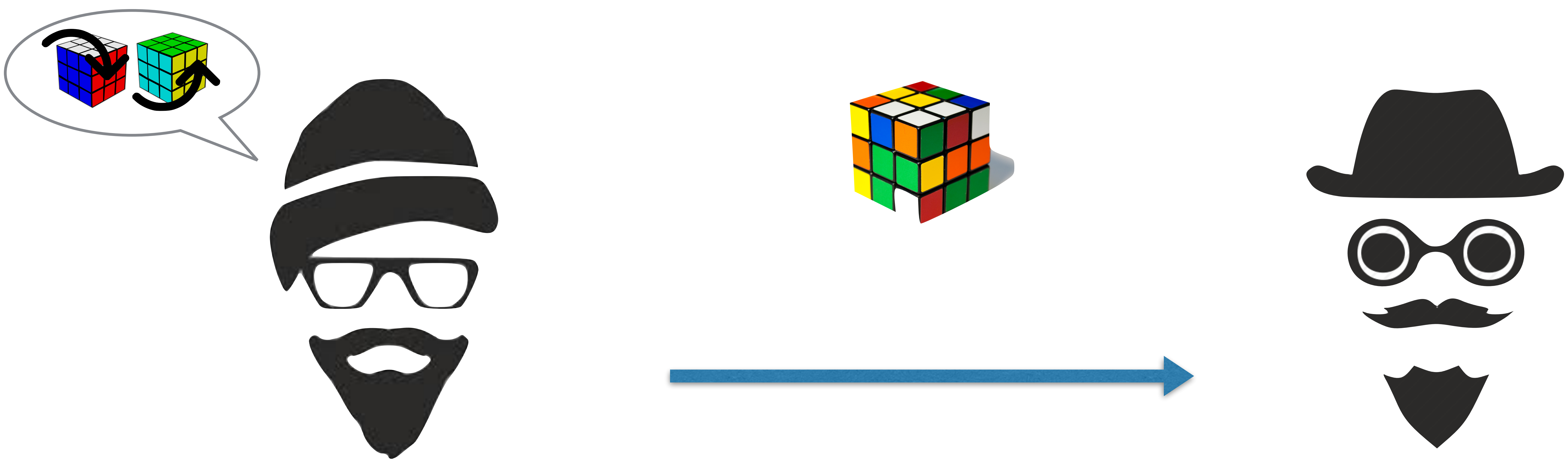
Non-Interactive Zero-Knowledge Proof



prover P verifier V

- Complete: if P knows a solution, V accepts
- Sound: if there is no solution, P cannot convince V
- Zero-Knowledge: V does not learn the solution

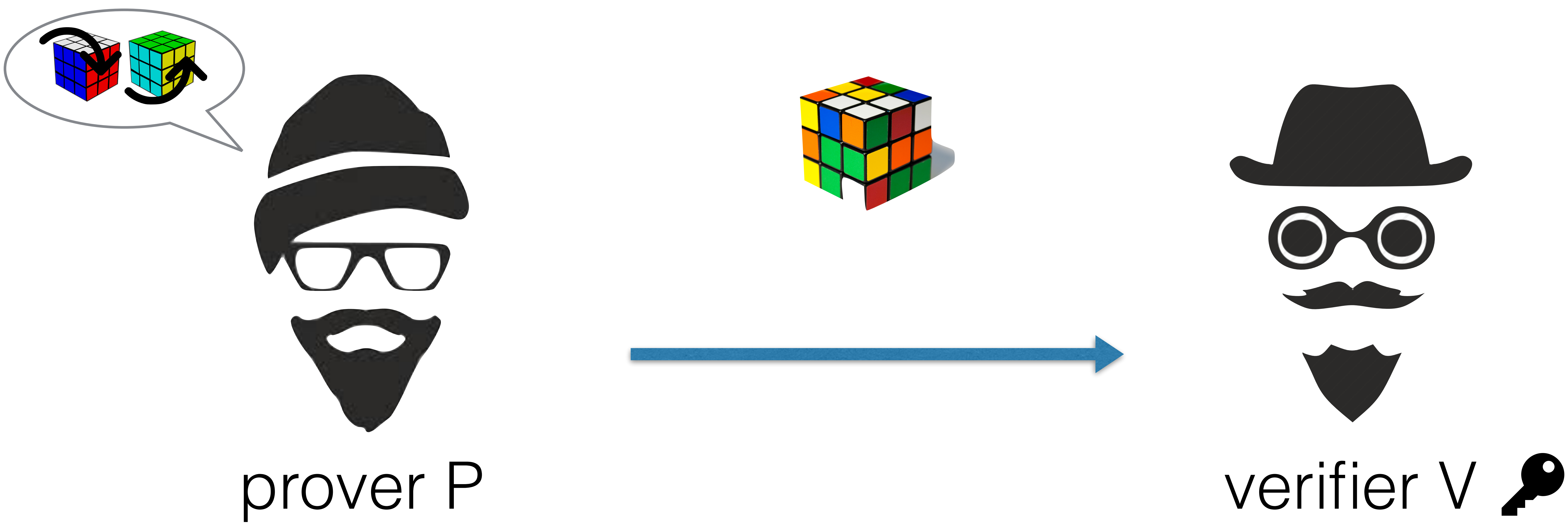
Designated-Verifier NIZK



prover P verifier V

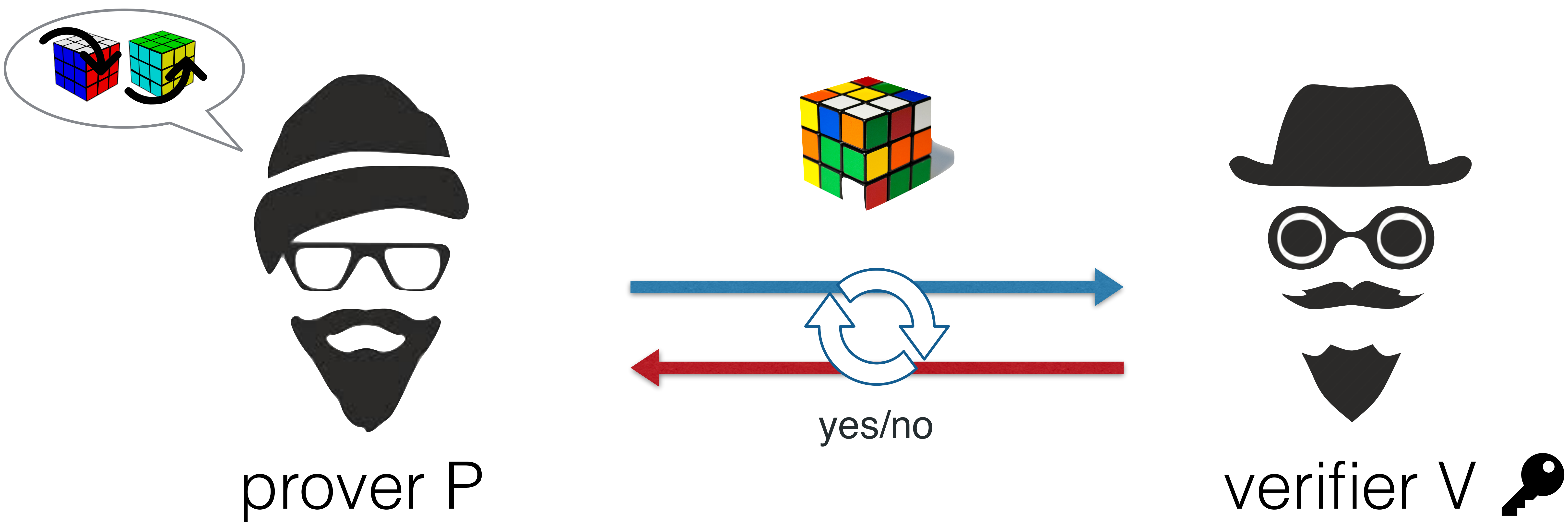
- Complete: if P knows a solution, V accepts
- Sound: if there is no solution, P cannot convince V
- Zero-Knowledge: V does not learn the solution

Designated-Verifier NIZK



- Complete: if P knows a solution, V accepts
- Sound: if there is no solution, P cannot convince V
- Zero-Knowledge: V does not learn the solution

Designated-Verifier NIZK



- Complete: if P knows a solution, V accepts
- Unbounded Soundness: if there is no solution, P cannot convince V
- Zero-Knowledge: V does not learn the solution

Brief History of (DV)NIZKs

ZKP

[GMR85], seminal paper on zero-knowledge proofs

NIZK

[Ore87], [GMR88], impossibility of NIZK in the plain model + seminal paper on NIZKs with CRS, NIZK from quadratic residuosity

NIZK from TDP

[FLS90], first NIZK from a generic assumption, introduces the hidden-bit model

NIZK from VPRG

[DN00], first NIZK from a necessary and sufficient assumption

NIZK from pairings

[CHK03], inefficient construction [GOS06b,GOS06a,GS08] & follow-ups, efficient pairing-based NIZKs.

NIZK from new assumptions

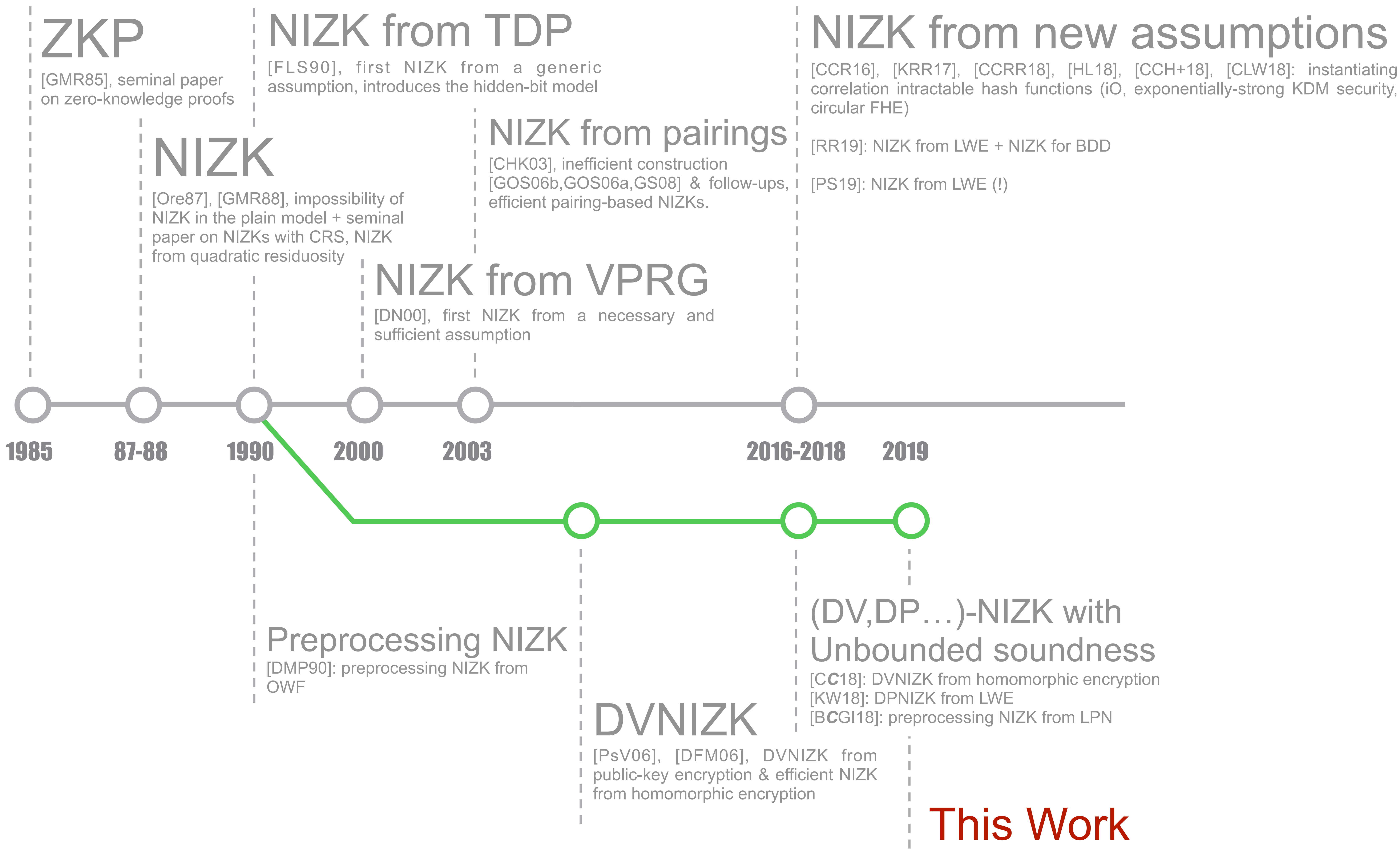
[CCR16], [KRR17], [CCRR18], [HL18], [CCH+18], [CLW18]: instantiating correlation intractable hash functions (iO, exponentially-strong KDM security, circular FHE)

[RR19]: NIZK from LWE + NIZK for BDD

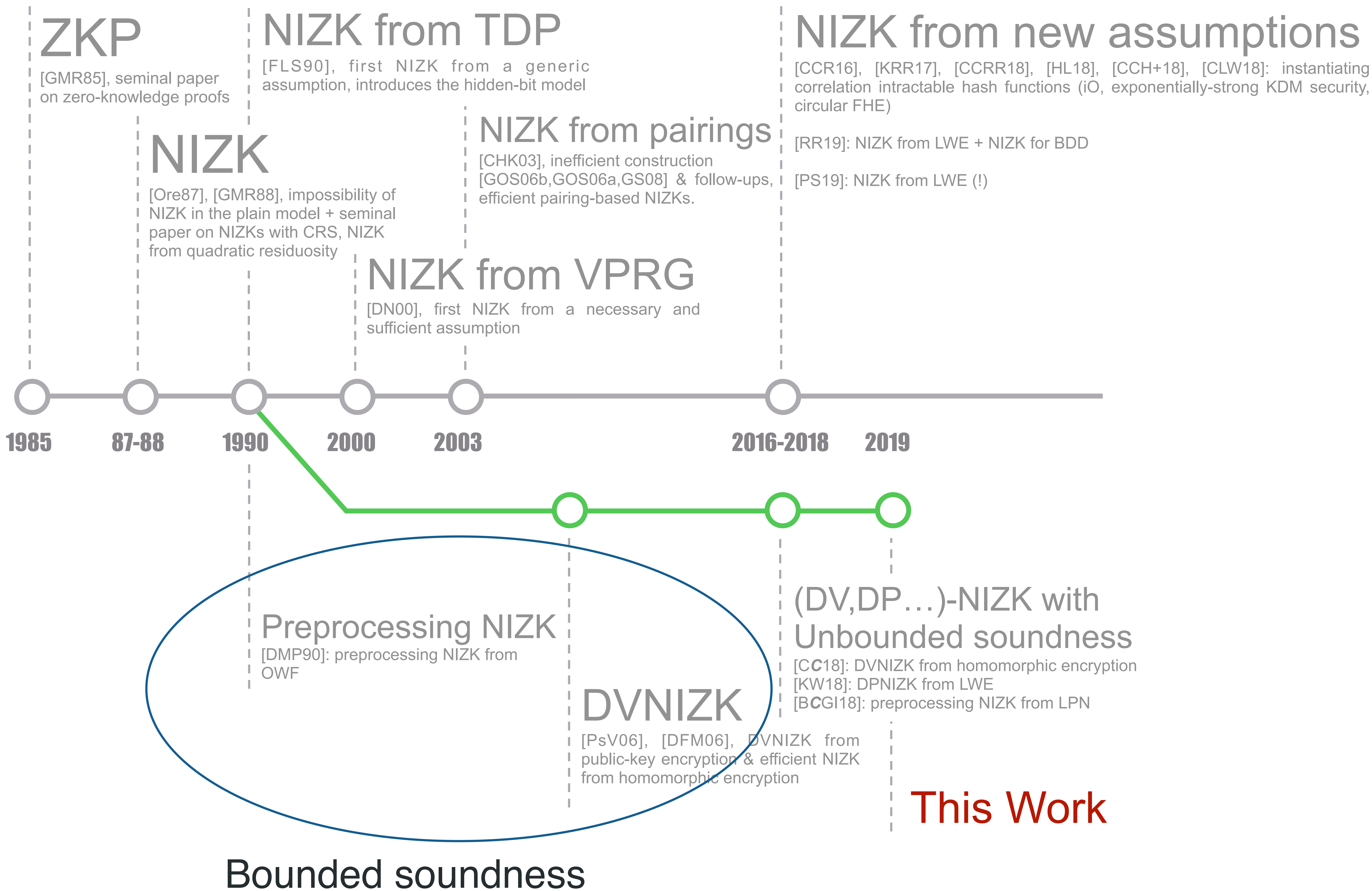
[PS19]: NIZK from LWE (!)

1985 — 87-88 — 1990 — 2000 — 2003 — 2016-2018 — 2019

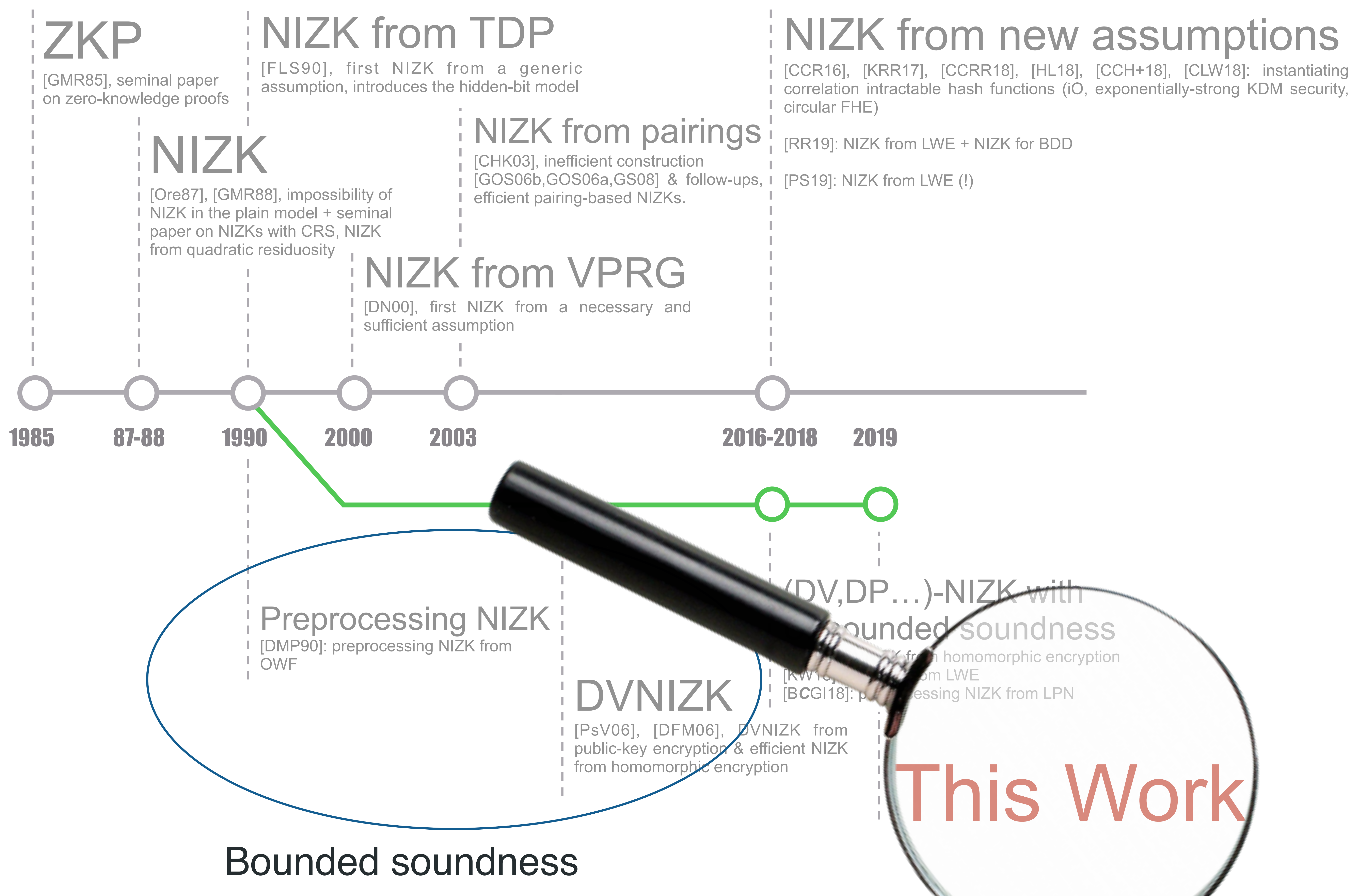
Brief History of (DV)NIZKs



Brief History of (DV)NIZKs



Brief History of (DV)NIZKs



Our Contribution

We obtain two new constructions:

1) A DVNIZK for NP under the CDH assumption

First direct indication that DVNIZK with unbounded soundness are actually easier to build than standard NIZK

2) A (DV)NIZK for NP assuming LWE and the existence of a (DV)NIWI for BDD

Improving over, and considerably simplifying, the recent result of [RSS19] which required a NIZK for BDD.

Our Contribution

We obtain two new constructions:

1) A DVNIZK for NP under the CDH assumption

First direct indication that DVNIZK with unbounded soundness are actually easier to build than standard NIZK

2) A (DV)NIZK for NP assuming LWE and the existence of a (DV)NIWI for BDD

Improving over, and considerably simplifying, the recent result of [RSS19] which required a NIZK for BDD.

But subsumed by [PS19] :)

Roadmap

[DN00]: Verifiable Pseudorandom Generator + NIZK in the hidden-bit model $\Rightarrow$ NIZK

Roadmap

[DN00]: Verifiable Pseudorandom Generator + NIZK in the hidden-bit model ⇨ NIZK

⇩

Verifiable Pseudorandom Generator:
- Relaxed soundness
- Generalization to the DV setting

Roadmap

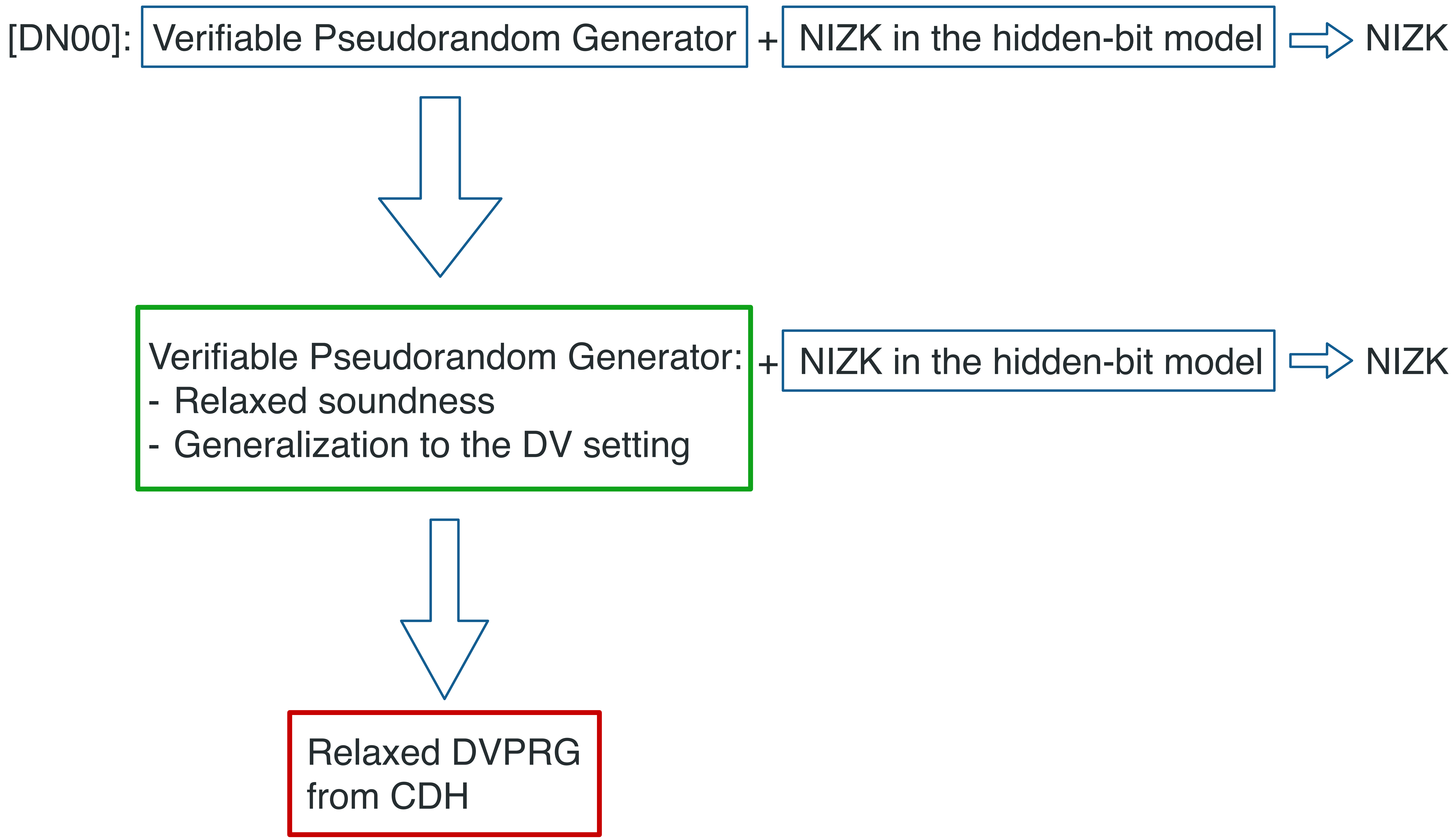
[DN00]: Verifiable Pseudorandom Generator + NIZK in the hidden-bit model ⇨ NIZK

⇩

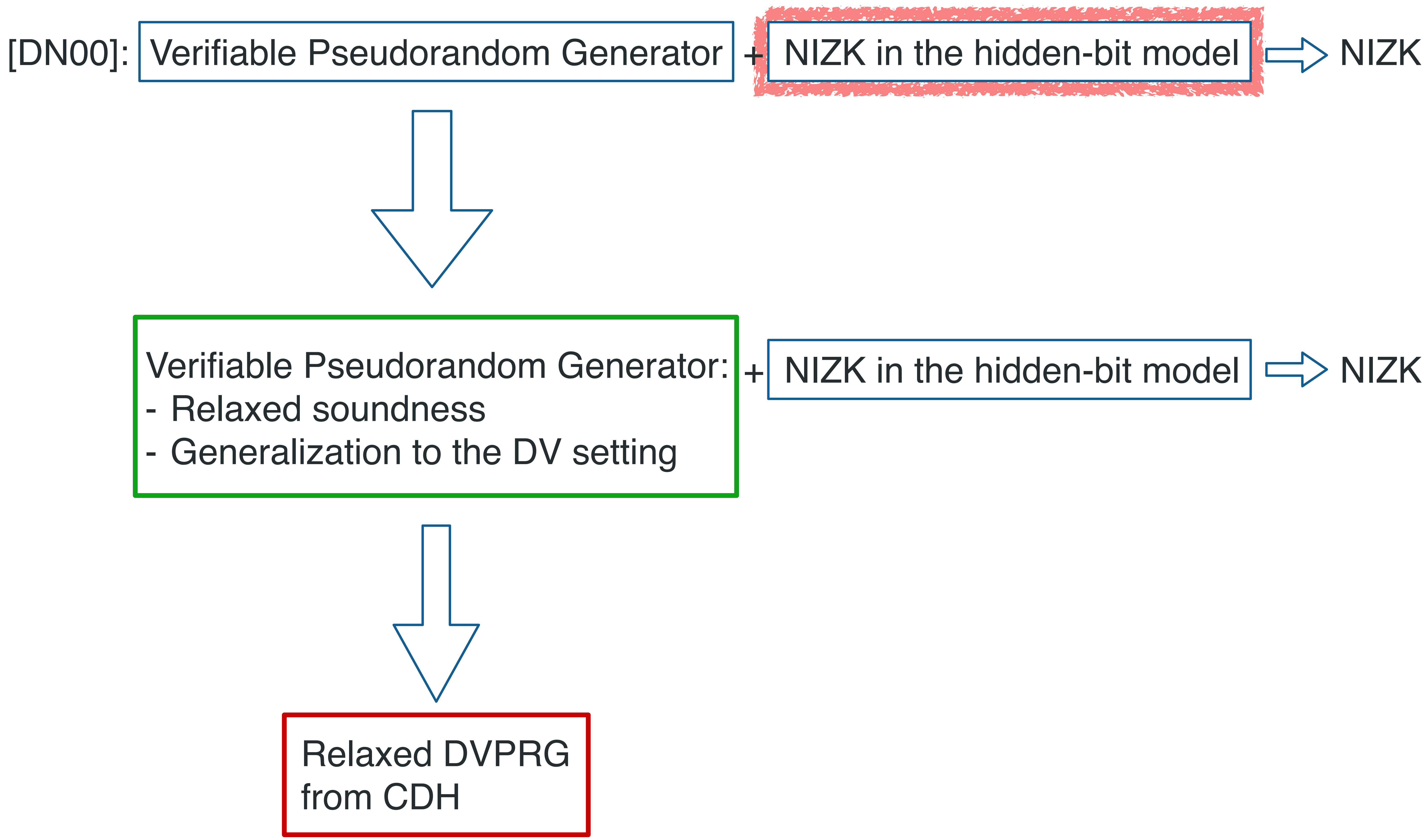
Verifiable Pseudorandom Generator:
- Relaxed soundness
- Generalization to the DV setting

\+ NIZK in the hidden-bit model ⇨ NIZK

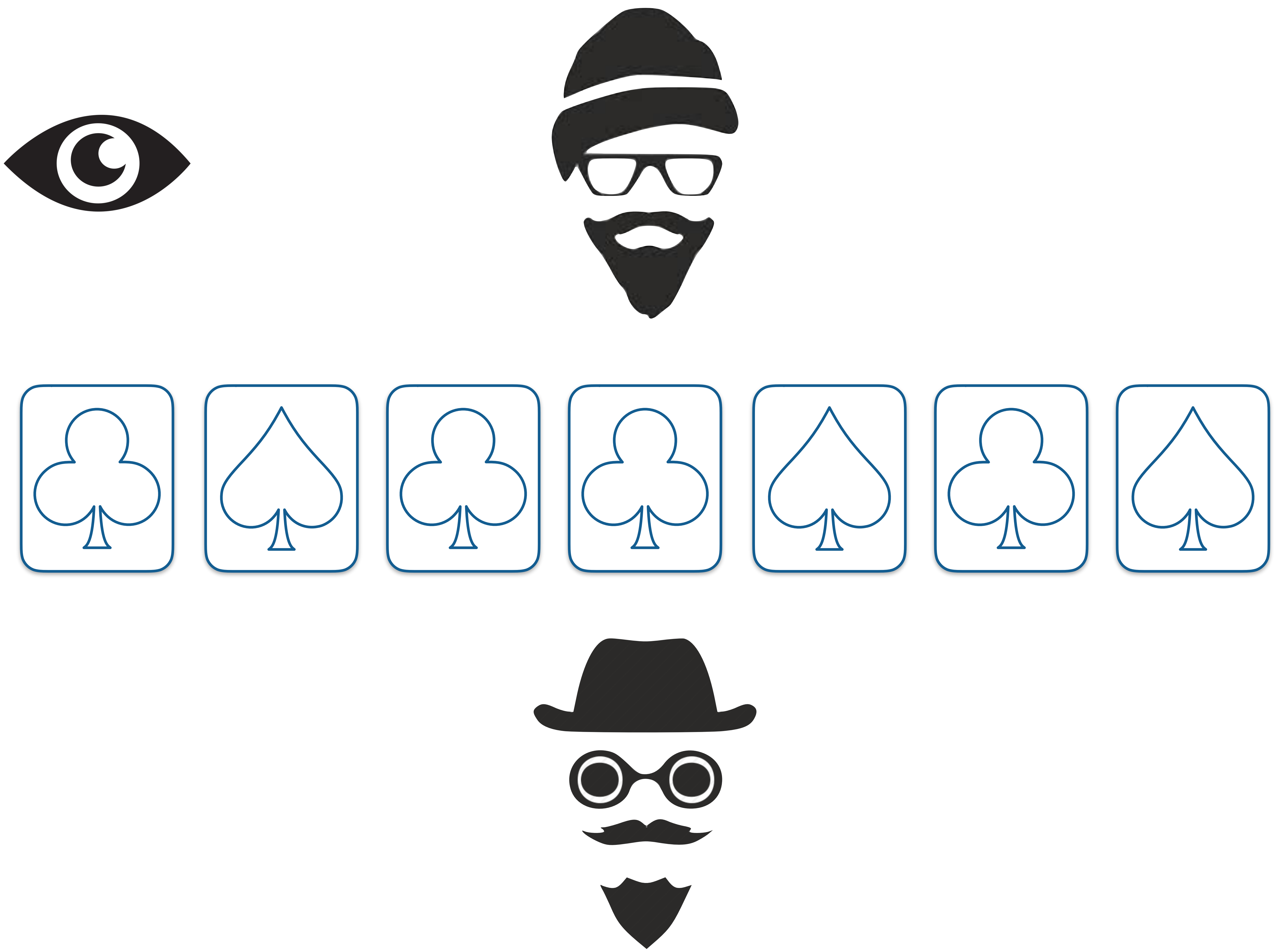
Roadmap


[DN00]:
Verifiable Pseudorandom Generator
+
NIZK in the hidden-bit model
NIZK
Verifiable Pseudorandom Generator:
- Relaxed soundness
- Generalization to the DV setting
+
NIZK in the hidden-bit model
NIZK
Relaxed DVPRG
from CDH

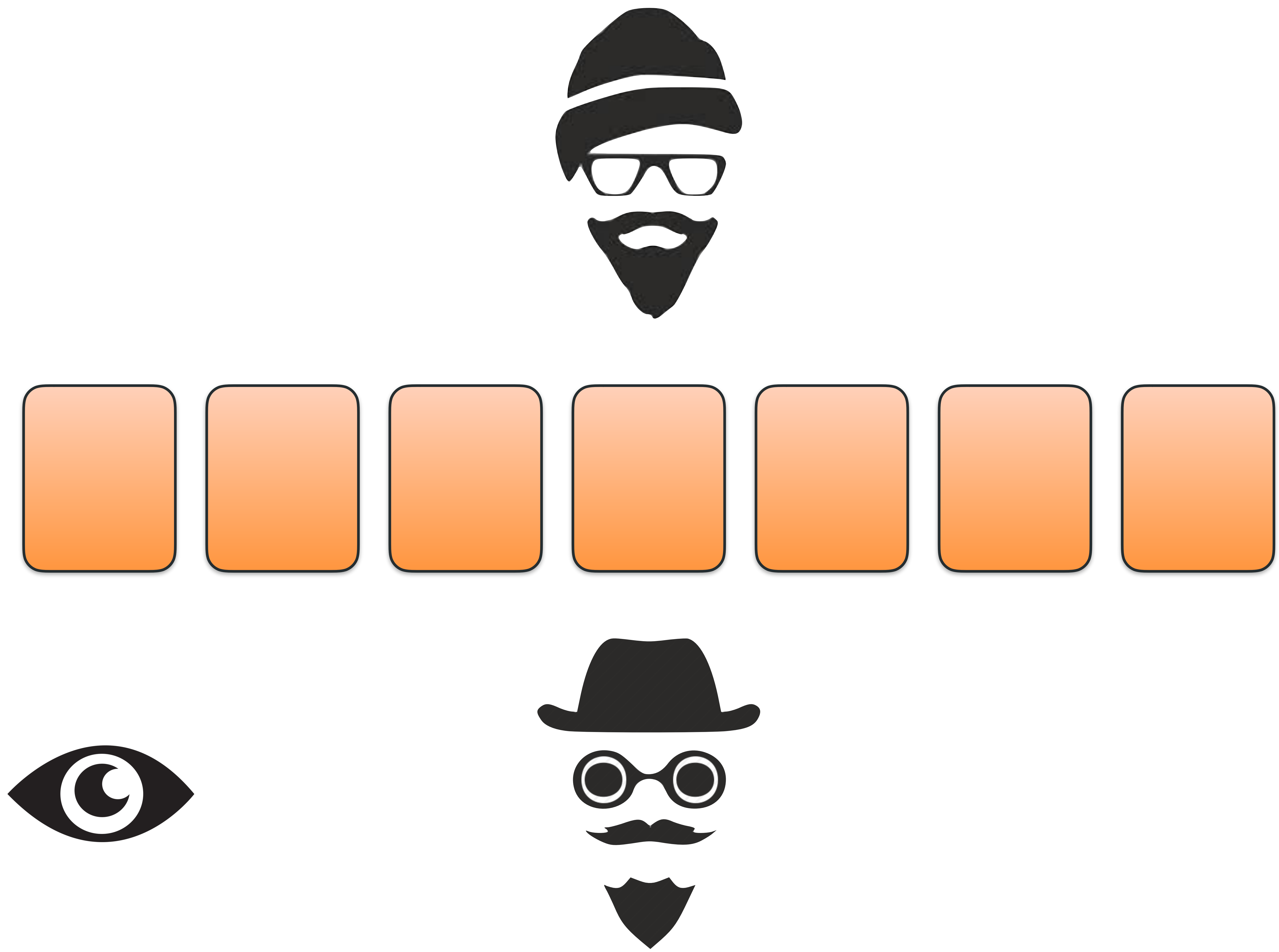
Roadmap


[DN00]:
Verifiable Pseudorandom Generator
+
NIZK in the hidden-bit model
NIZK
Verifiable Pseudorandom Generator:
- Relaxed soundness
- Generalization to the DV setting
+
NIZK in the hidden-bit model
NIZK
Relaxed DVPRG
from CDH

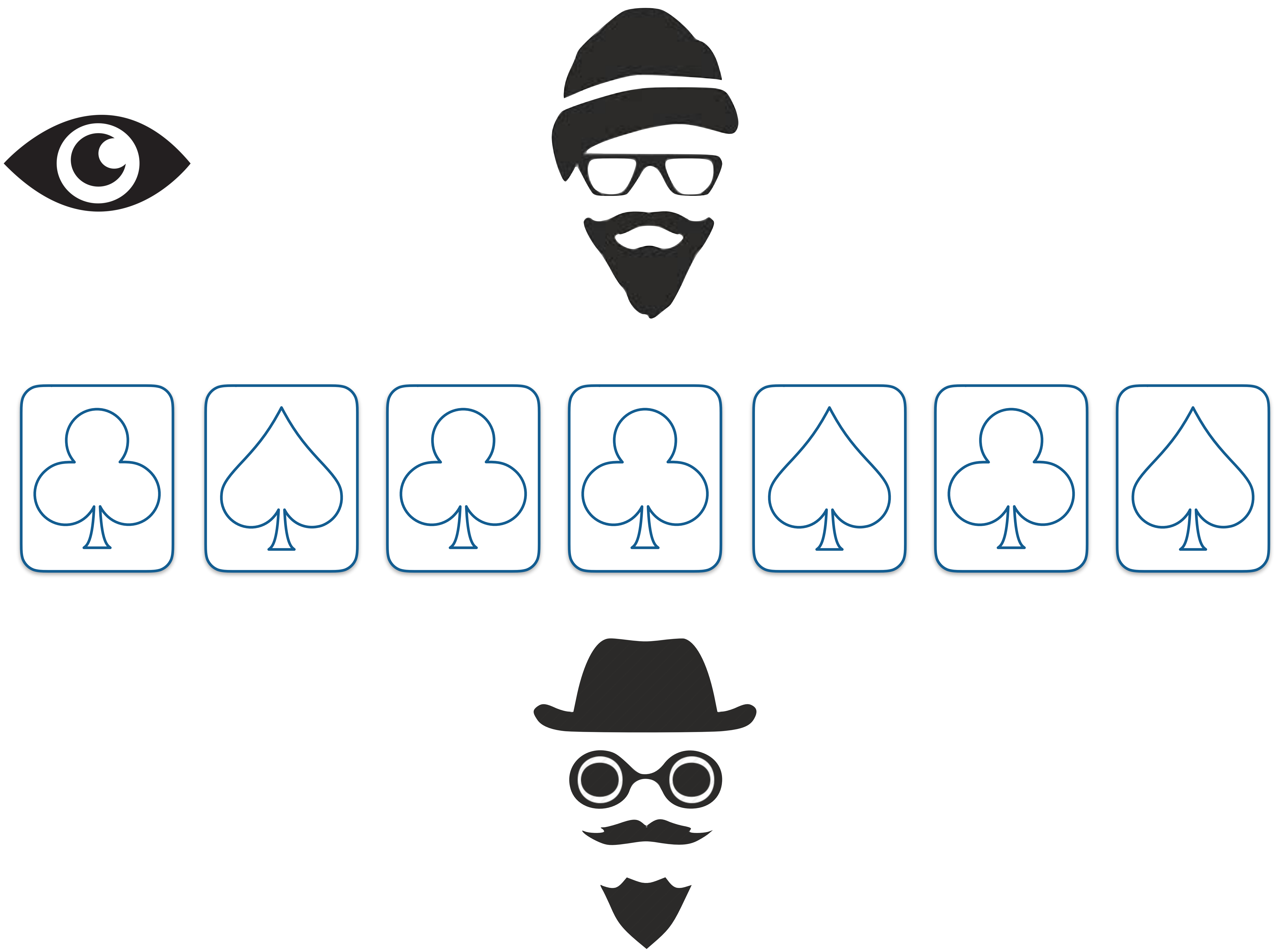
The Hidden-Bit Model



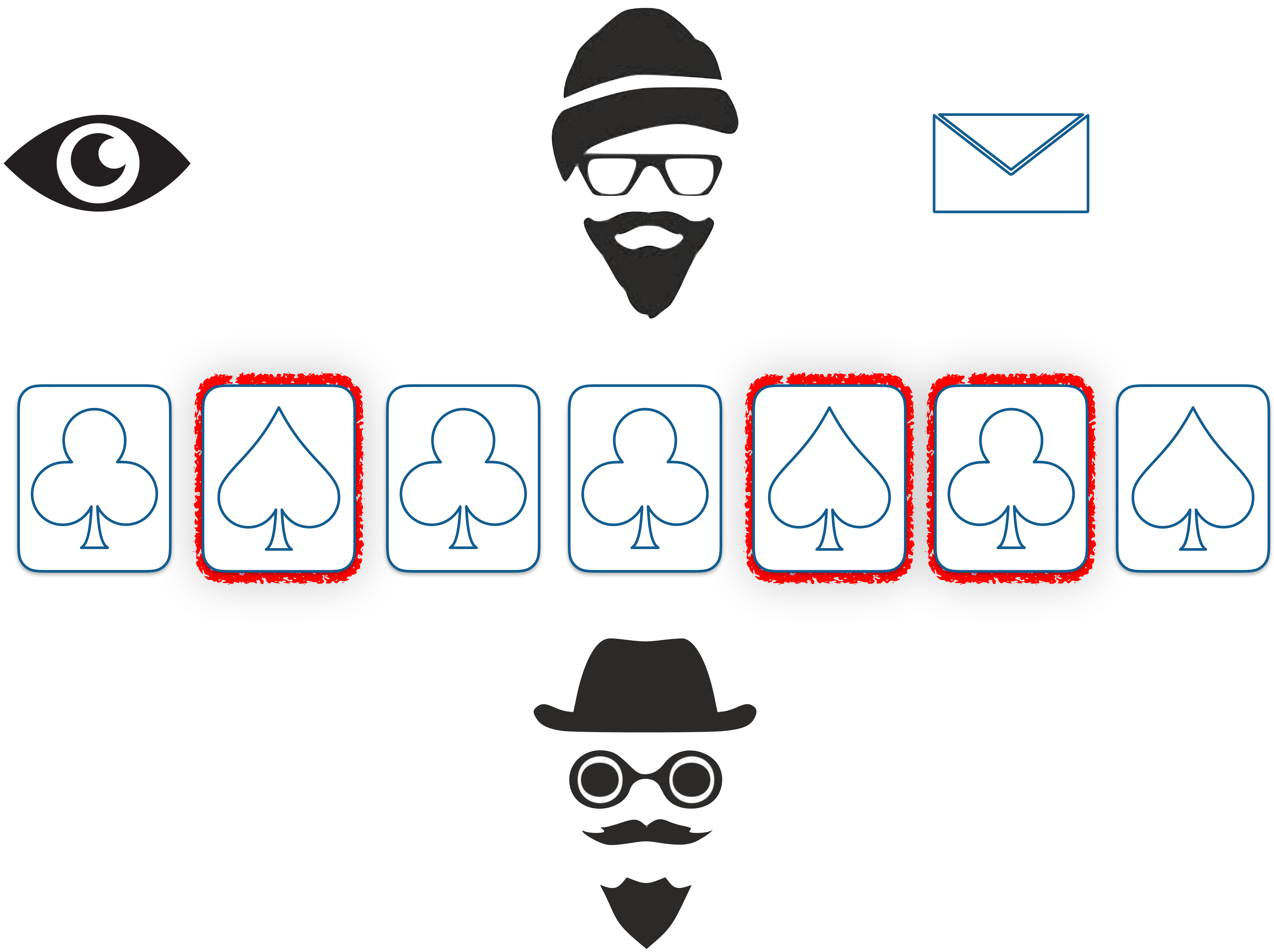
The Hidden-Bit Model



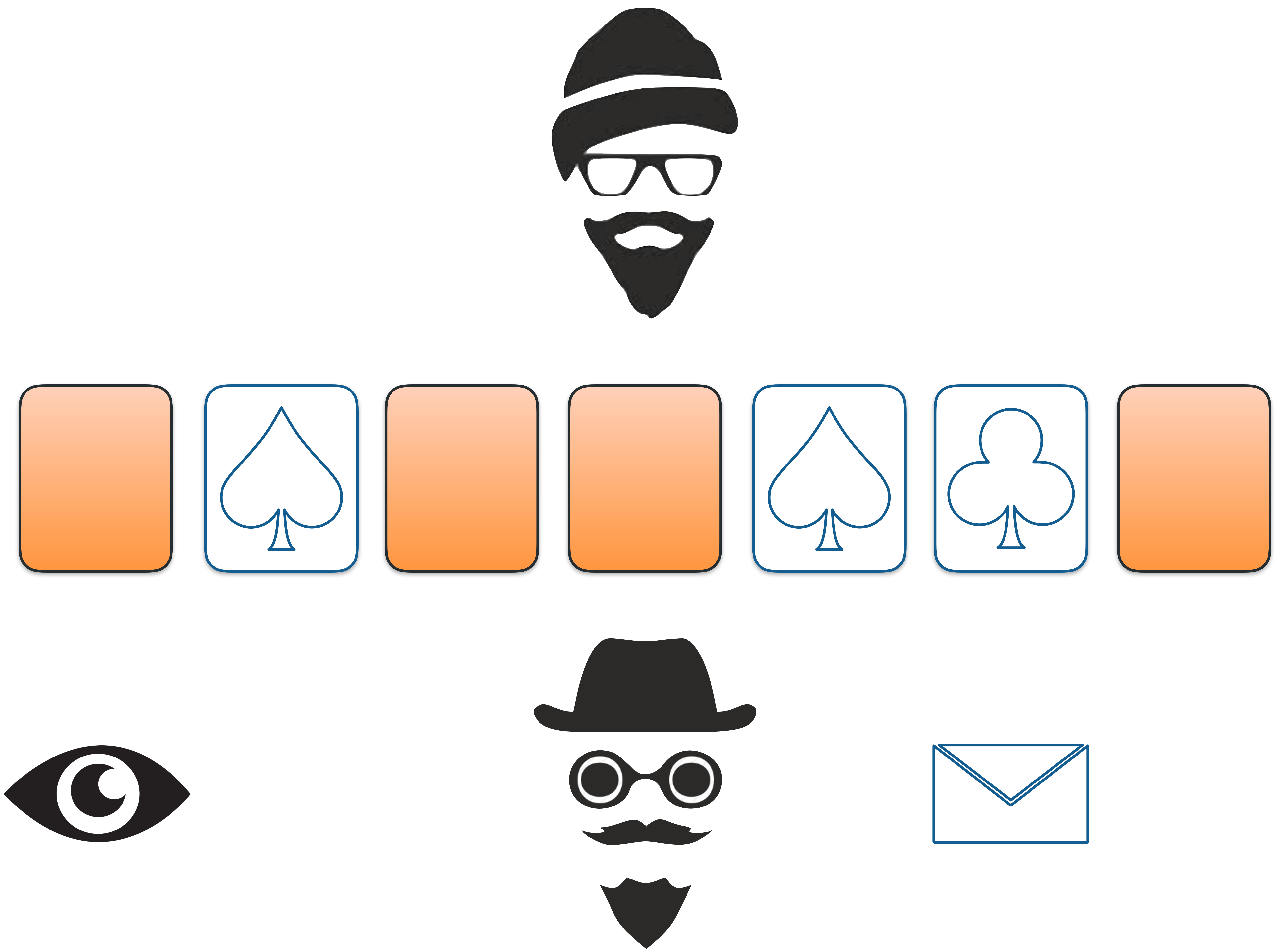
The Hidden-Bit Model



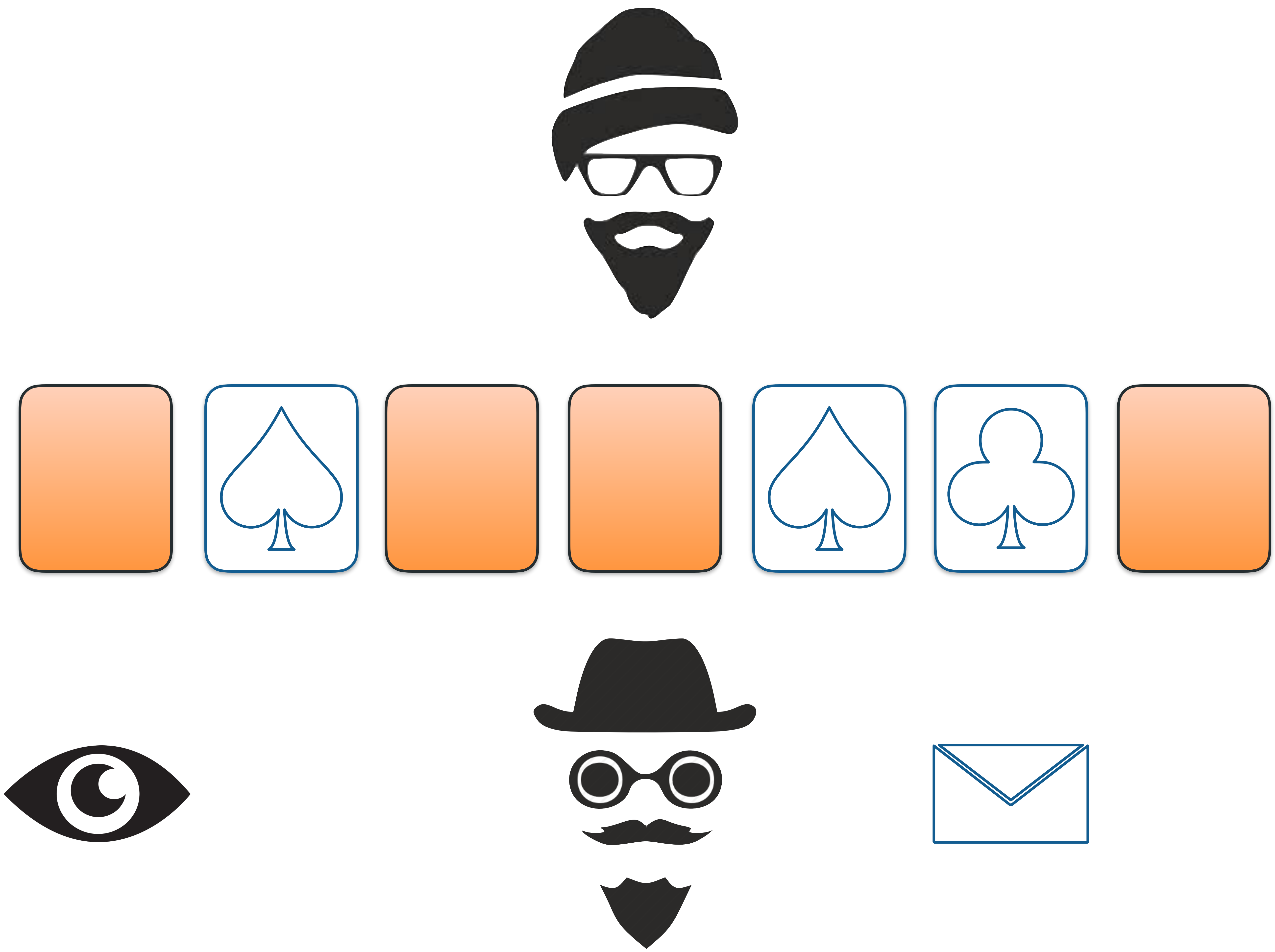
The Hidden-Bit Model



The Hidden-Bit Model

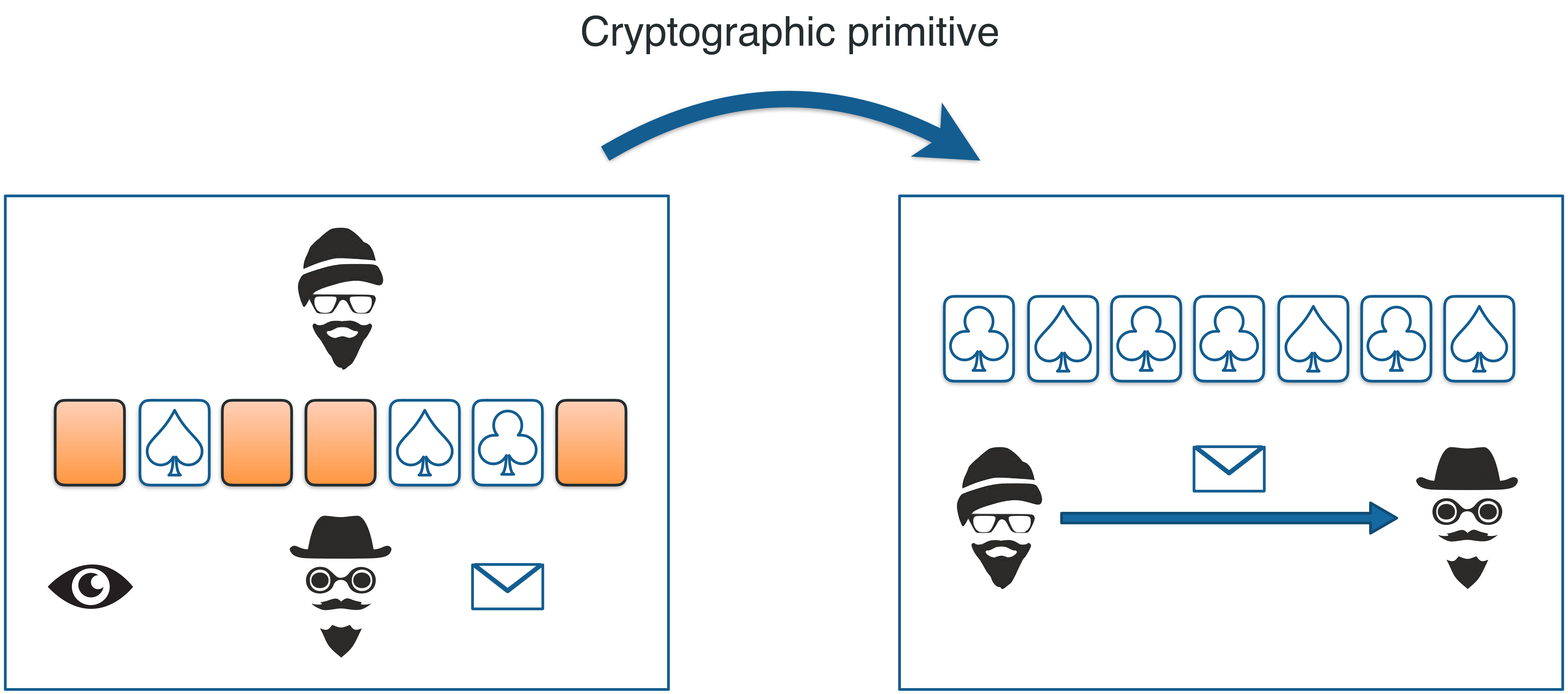


The Hidden-Bit Model



[FLS90]: NIZKs for NP exist unconditionally in the HBM

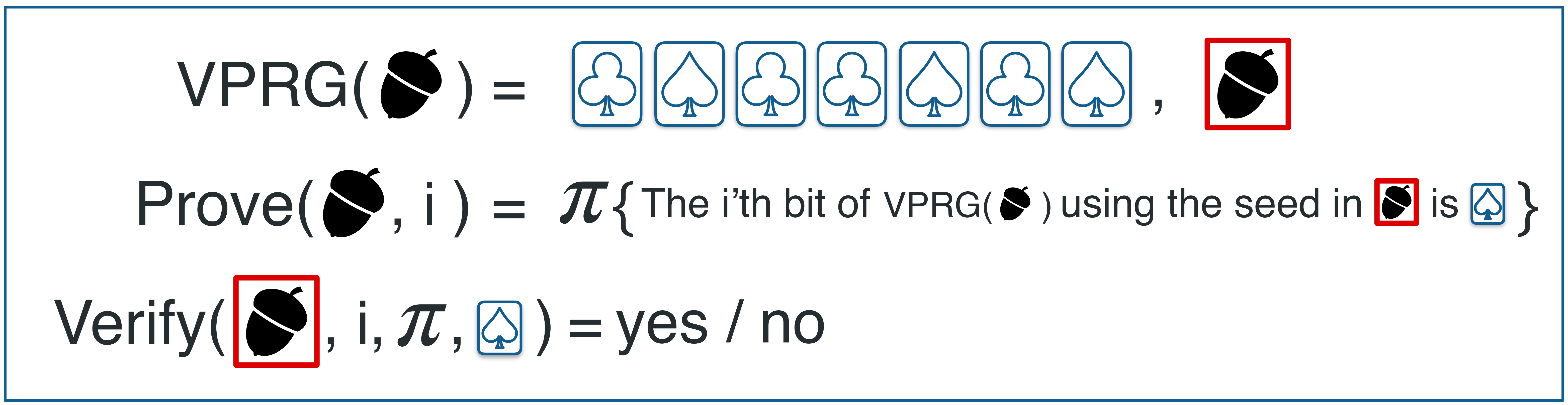
Instantiating The Hidden-Bit Model



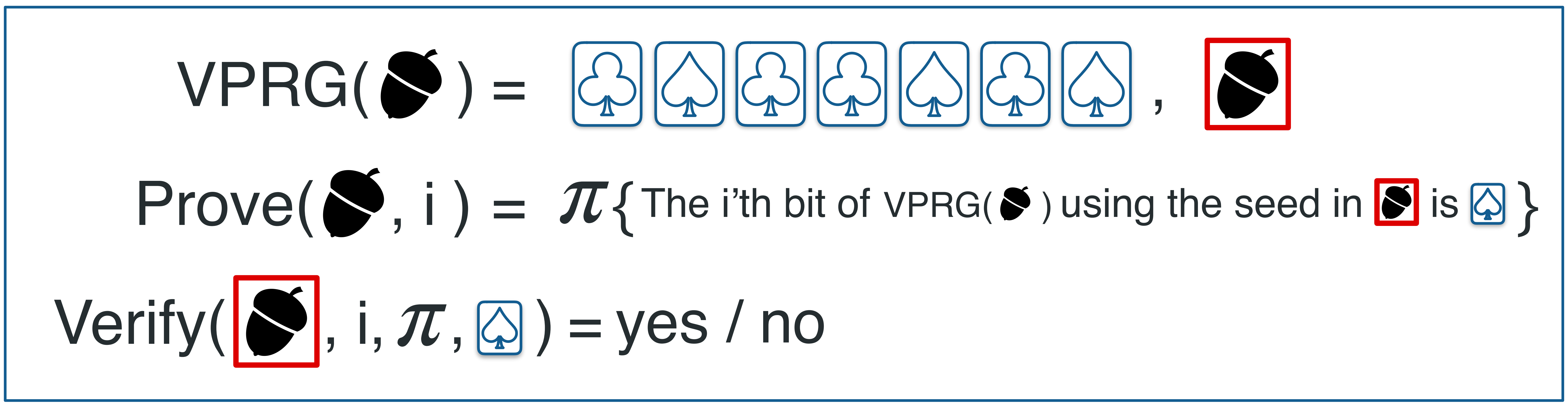
Prover’s task, given the CRS:

1. Produce a string which is indistinguishable from random
2. Be able to provably ‘open’ positions of this pseudorandom string
3. The openings should not reveal the non-opened positions

Verifiable Pseudorandom Generators

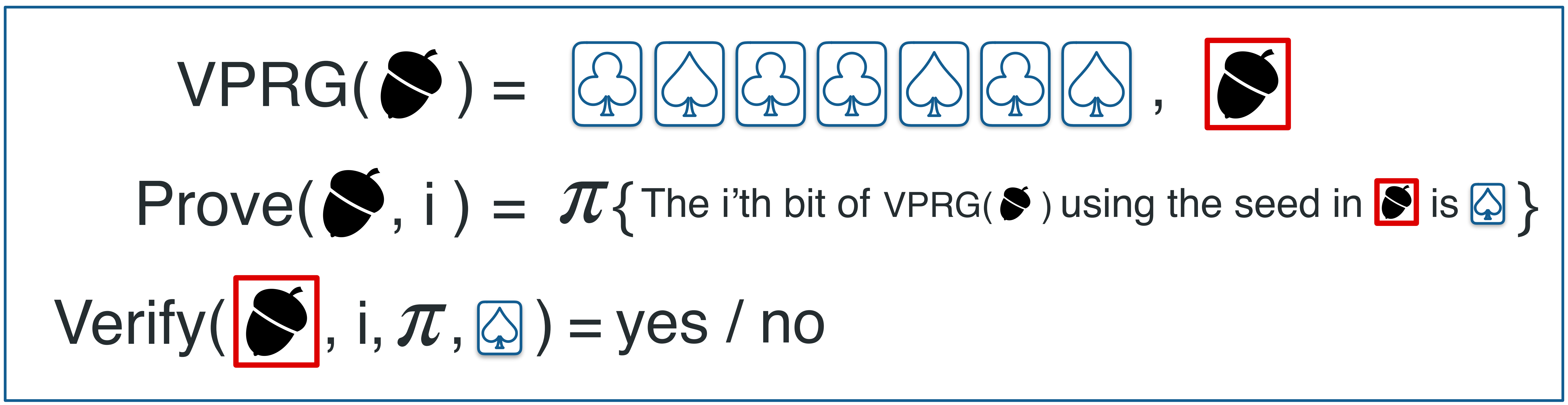


Verifiable Pseudorandom Generators



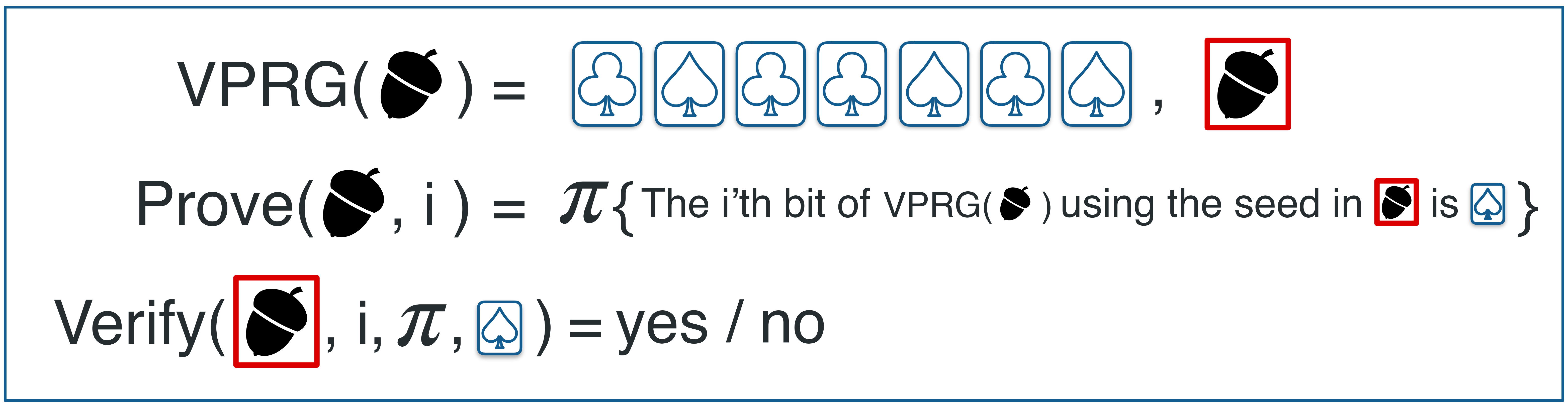
- Is short
- The proof leaks nothing more about
- The proof is sound in a strong sense

Verifiable Pseudorandom Generators



- Is short
- The proof leaks nothing more about
- The proof is sound in a strong sense

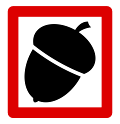
Verifiable Pseudorandom Generators



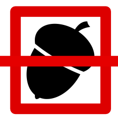
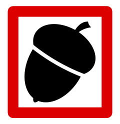
- Is short
- The proof leaks nothing more about
- The proof is sound in a strong sense

1. Every is in the image of VPRG(.)
2. For every possible , there is a unique associated
3. Proofs of opening to bits inconsistent with do not exist

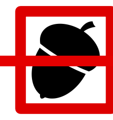
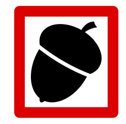
Relaxing VPRGs

1. Every  is in the image of VPRG(.)
2. For every possible  , there is a unique associated ♧♤♧♧♤♧♤
3. Proofs of opening to bits inconsistent with ♧♤♧♧♤♧♤ do not exist

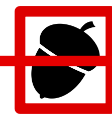
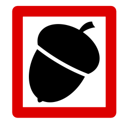
Relaxing VPRGs

1. ~~Every  is in the image of VPRG(.)~~
2. For every possible  , there is a unique associated ♧♤♧♧♤♧♤
3. Proofs of opening to bits inconsistent with ♧♤♧♧♤♧♤ do not exist

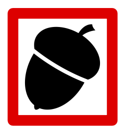
Relaxing VPRGs

~~1. Every  is in the image of VPRG(.)~~
2. For every possible  , there is a unique associated ♣♠♣♣♠♣♠
3'. Proofs of opening to bits inconsistent with ♣♠♣♣♠♣♠ are hard to find

Relaxing VPRGs

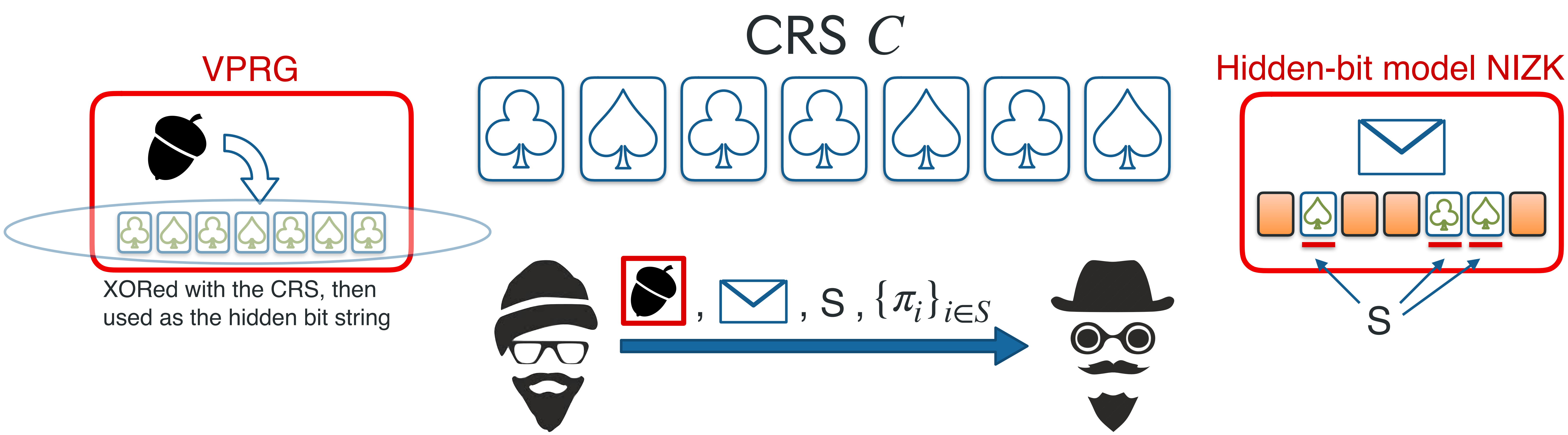
1. ~~Every  is in the image of VPRG(.)~~
2. For every possible  , there is a unique associated ♧♤♧♧♤♧♤

3'. Proofs of opening to bits inconsistent with ♧♤♧♧♤♧♤ are hard to find

4.  is short

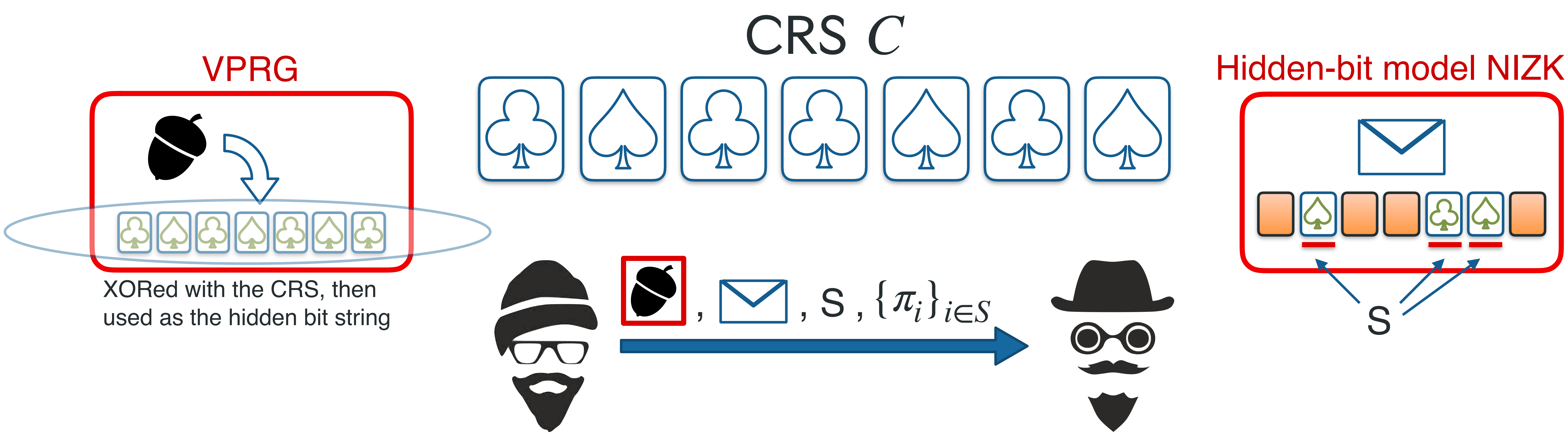
Relaxing VPRGs

1. ~~Every is in the image of VPRG(.)~~
2. For every possible , there is a unique associated
3’. Proofs of opening to bits inconsistent with are hard to find
4. is short



Relaxing VPRGs

1. ~~Every is in the image of VPRG(.)~~
2. For every possible , there is a unique associated
3'. Proofs of opening to bits inconsistent with are hard to find
4. is short



Proof Idea:

- C is 'close to a bad string' if $\exists$, Ext( ) $\oplus$ C is bad
- Proof accepted iff inconsistent opening OR the CRS is « close to a bad string » (requires (2))
- Inconsistent opening → contradiction to VPRG (3')
- Since is short, few CRS are close to a bad string.

Main Instantiation: DVPRG from CDH

CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

Main Instantiation: DVPRG from CDH

CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

[CKS08], gap twin-CDH: given random g, g^a, g^b, g^c, it is hard to find g^{ab}, g^{ac} even given an oracle for the twin-DDH problem

CDH $\Longleftrightarrow$ gap twin-CDH using some secret 'twin-DDH checking key'

Main Instantiation: DVPRG from CDH

CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

[CKS08], gap twin-CDH: given random g, g^a, g^b, g^c, it is hard to find g^{ab}, g^{ac} even given an oracle for the twin-DDH problem

CDH ⟺ gap twin-CDH using some secret 'twin-DDH checking key'

[GL89]: explicit predicate B(.) such that given random g, g^a, g^b, g^c, it is hard to find $B(g^{ab}, g^{ac})$ with probability >> 1/2 even given an oracle for the twin-DDH problem

Equivalent to CDH

Main Instantiation: DVPRG from CDH

CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

[CKS08], gap twin-CDH: given random g, g^a, g^b, g^c, it is hard to find g^{ab}, g^{ac} even given an oracle for the twin-DDH problem

CDH ⟺ gap twin-CDH using some secret 'twin-DDH checking key'

[GL89]: explicit predicate B(.) such that given random g, g^a, g^b, g^c, it is hard to find $B(g^{ab}, g^{ac})$ with probability >> 1/2 even given an oracle for the twin-DDH problem

Equivalent to CDH

Main Instantiation: DVPRG from CDH

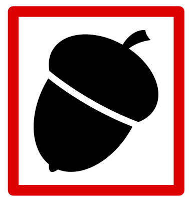
CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

[CKS08], gap twin-CDH: given random g, g^a, g^b, g^c, it is hard to find g^{ab}, g^{ac} even given an oracle for the twin-DDH problem

CDH ⇔ gap twin-CDH using some secret 'twin-DDH checking key'

[GL89]: explicit predicate B(.) such that given random g, g^a, g^b, g^c, it is hard to find $B(g^{ab}, g^{ac})$ with probability >> 1/2 even given an oracle for the twin-DDH problem

Equivalent to CDH

(red oval) = 

(green oval) = public parameters

Main Instantiation: DVPRG from CDH

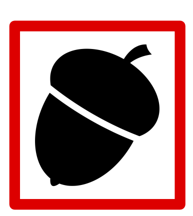
CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

[CKS08], gap twin-CDH: given random g, g^a, g^b, g^c, it is hard to find g^{ab}, g^{ac} even given an oracle for the twin-DDH problem

CDH ⇔ gap twin-CDH using some secret 'twin-DDH checking key'

[GL89]: explicit predicate B(.) such that given random g, g^a, g^b, g^c, it is hard to find $B(g^{ab}, g^{ac})$ with probability >> 1/2 even given an oracle for the twin-DDH problem

Equivalent to CDH

= 

= public parameters

= pseudorandom bit associated to with respect to

Main Instantiation: DVPRG from CDH

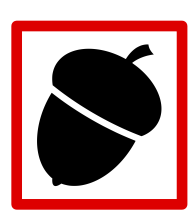
CDH over a group $\mathbb{G}$ states that given random g, g^a, g^b, it is hard to find g^{ab}

[CKS08], gap twin-CDH: given random g, g^a, g^b, g^c, it is hard to find g^{ab}, g^{ac} even given an oracle for the twin-DDH problem

CDH ⟺ gap twin-CDH using some secret 'twin-DDH checking key'

[GL89]: explicit predicate B(.) such that given random g, g^a, g^b, g^c, it is hard to find $B(g^{ab}, g^{ac})$ with probability >> 1/2 even given an oracle for the twin-DDH problem

Equivalent to CDH

= 

= public parameters

= pseudorandom bit associated to (red) with respect to (green)

Proof: g^{ab}, g^{ac}
+ twin-DDH check

Part II: Malicious Designated-Verifier NIZKs

Reusable Designated-Verifier NIZKs for all NP from CDH

Willy Quach
Northeastern

Ron D. Rothblum
Technion

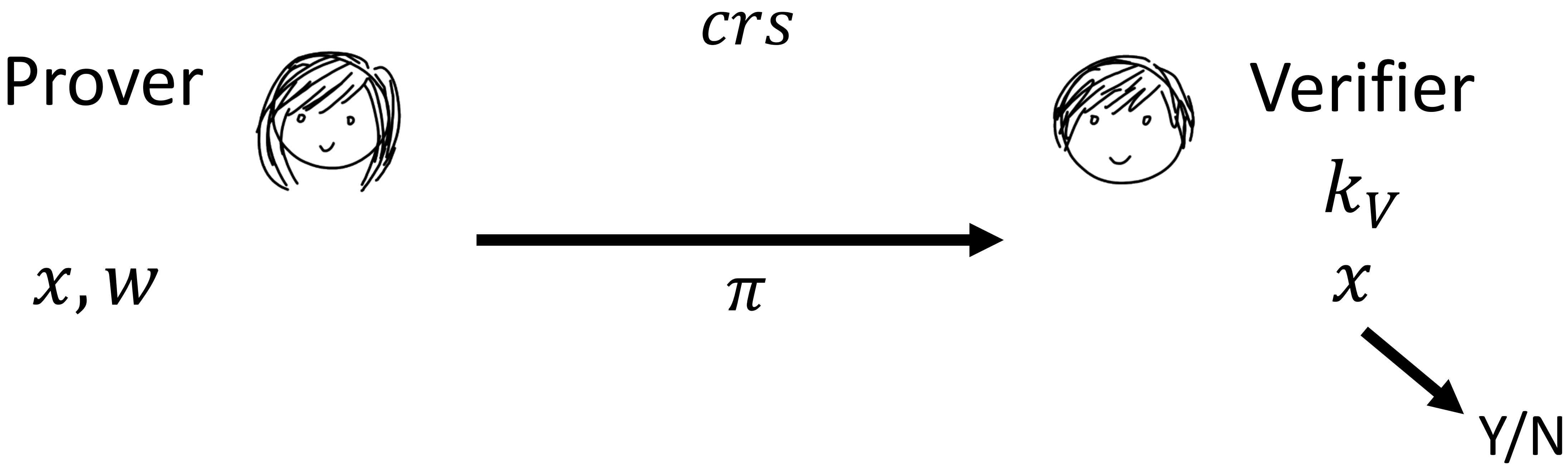
Daniel Wichs
Northeastern

Designated-Verifier NIZK

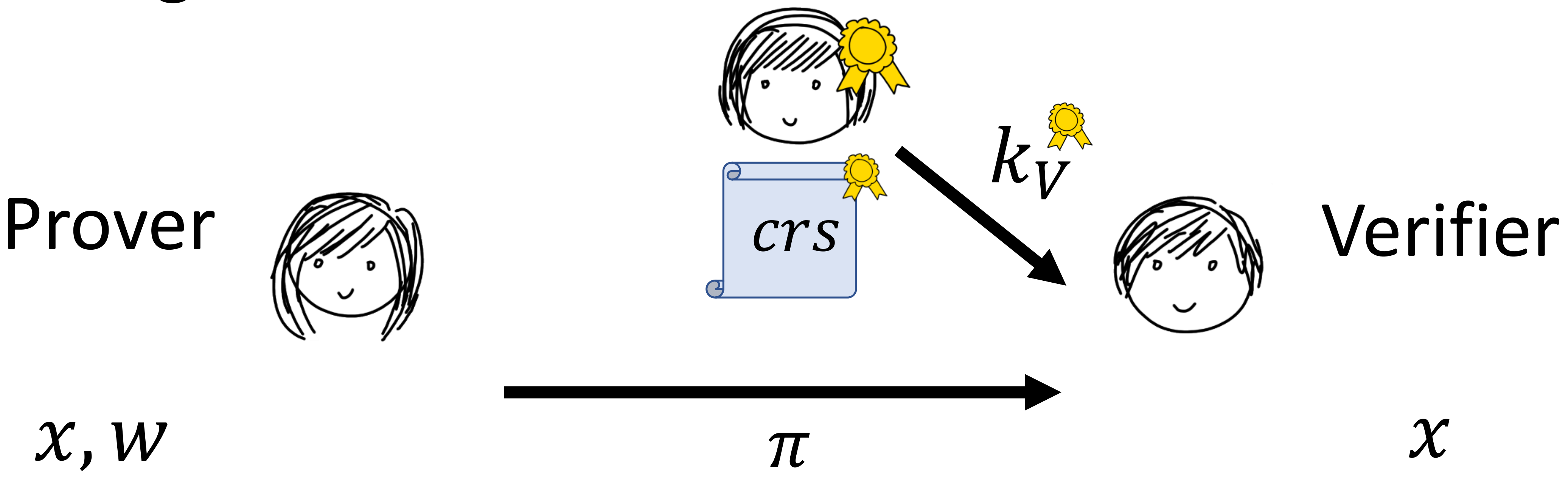
Prover

Verifier

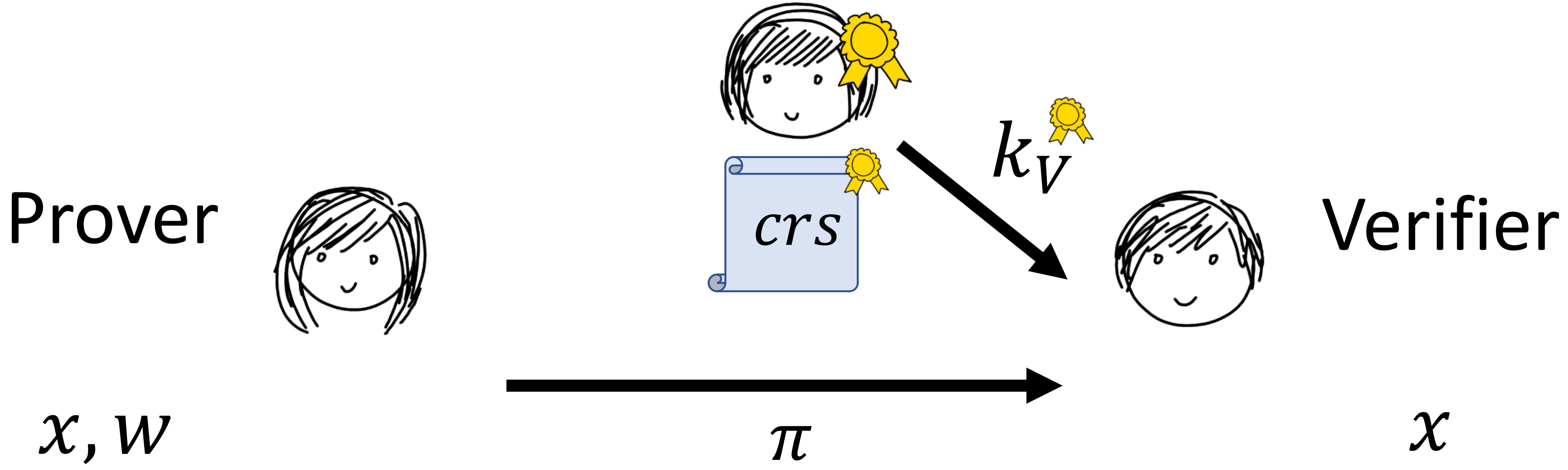
Designated-Verifier NIZK



Designated-Verifier NIZK

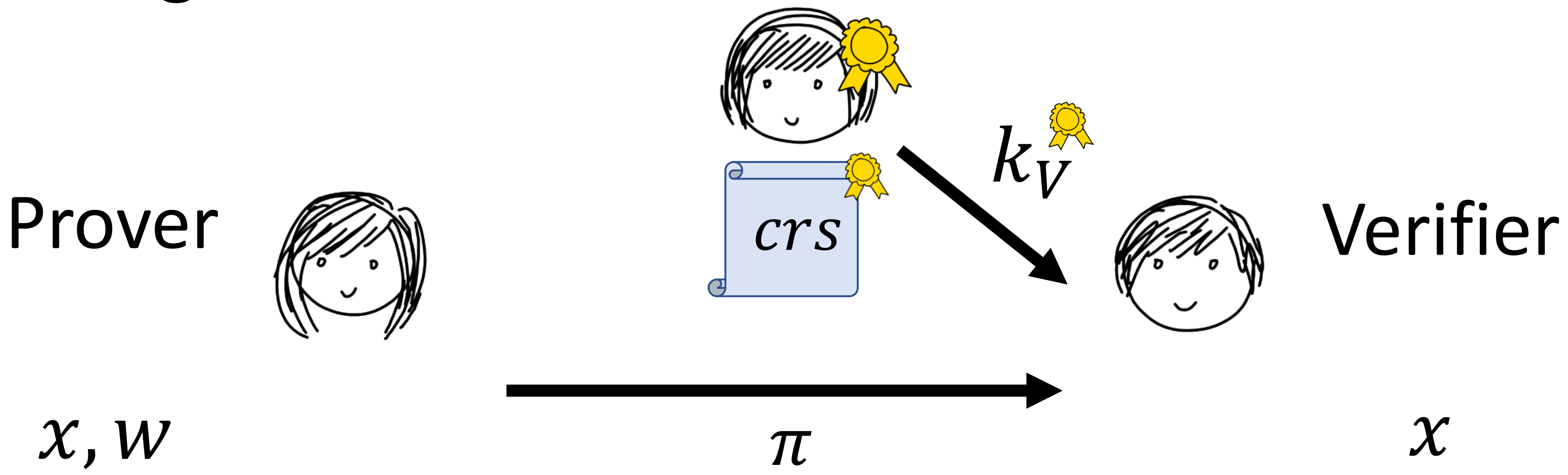


Designated-Verifier NIZK



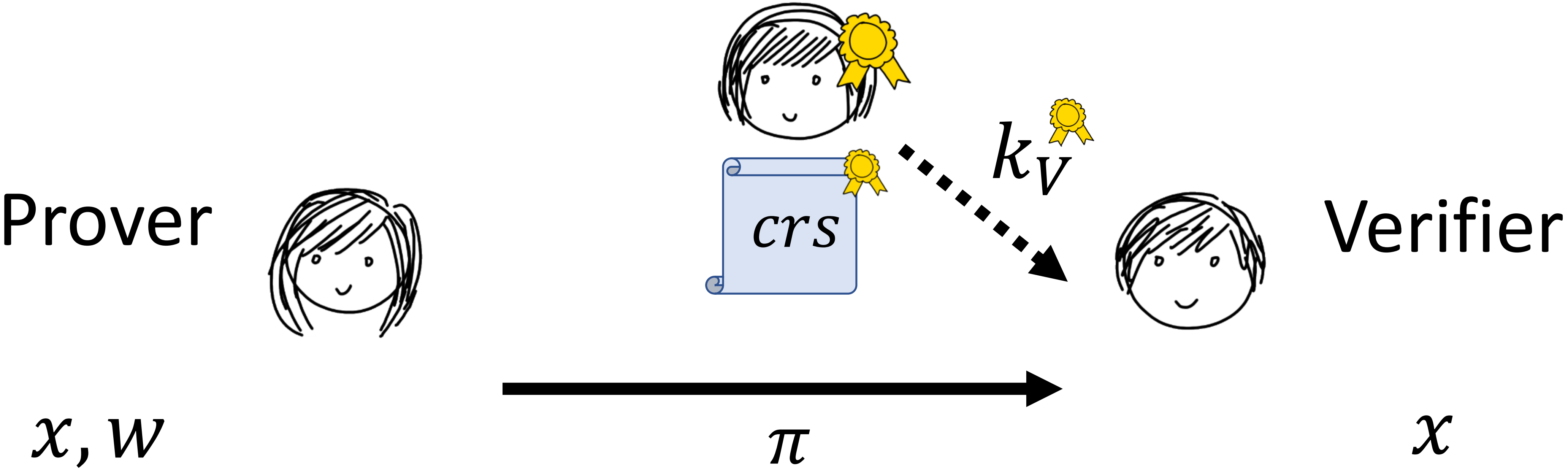
- Need **complex setup** that **interacts** with Verifiers

Designated-Verifier NIZK



- Need **complex setup** that **interacts** with Verifiers
- Simpler setup?

Designated-Verifier NIZK



- Need **complex setup** that **interacts** with Verifiers
- Simpler setup?
  - Setup of a NIZK?

Malicious Designated-Verifier NIZK (MDV-NIZK)

Prover

Verifier

x, w

x

- Simple Trusted Setup: only puts a CRS in the sky

Malicious Designated-Verifier NIZK (MDV-NIZK)

Prover

Verifier

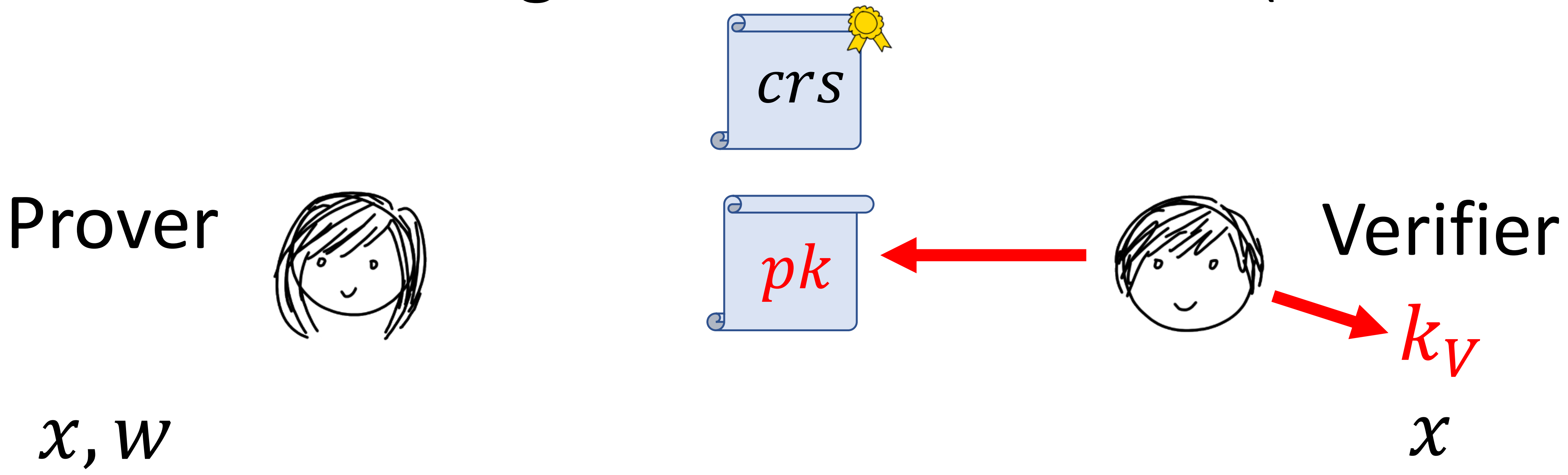
→ k_V

x, w

x

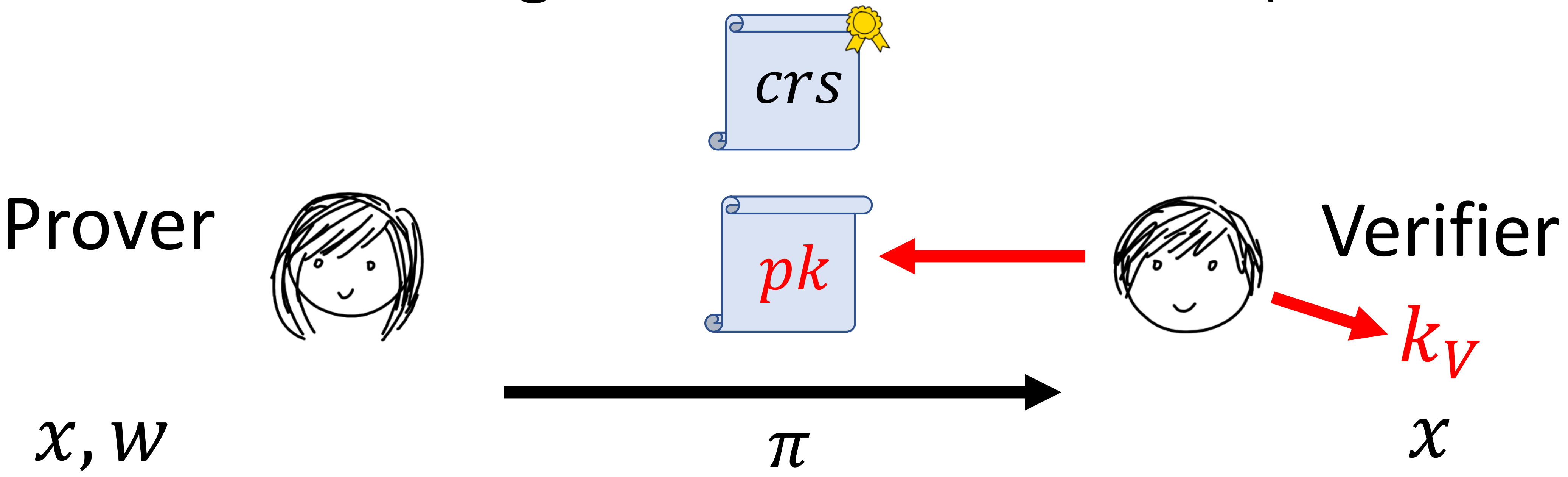
- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks a secret key himself

Malicious Designated-Verifier NIZK (MDV-NIZK)



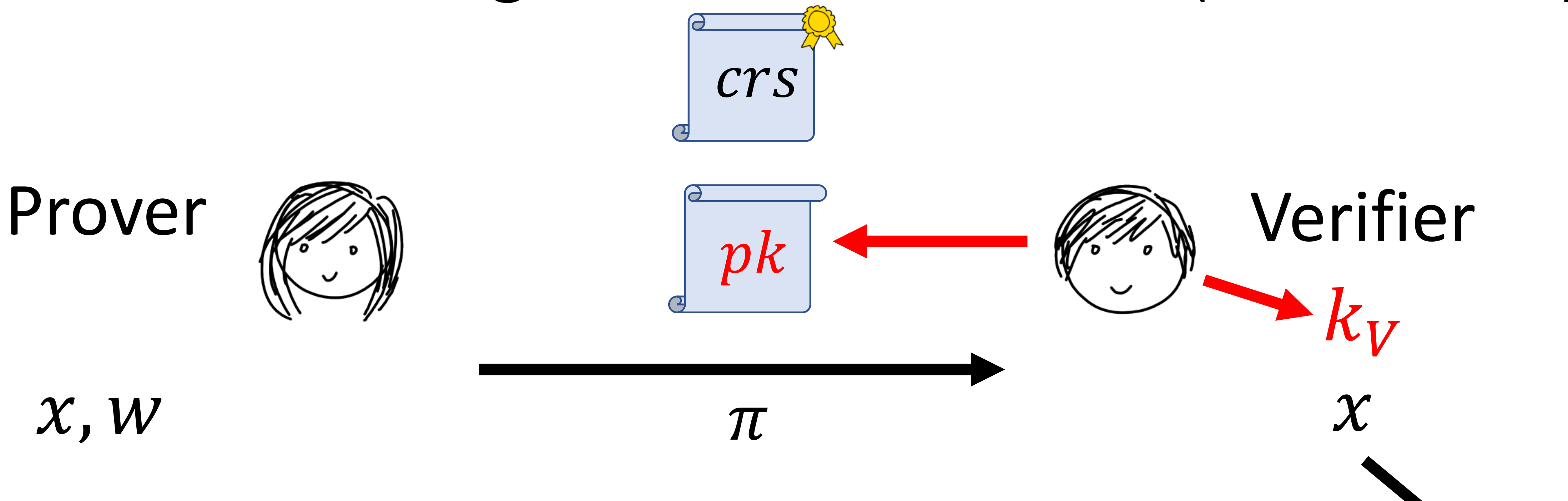
- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself

Malicious Designated-Verifier NIZK (MDV-NIZK)



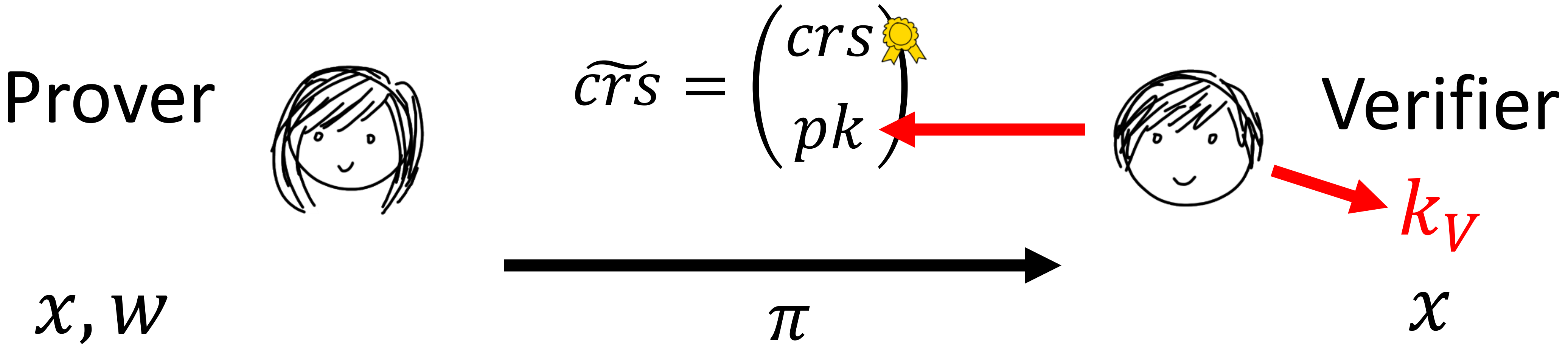
- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself
- (Any) Prover uses (crs, pk) to generate proofs

Malicious Designated-Verifier NIZK (MDV-NIZK)



- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself
- (Any) Prover uses (crs, pk) to generate proofs

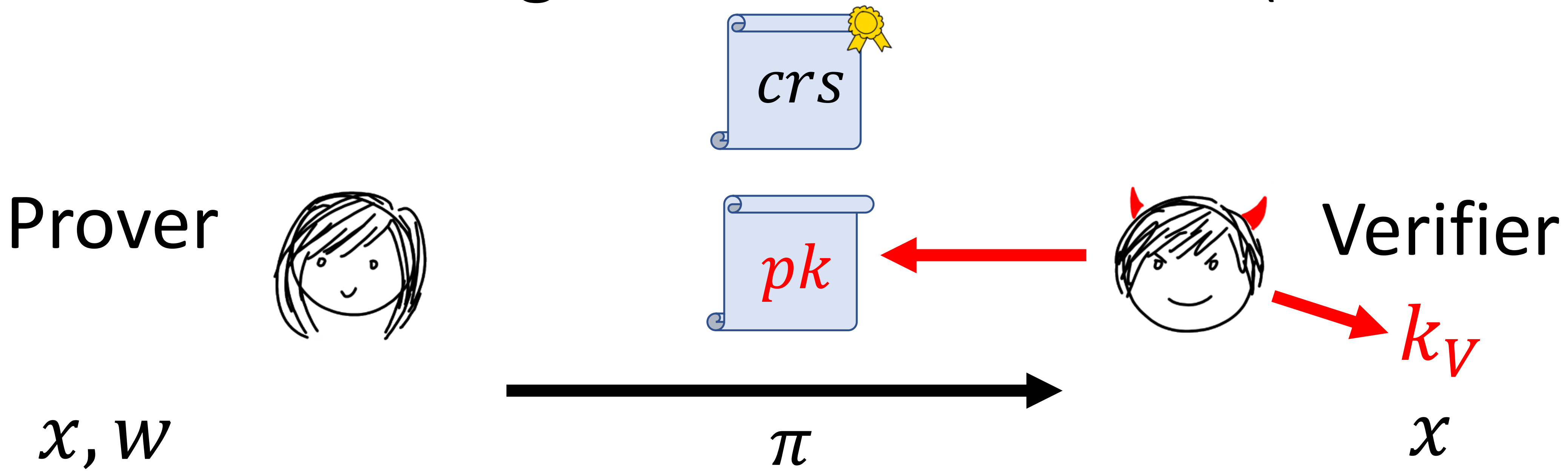
Malicious Designated-Verifier NIZK (MDV-NIZK)



- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself
- (Any) Prover uses (crs, pk) to generate proofs

Syntax: DV-NIZK-like

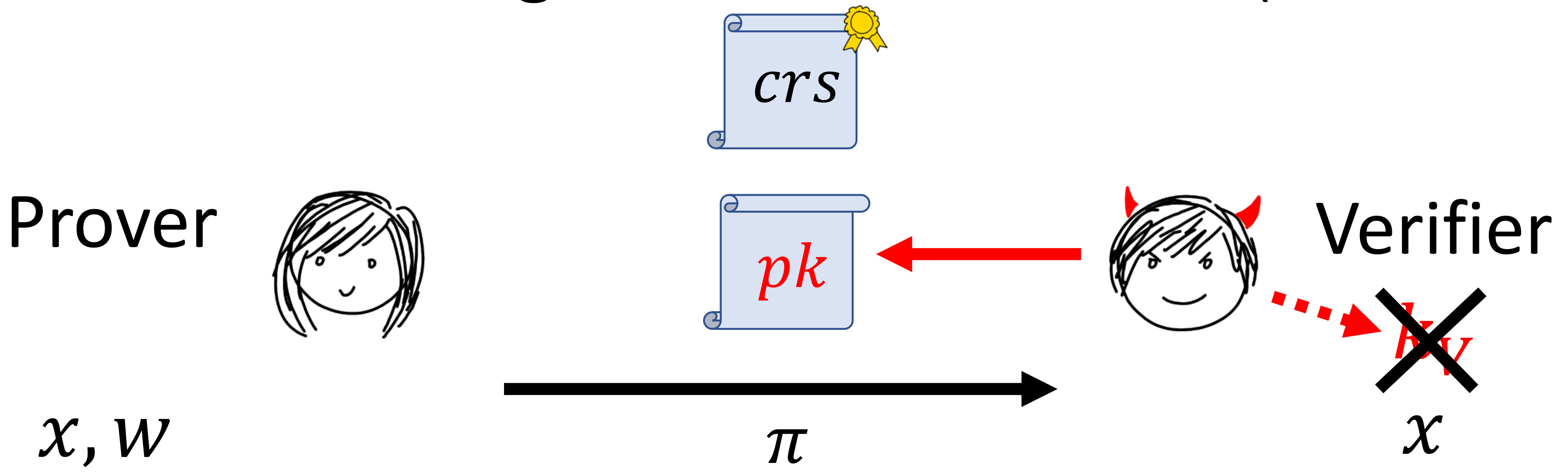
Malicious Designated-Verifier NIZK (MDV-NIZK)



- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself
- (Any) Prover uses (crs, pk) to generate proofs
- **Zero-Knowledge**?

Syntax: DV-NIZK-like

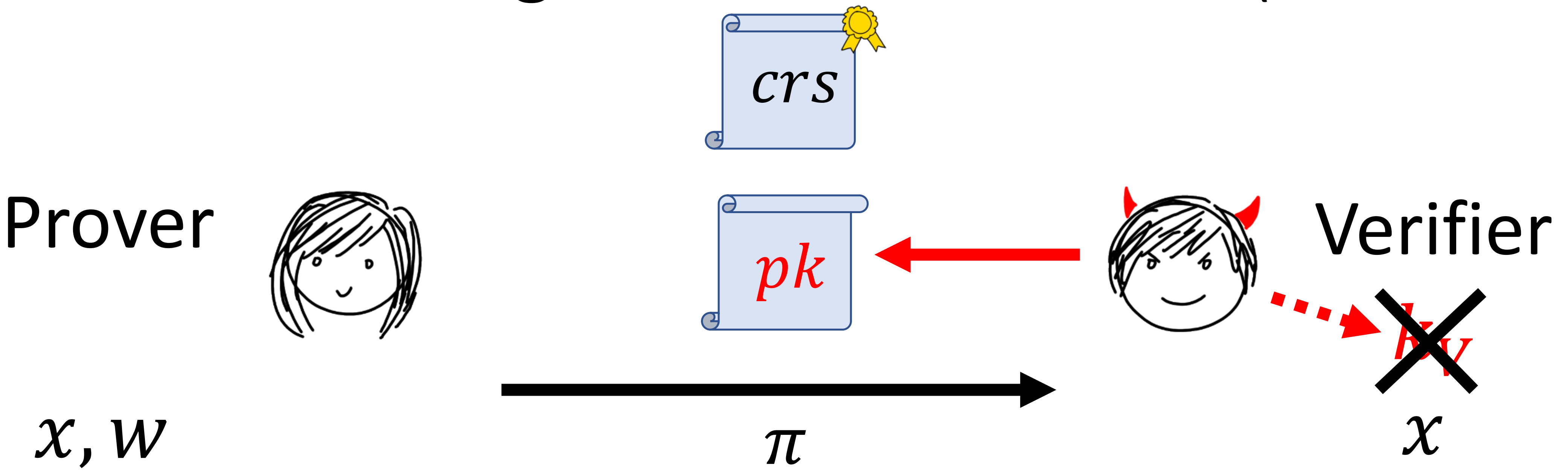
Malicious Designated-Verifier NIZK (MDV-NIZK)



- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks pk himself
- (Any) Prover uses (crs, pk) to generate proofs
- **Zero-Knowledge**?

Syntax: DV-NIZK-like

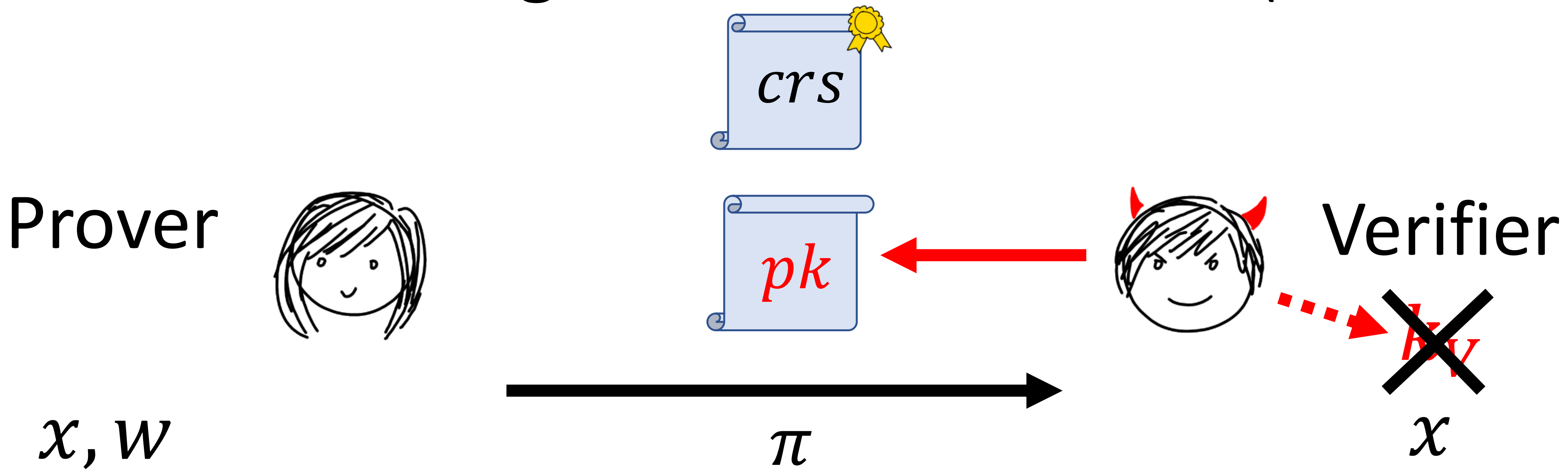
Malicious Designated-Verifier NIZK (MDV-NIZK)



- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks pk himself
- (Any) Prover uses (crs, pk) to generate proofs
- **Zero-Knowledge** against **malicious verifiers**

Syntax: DV-NIZK-like

Malicious Designated-Verifier NIZK (MDV-NIZK)

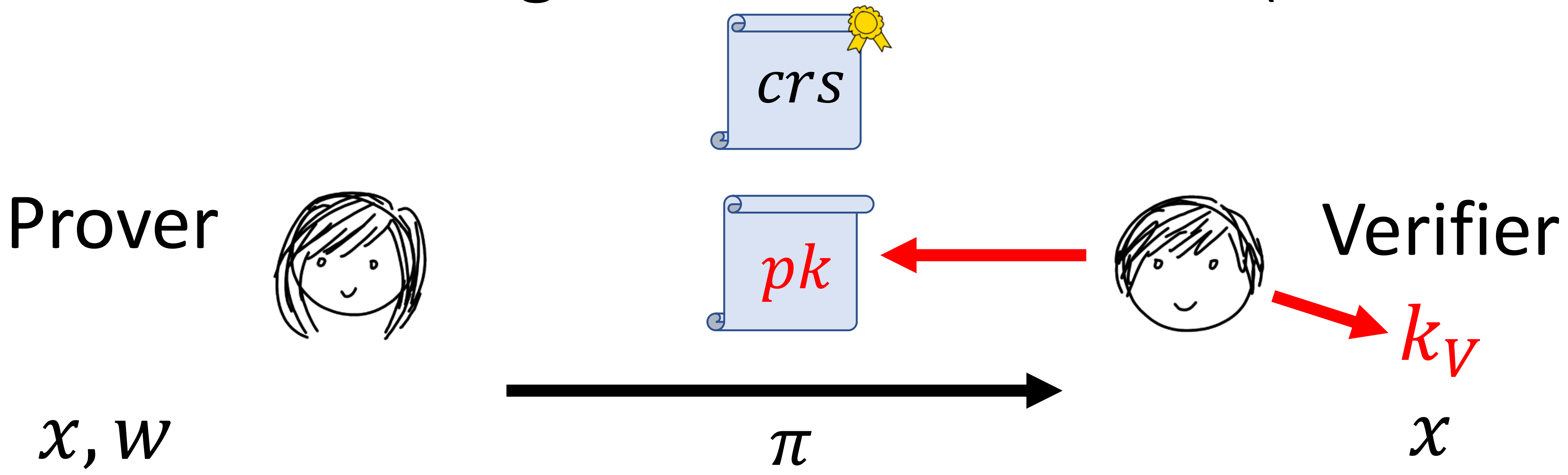


- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks pk himself
- (Any) Prover uses (crs, pk) to generate proofs
- **Zero-Knowledge** against **malicious verifiers**

Syntax: DV-NIZK-like

Security: NIZK-like
(only CRS is trusted)

Malicious Designated-Verifier NIZK (MDV-NIZK)

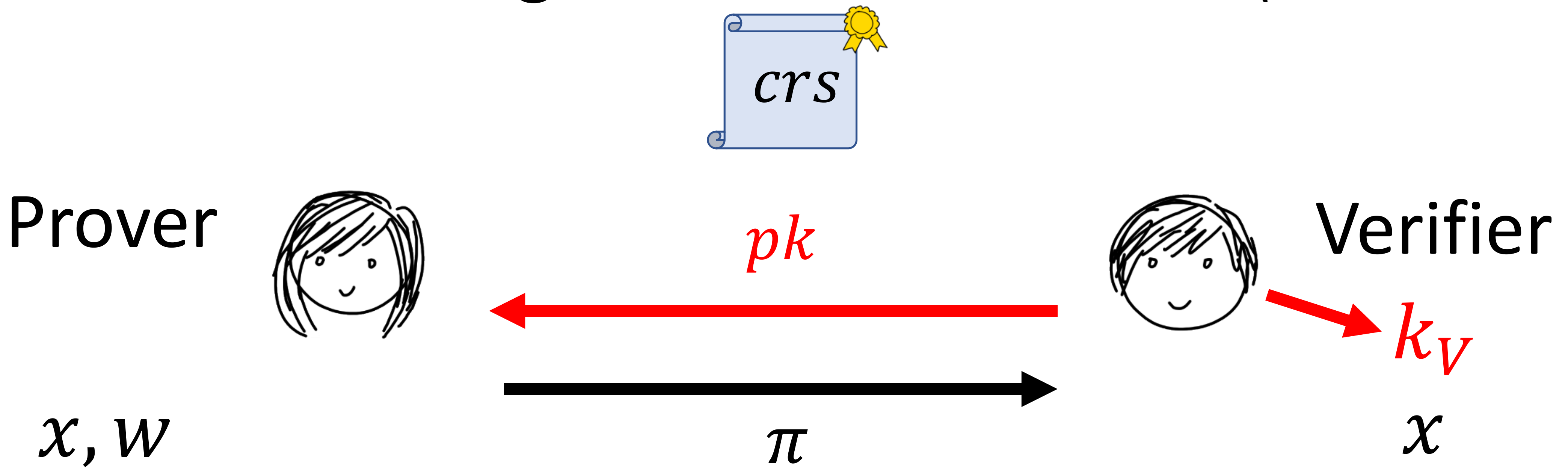


- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself
- (Any) Prover uses (crs, pk) to generate proofs
- **Zero-Knowledge** against **malicious verifiers**

Syntax: DV-NIZK-like

Security: NIZK-like (only CRS is trusted)

Malicious Designated-Verifier NIZK (MDV-NIZK)

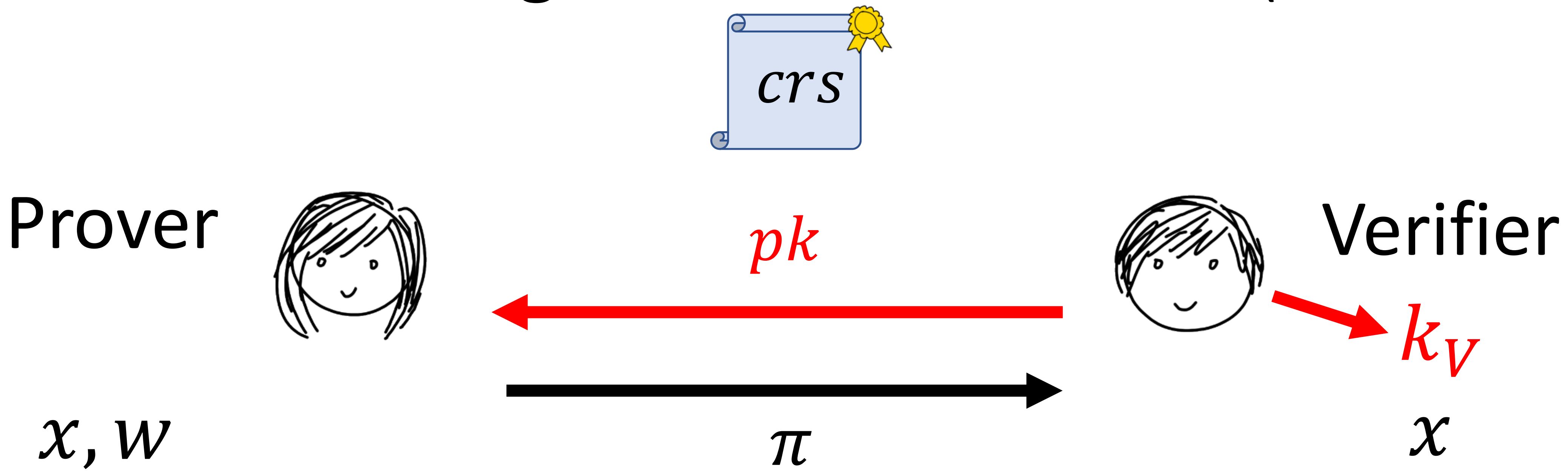


- Simple Trusted Setup: only puts a CRS in the sky
- (Any) Verifier picks (pk, k_V) himself
- (Any) Prover uses (crs, pk) to generate proofs
- **Zero-Knowledge** against **malicious verifiers**

Syntax: DV-NIZK-like

Security: NIZK-like
(only CRS is trusted)

Malicious Designated-Verifier NIZK (MDV-NIZK)



- Simple Trusted Setup: only puts a CRS in the sky

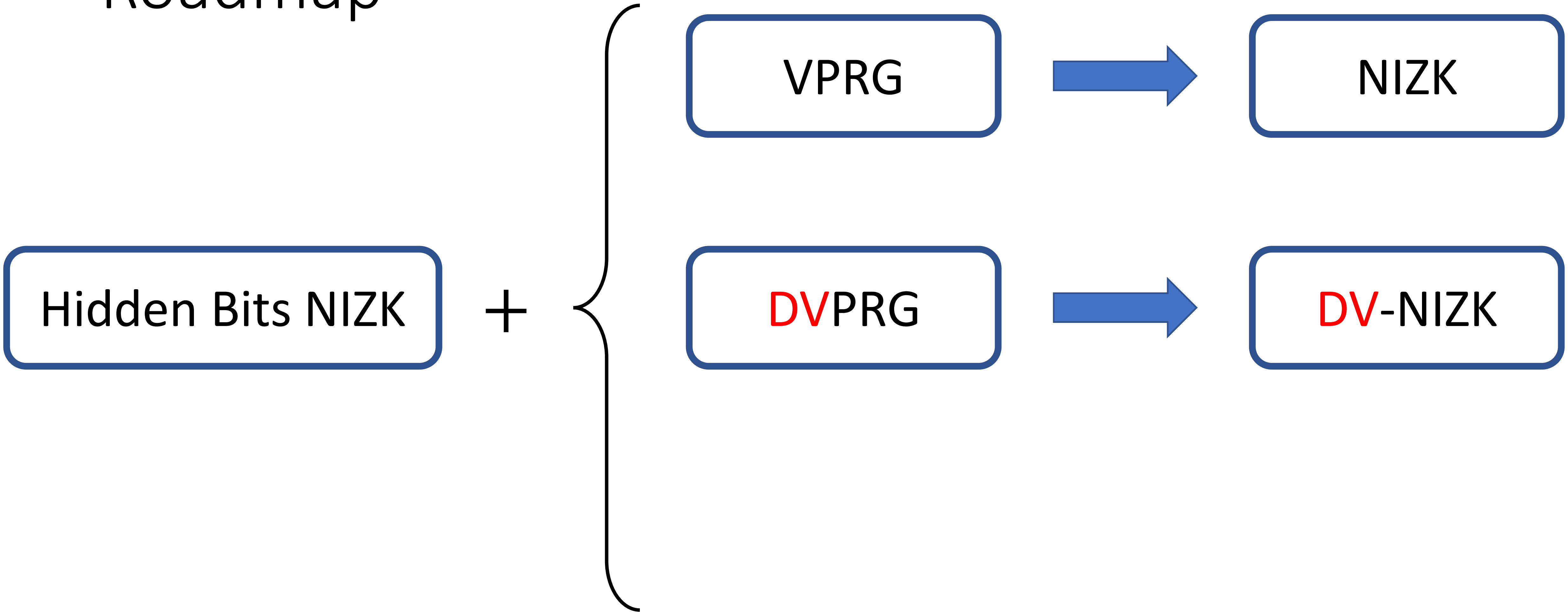
2-round Zero-Knowledge
with **reusable** first message

- **Zero-Knowledge** against **malicious verifiers**

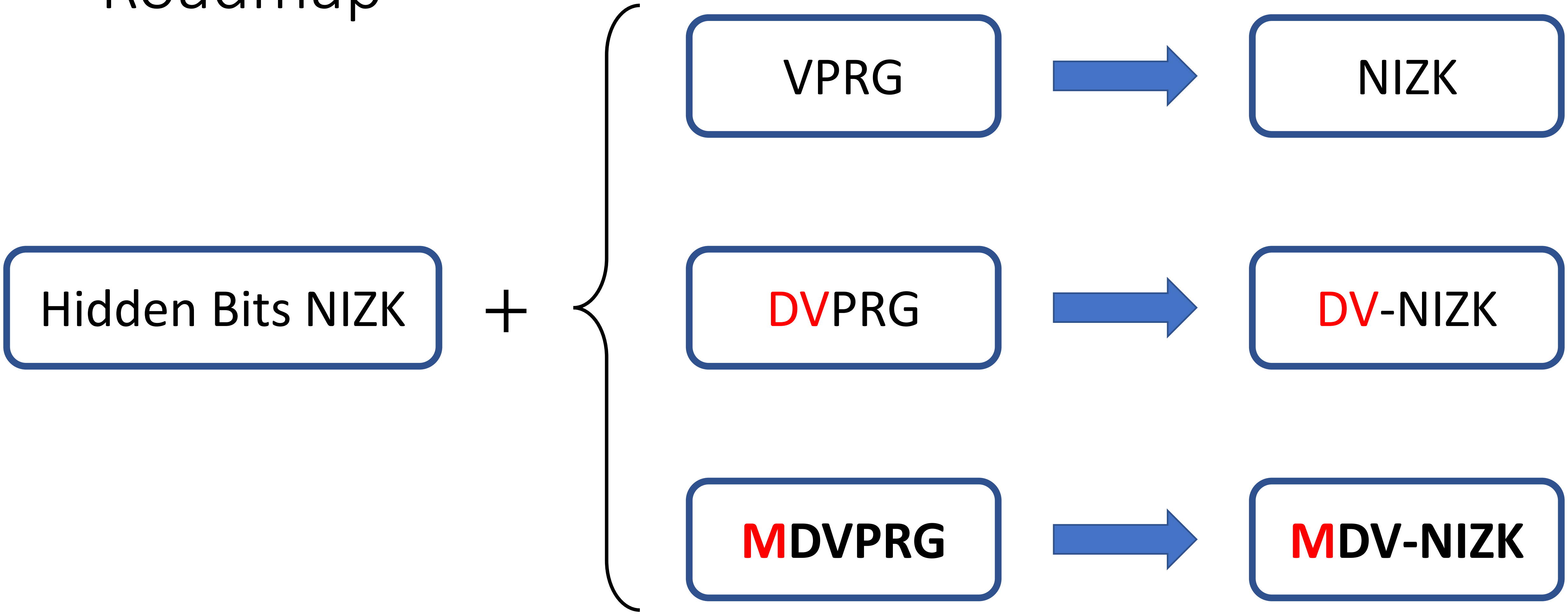
Syntax: DV-NIZK-like

Security: NIZK-like
(only CRS is trusted)

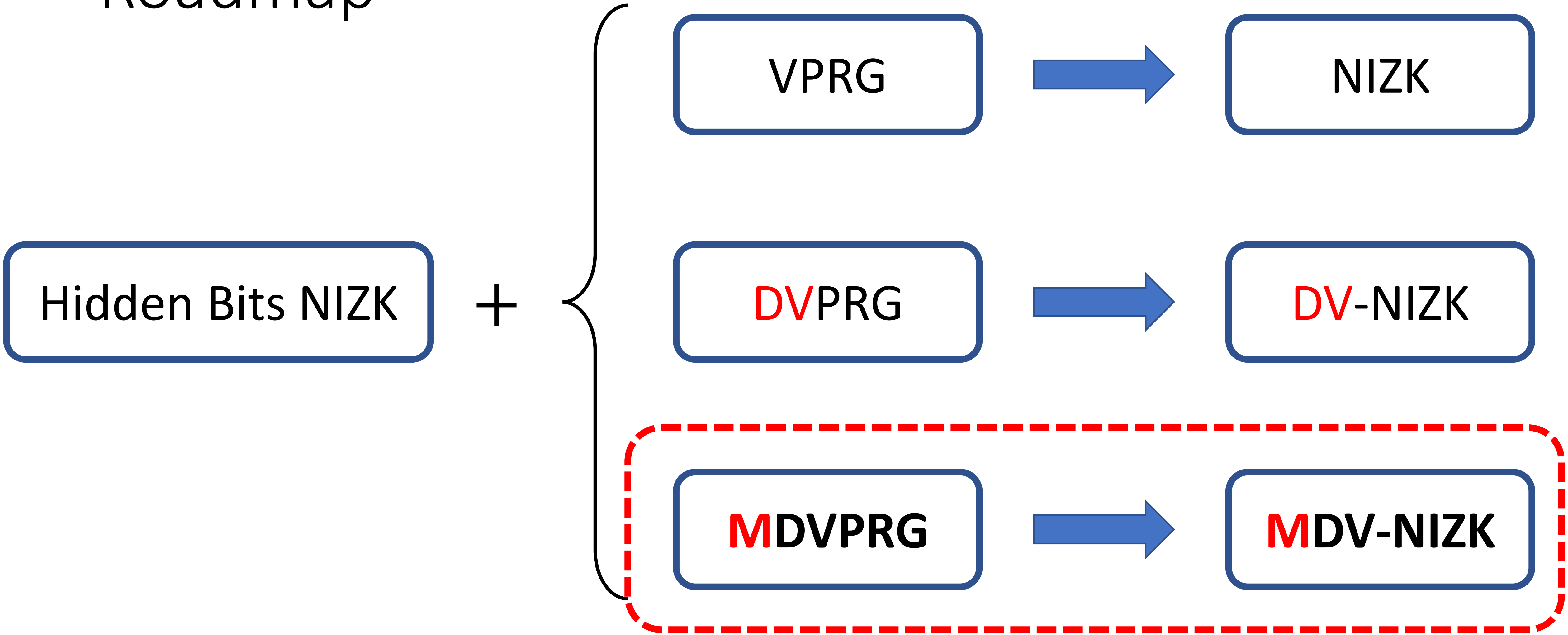
Roadmap



Roadmap



Roadmap



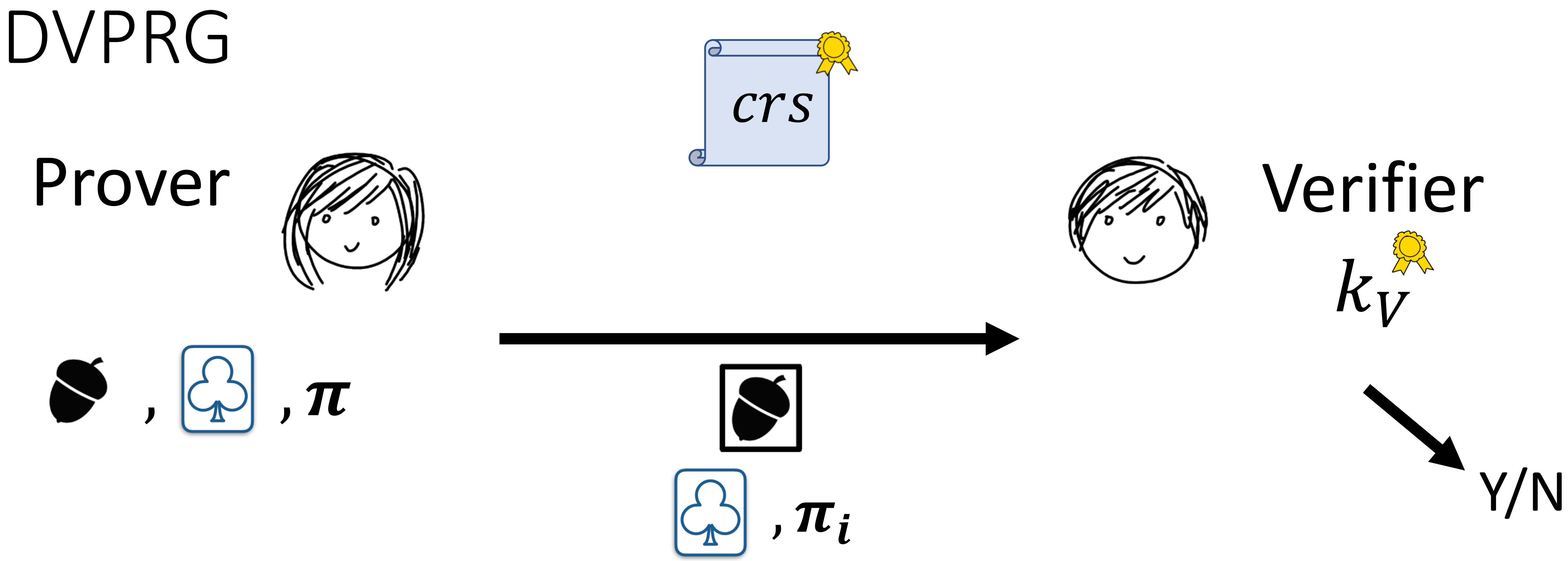
DVPRG

Prover 

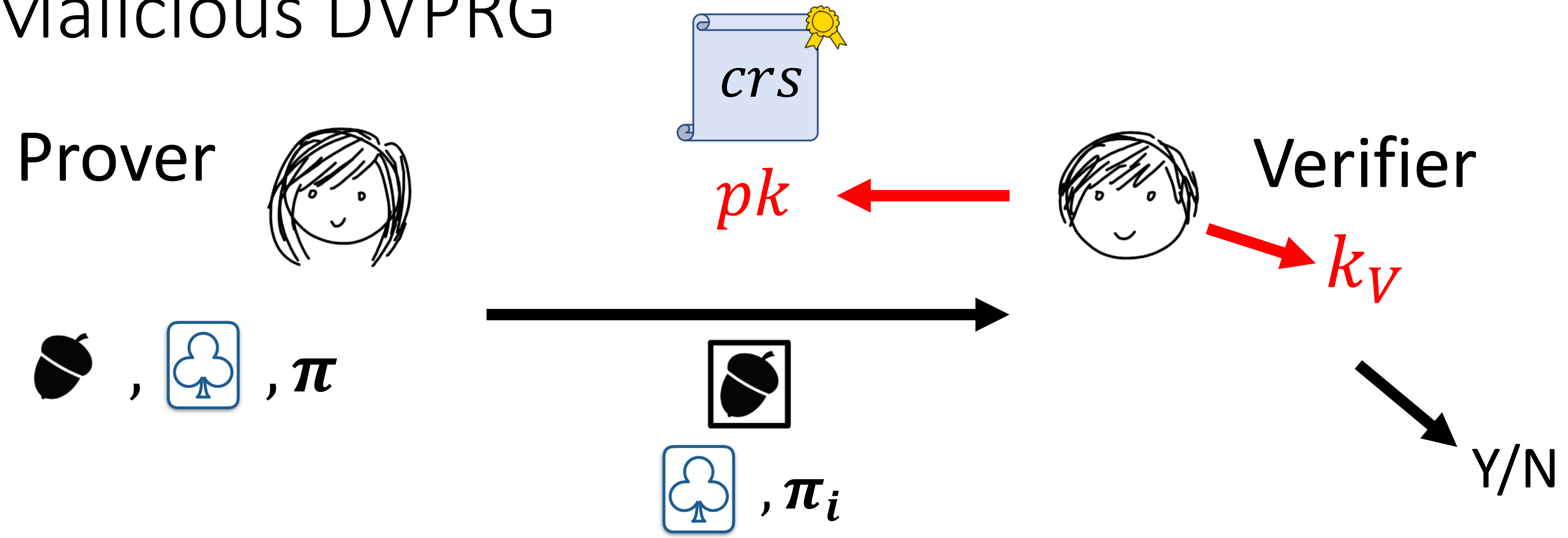
 Verifier

k_V 

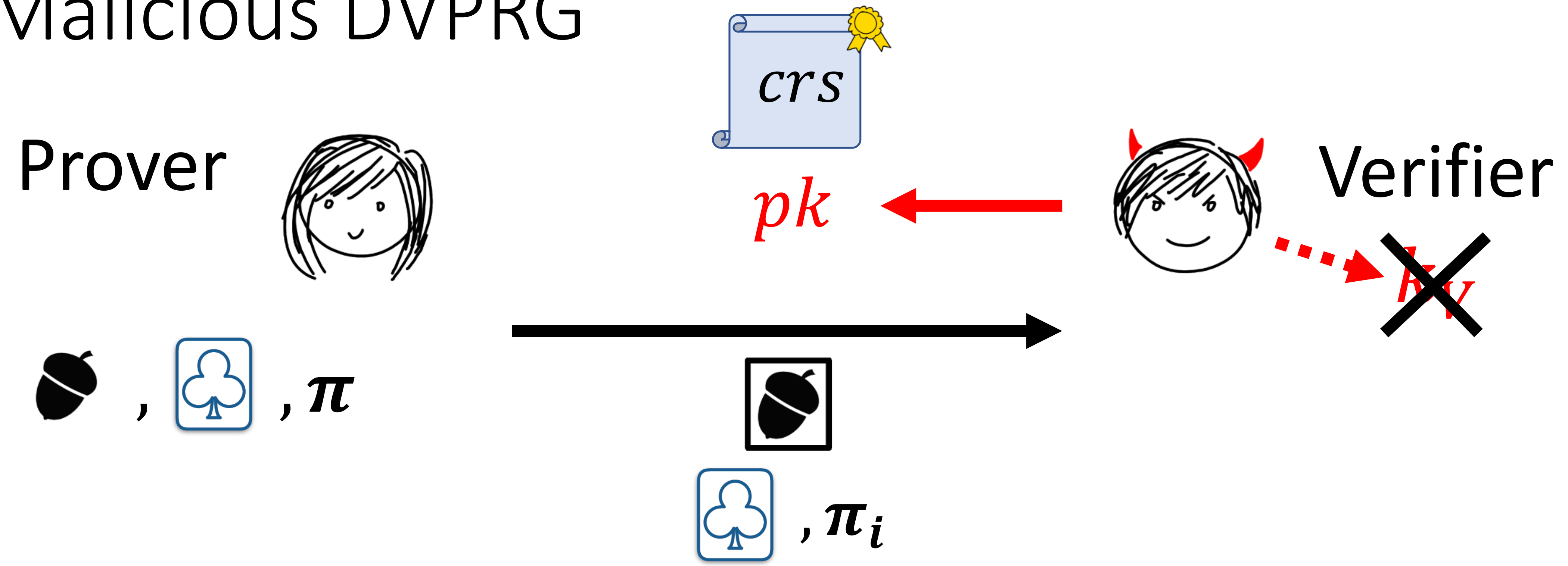
 ,  , $\boldsymbol{\pi}$


DVPRG
Prover
Verifier
crs
k_V
$, \boldsymbol{\pi}$
$, \boldsymbol{\pi_i}$
Y/N

Malicious DVPRG

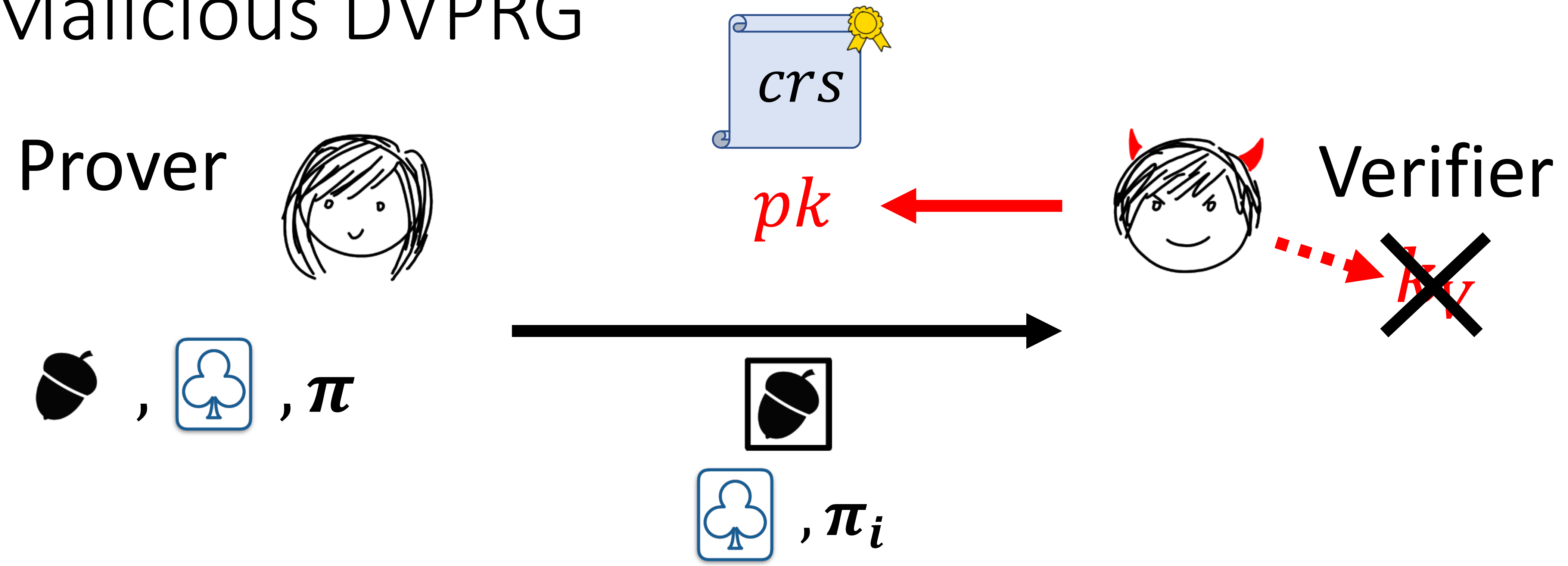


Malicious DVPRG



- **Non-opened bits hidden** against ***malicious* public keys**

Malicious DVPRG



- **Non-opened bits hidden** against ***malicious* public keys**

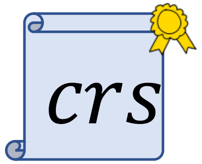
Malicious DVPRG ⇒ Malicious DV-NIZK

MDV-PRG from DDH?

MDV-PRG from DDH?

g^s

MDV-PRG from DDH?



h_1

g^s $+$ $\vdots$

h_k

a.k.a g^{b_i}

MDV-PRG from DDH?

crs

$$g^s \quad + \quad \begin{matrix} h_1 & \longrightarrow & s_1 = h_1^s \\ \vdots & & \vdots \\ h_k & \longrightarrow & s_k = h_k^s \end{matrix}$$

a.k.a g^{b_i}

MDV-PRG from DDH?

crs | crs

g^s + $h_1 \longrightarrow s_1 = h_1^s$

$\vdots \quad \vdots$

$h_k \longrightarrow s_k = h_k^s$

f_1

$\vdots$

f_k

a.k.a g^{b_i}

a.k.a g^{c_i}

MDV-PRG from DDH?

Twin DDH Check
a.k.a
Cramer Shoup proof

crs

$$h_1 \longrightarrow s_1 = h_1^s$$

$$g^s \quad + \quad \vdots \qquad \vdots$$

$$h_k \longrightarrow s_k = h_k^s$$

a.k.a g^{b_i}

crs $\qquad$ $\boldsymbol{\pi}$

$$f_1 \longrightarrow \pi_1 = f_1^s$$

$$\vdots \qquad \vdots$$

$$f_k \longrightarrow \pi_k = f_k^s$$

a.k.a g^{c_i}

MDV-PRG from DDH?

Twin DDH Check
a.k.a
Cramer Shoup proof

crs

pk

$\boldsymbol{\pi}$

$g^s \quad +$

$h_1 \longrightarrow s_1 = h_1^s$

$\vdots \qquad \vdots$

$h_k \longrightarrow s_k = h_k^s$

$f_1 \longrightarrow \pi_1 = f_1^s$

$\vdots \qquad \vdots$

$f_k \longrightarrow \pi_k = f_k^s$

a.k.a g^{b_i}

a.k.a g^{c_i}

MDV-PRG from DDH?

Twin DDH Check
a.k.a
Cramer Shoup proof

crs | pk | $\boldsymbol{\pi}$

$$g^s + \quad h_1 \longrightarrow s_1 = h_1^s \qquad f_1 \longrightarrow \pi_1 = f_1^s$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$h_k \longrightarrow s_k = h_k^s \qquad f_k \longrightarrow \pi_k = f_k^s$$

- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

Twin DDH Check
a.k.a
Cramer Shoup proof

MDV-PRG from DDH?

crs | pk | $\boldsymbol{\pi}$

g^s +

$h_1 \longrightarrow s_1 = h_1^s$

$\vdots$

$h_k \longrightarrow s_k = h_k^s$

$f_1 \longrightarrow$ ~~$\pi_1 = f_1^s$~~

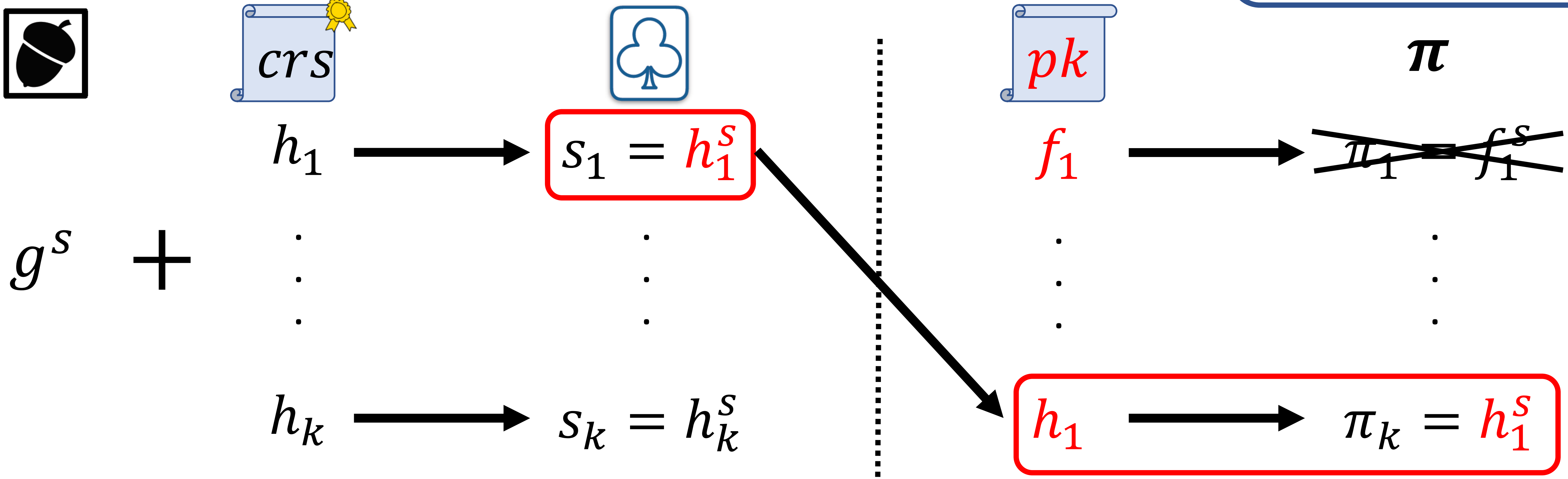
$\vdots$

$f_k \longrightarrow \pi_k = f_k^s$

- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

MDV-PRG from DDH?

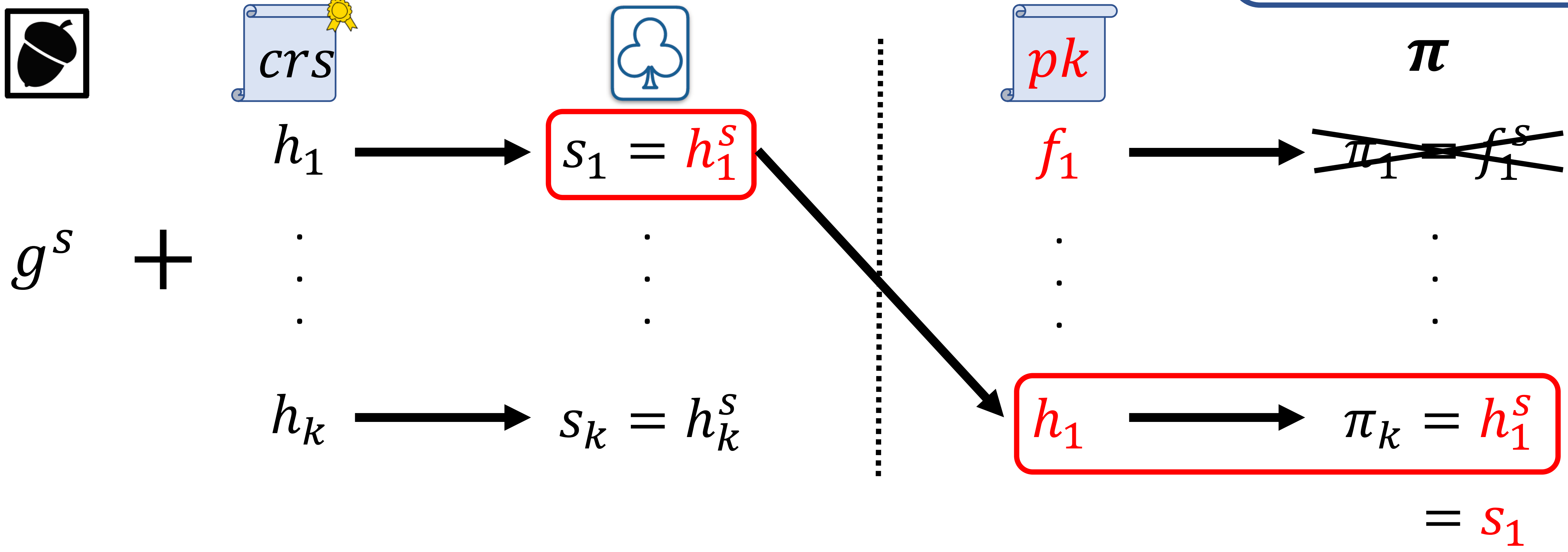
Twin DDH Check
a.k.a
Cramer Shoup proof



- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

MDV-PRG from DDH?

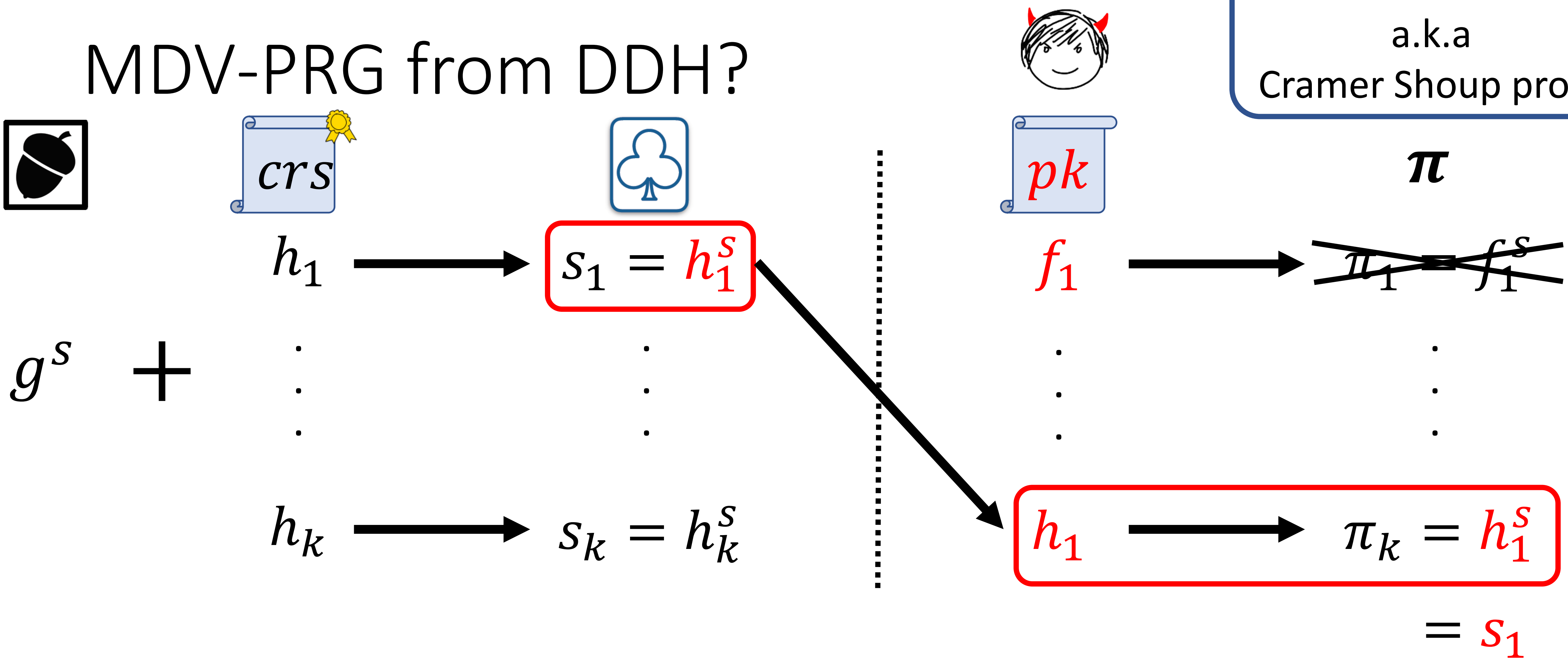
Twin DDH Check
a.k.a
Cramer Shoup proof



- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

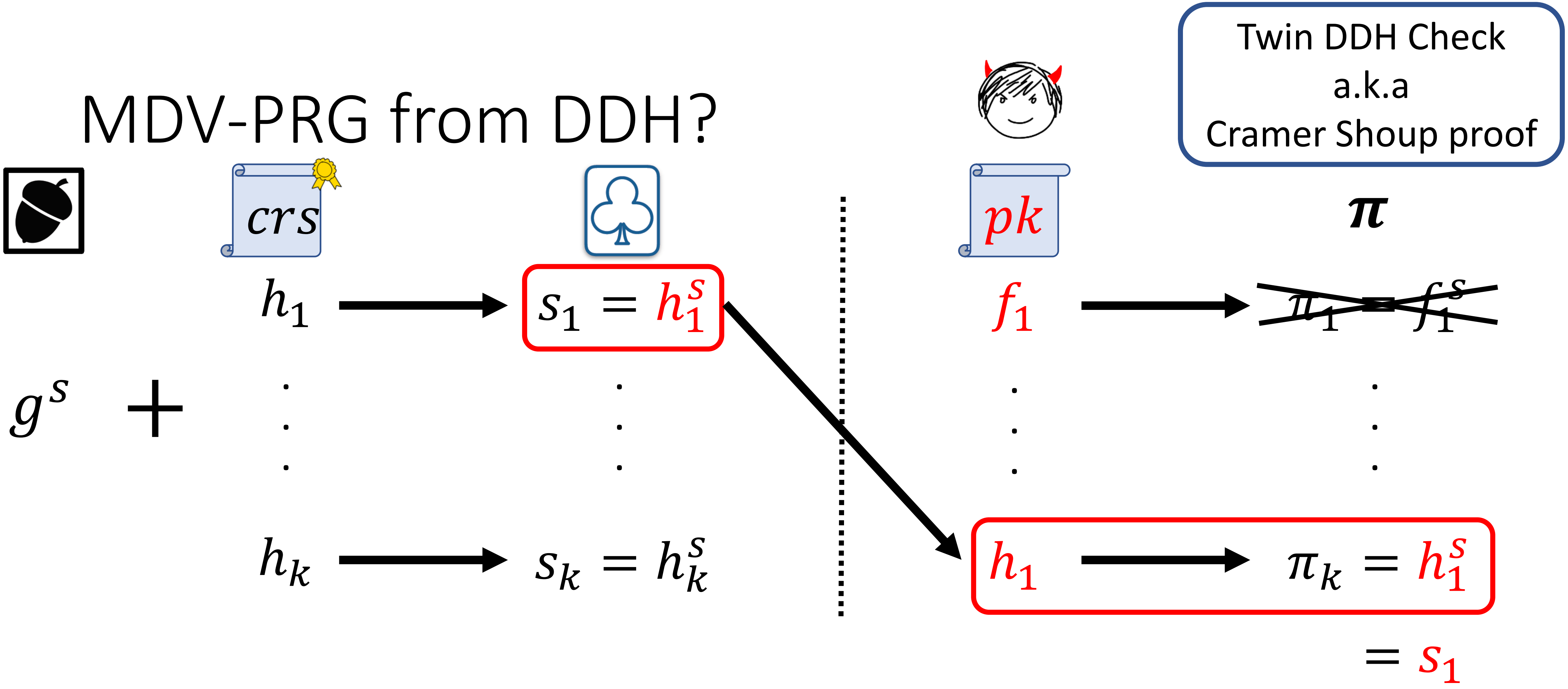
MDV-PRG from DDH?

Twin DDH Check
a.k.a
Cramer Shoup proof



- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$
  - Malicious Verifier can learn other bits!

MDV-PRG from DDH?



- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$
  - Add **random dependencies**?

Adding dependencies

Twin DDH Check
a.k.a
Cramer Shoup proof

crs | pk | $\boldsymbol{\pi}$

g^s $+$

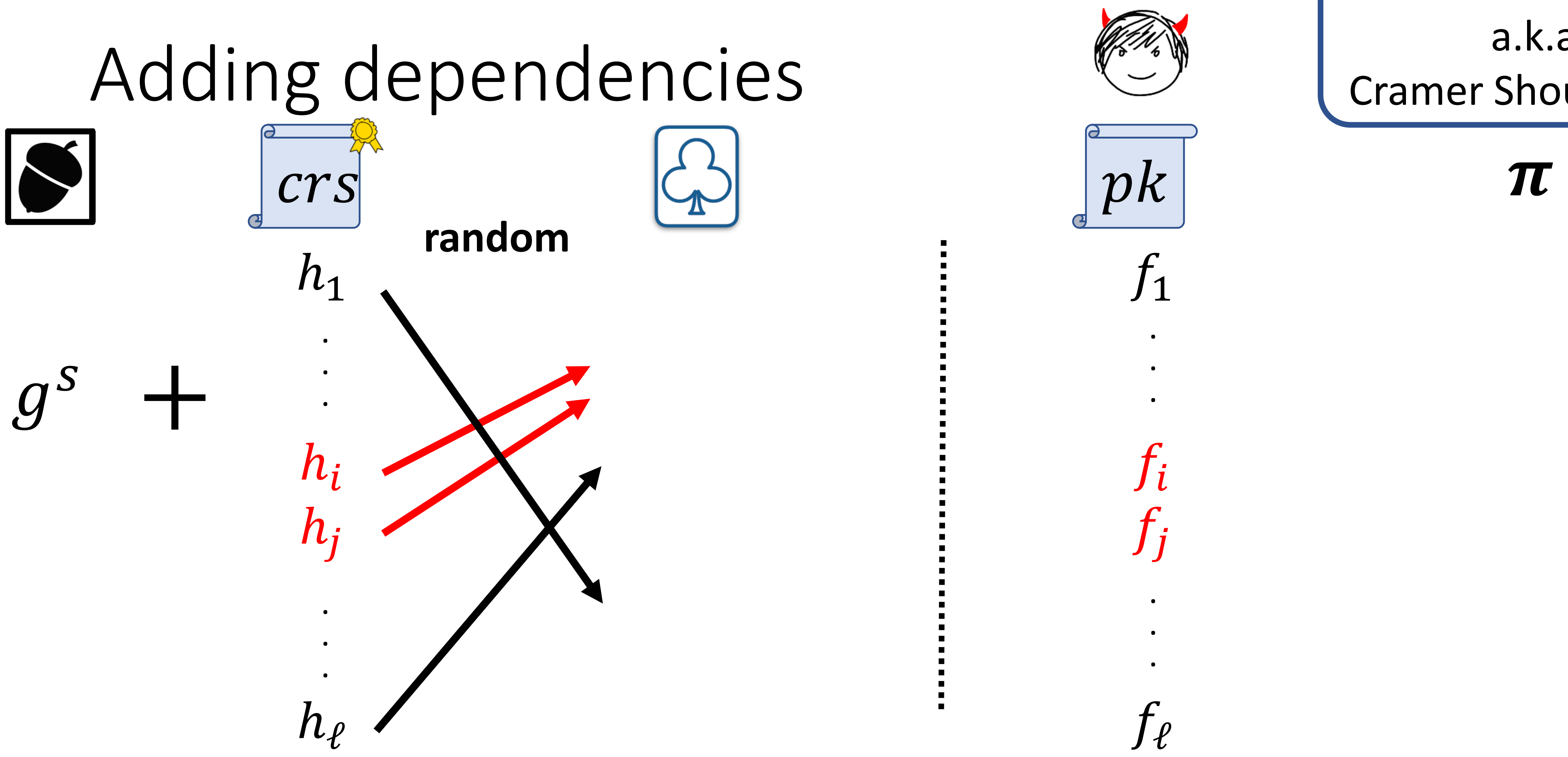
h_1 . . . h_i h_j . . . h_ℓ

f_1 . . . f_i f_j . . . f_ℓ

- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

Adding dependencies

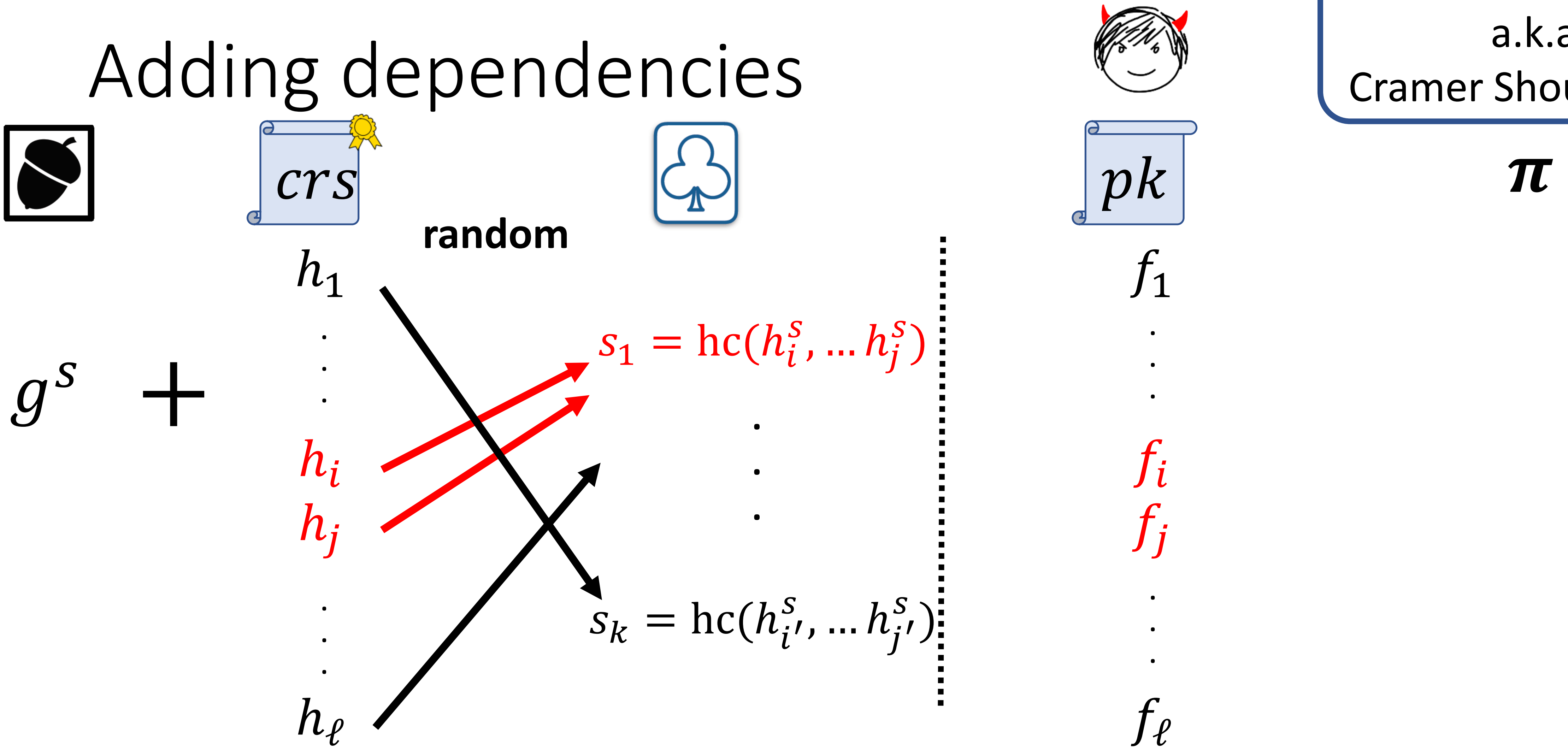
Twin DDH Check
a.k.a
Cramer Shoup proof



- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

Adding dependencies

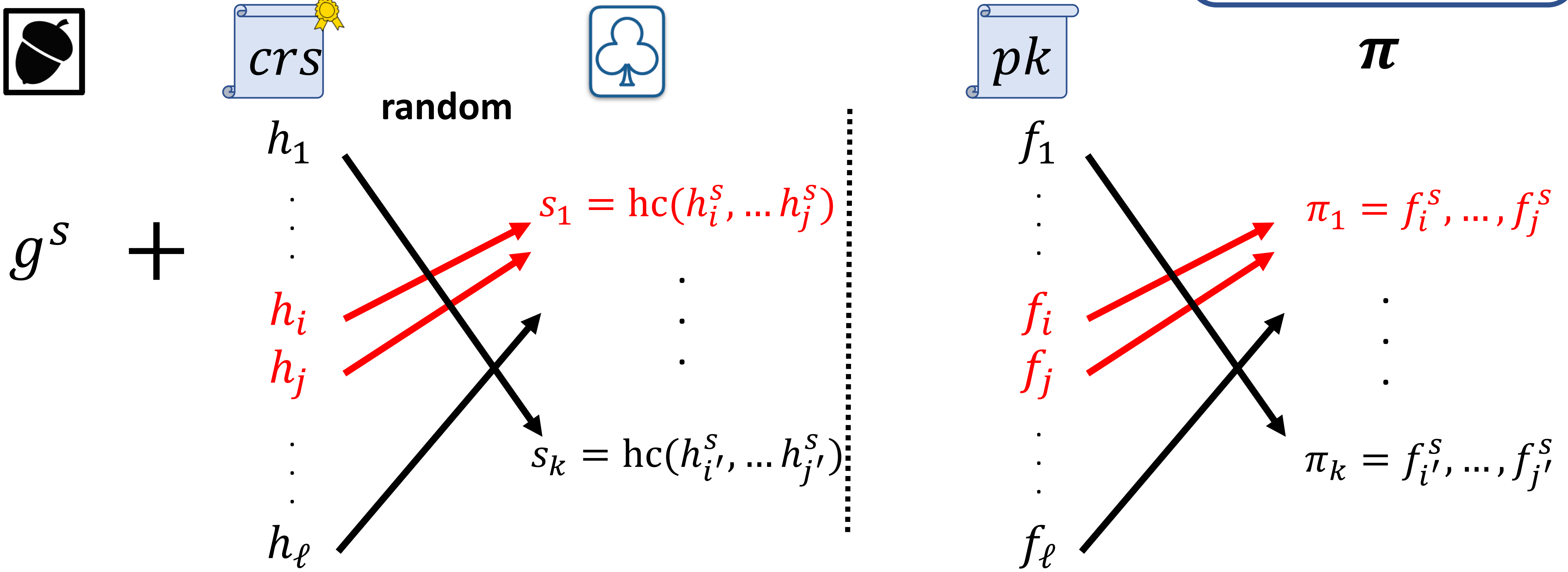
Twin DDH Check
a.k.a
Cramer Shoup proof



- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

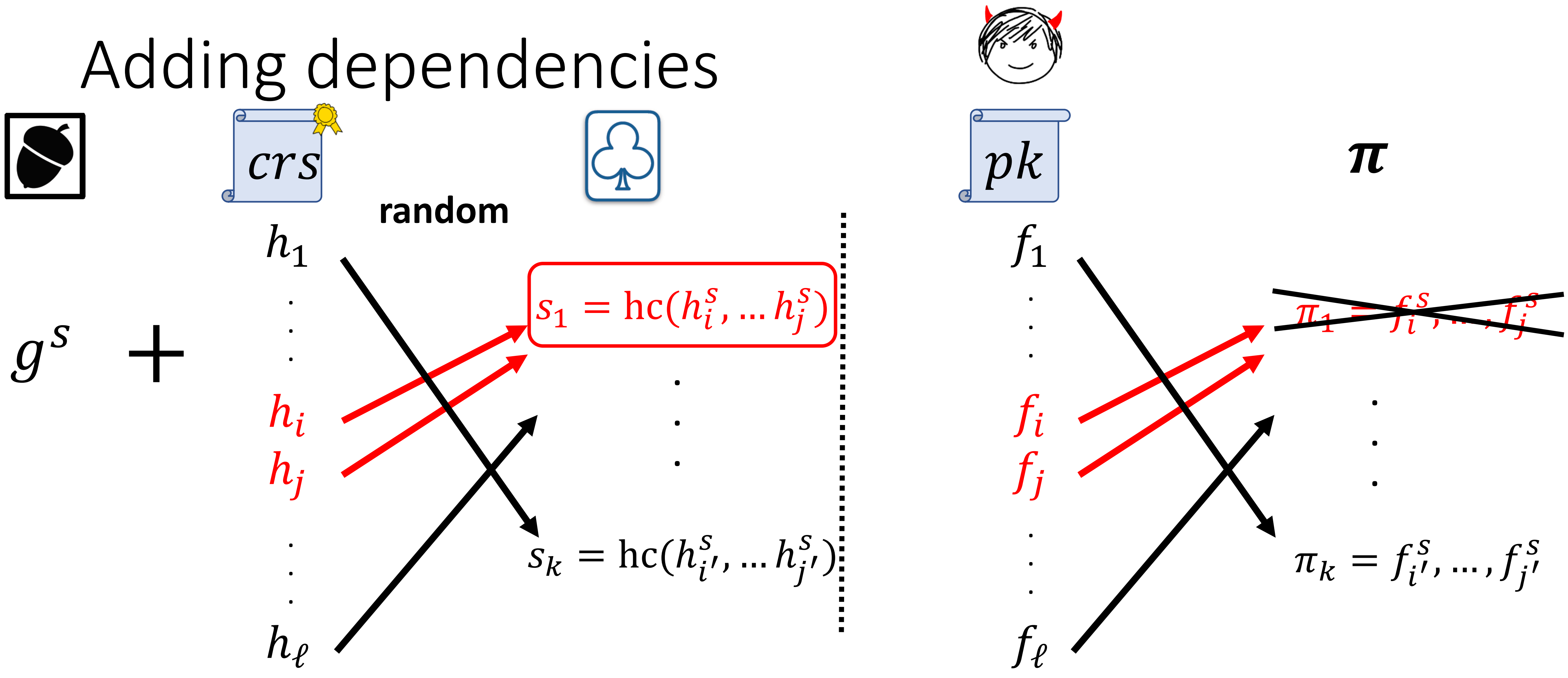
Adding dependencies

Twin DDH Check
a.k.a
Cramer Shoup proof

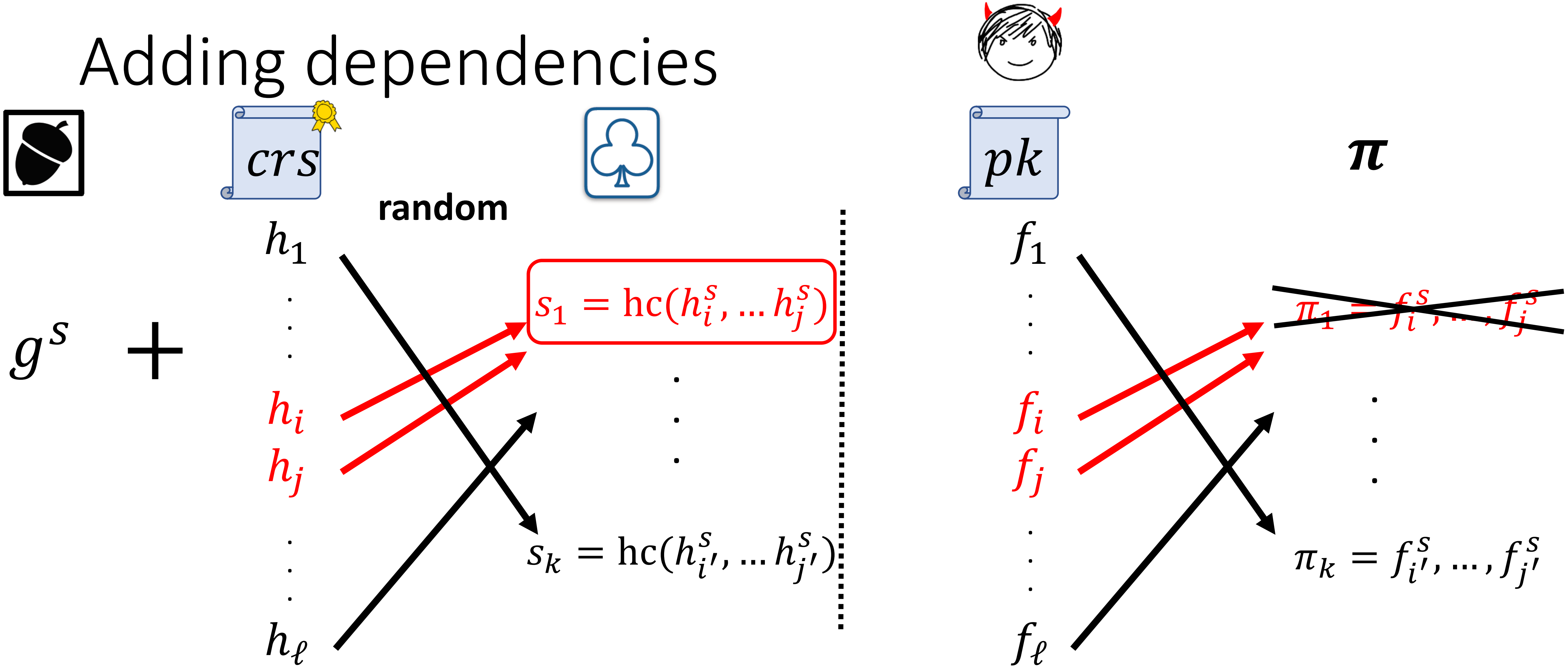


- **Malicious Hiding:** even against **adversarial** pk, proof $\boldsymbol{\pi}_i$ hides $\boldsymbol{s}_j$ for $i \neq j$

Adding dependencies



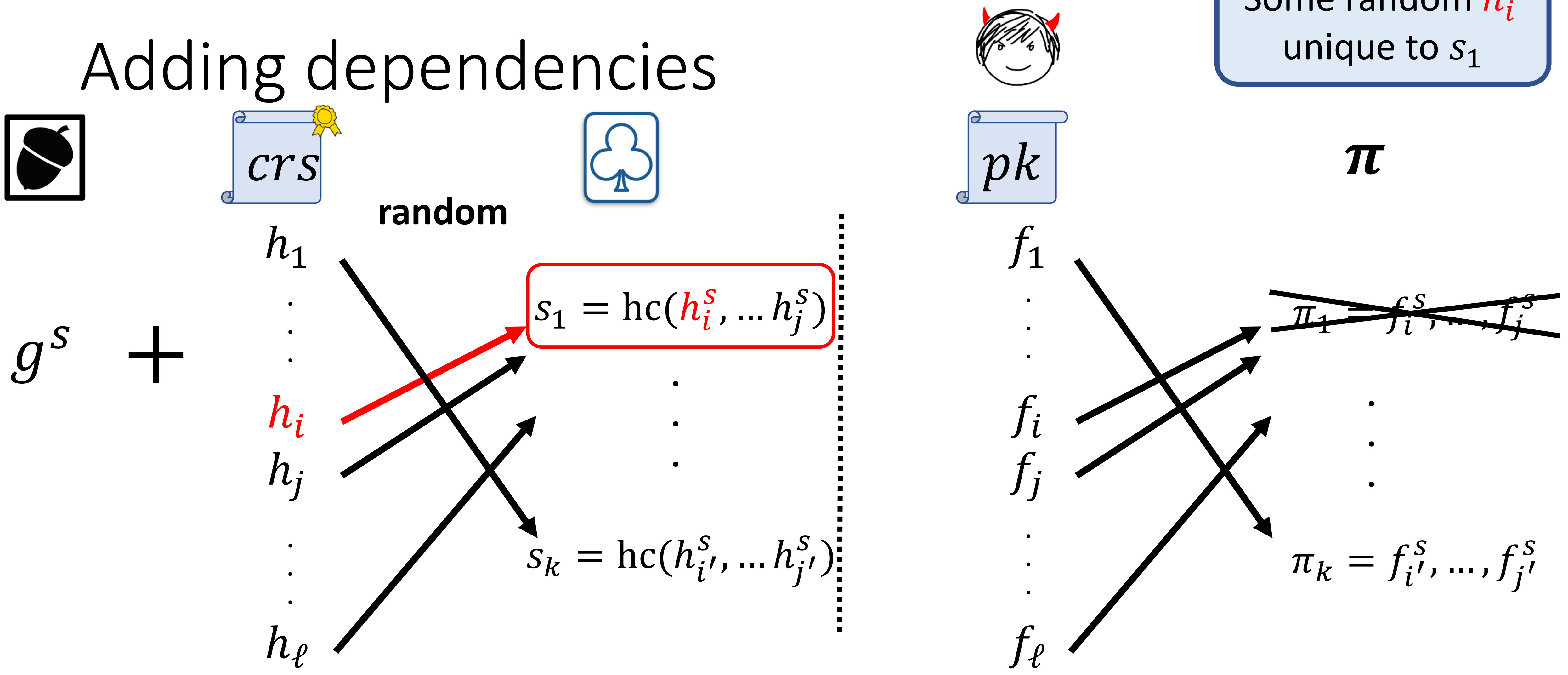
Adding dependencies



- needs ***all*** the elements $h_i^s, \ldots h_j^s$ to recover s_1

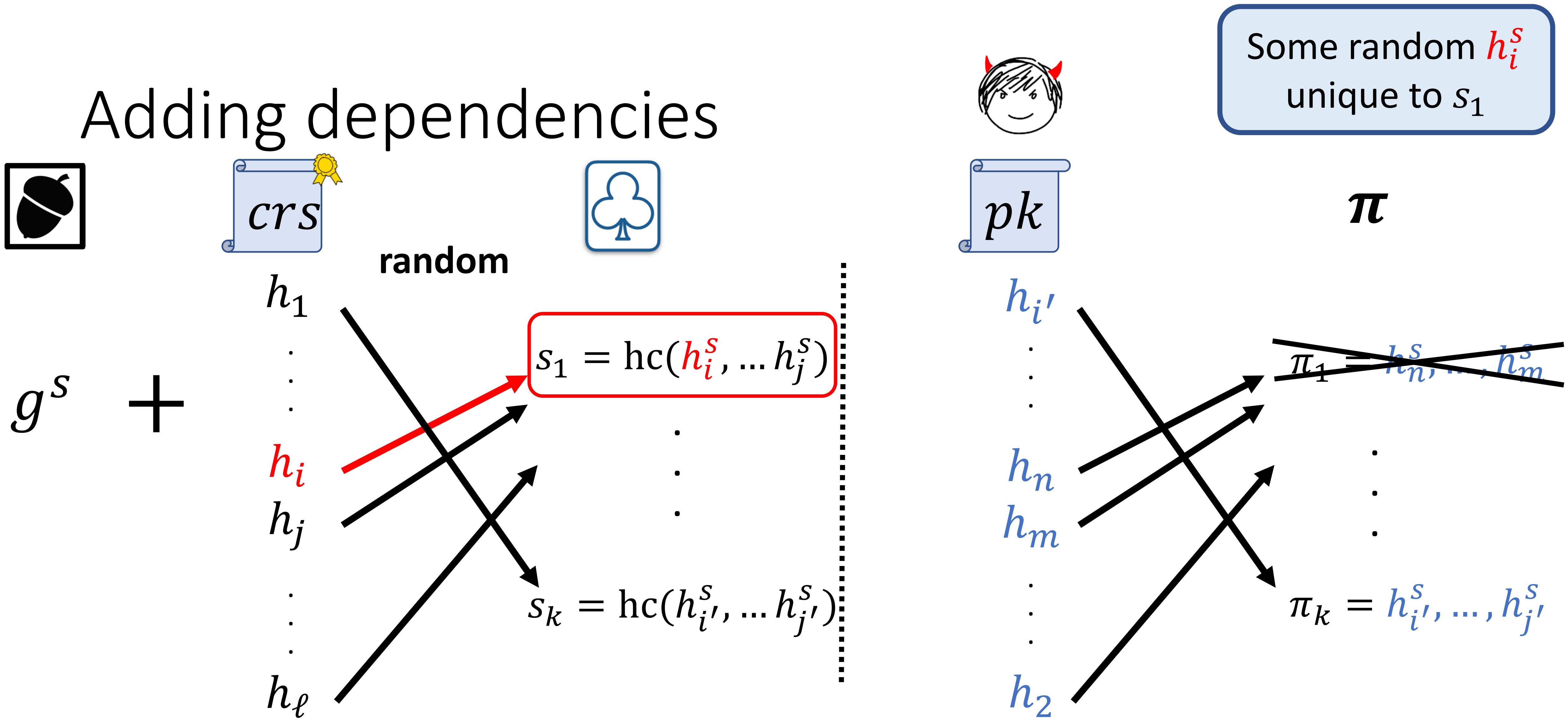
Adding dependencies

Some random h_i^s unique to s_1



- needs ***all*** the elements $h_i^s, \ldots h_j^s$ to recover s_1

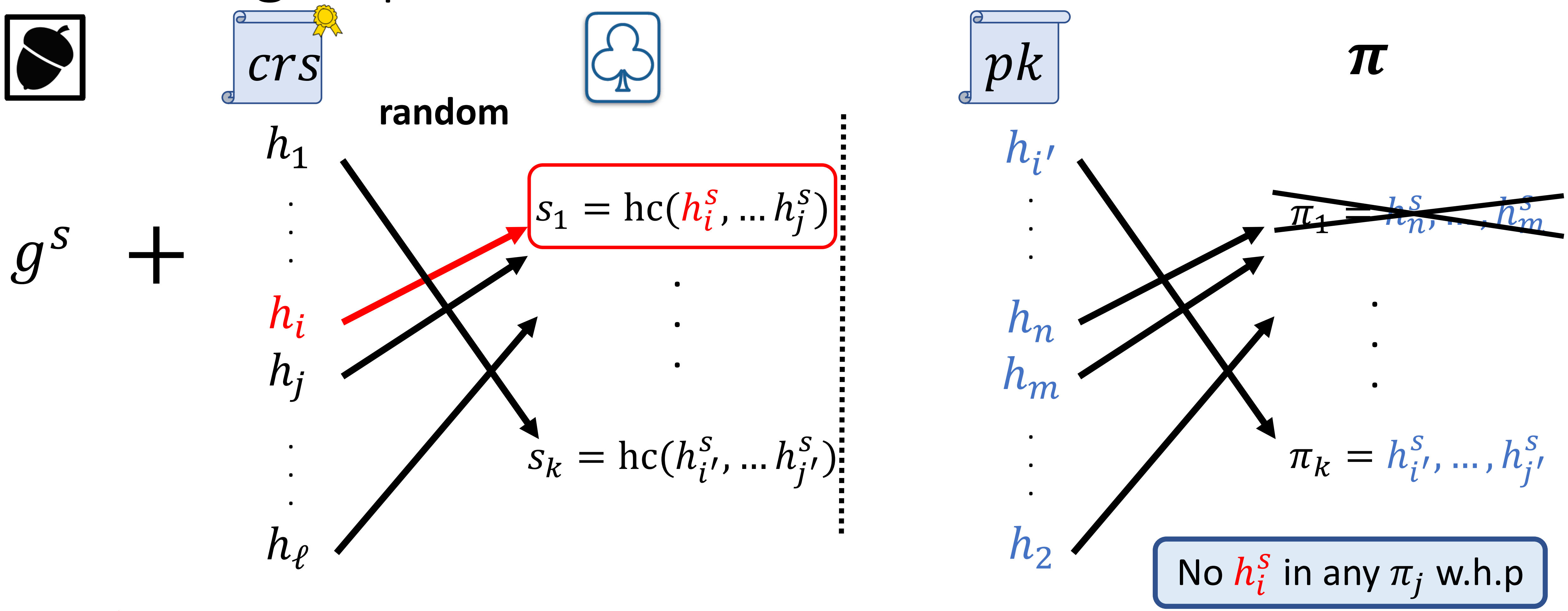
Adding dependencies



- needs ***all*** the elements $h_i^s, \dots h_j^s$ to recover s_1

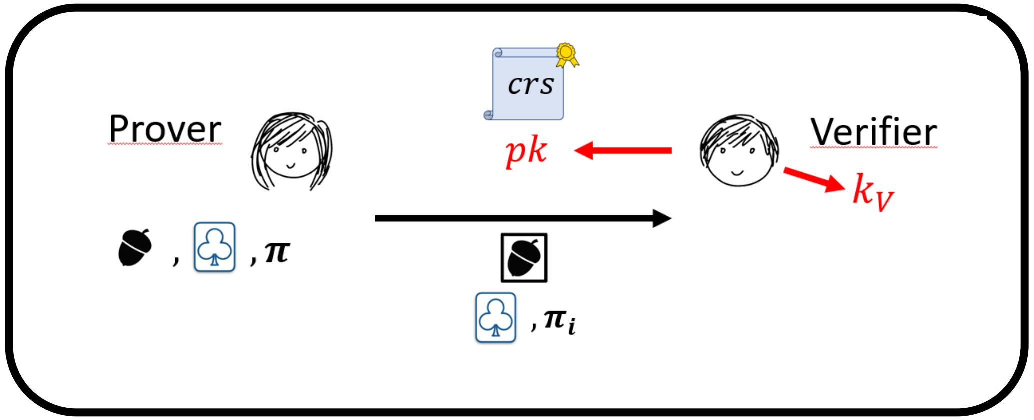
Adding dependencies

Some random h_i^s unique to s_1

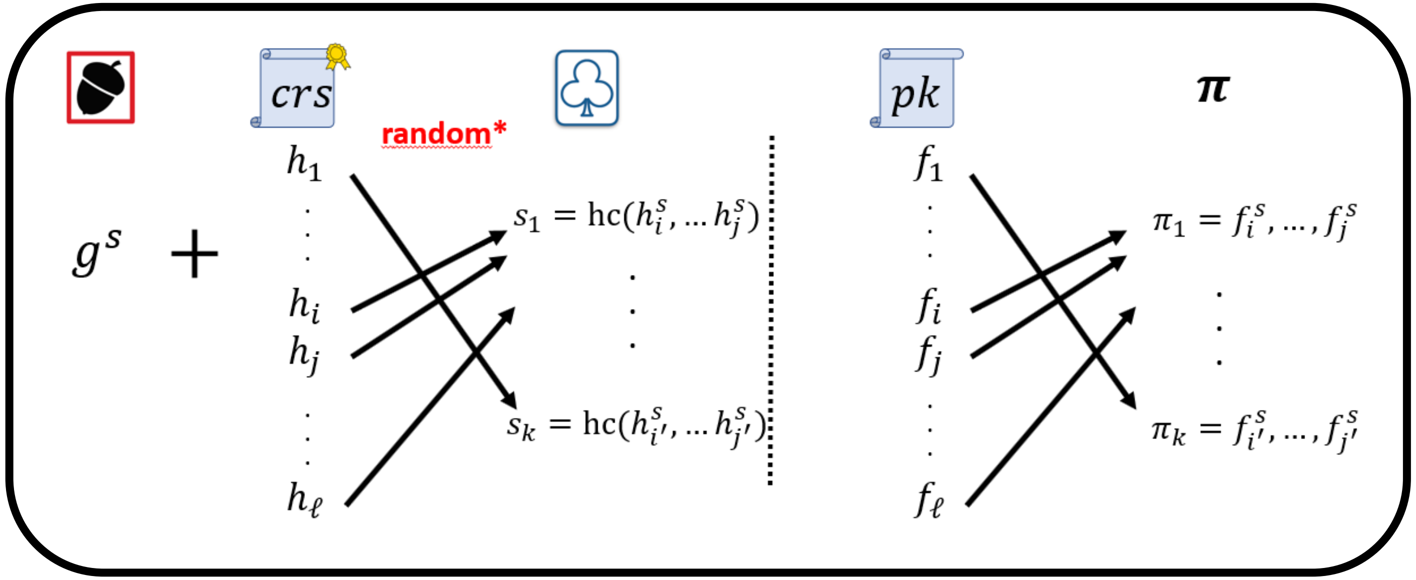


- needs ***all*** the elements $h_i^s, \ldots h_j^s$ to recover s_1

Our result

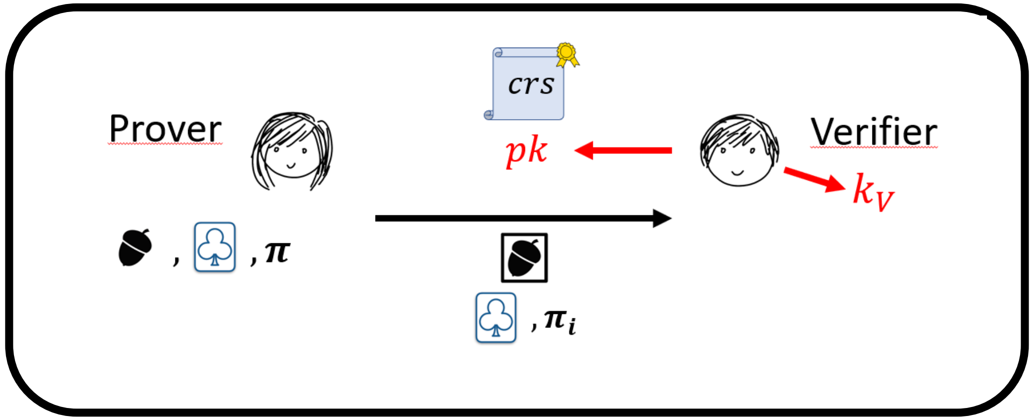


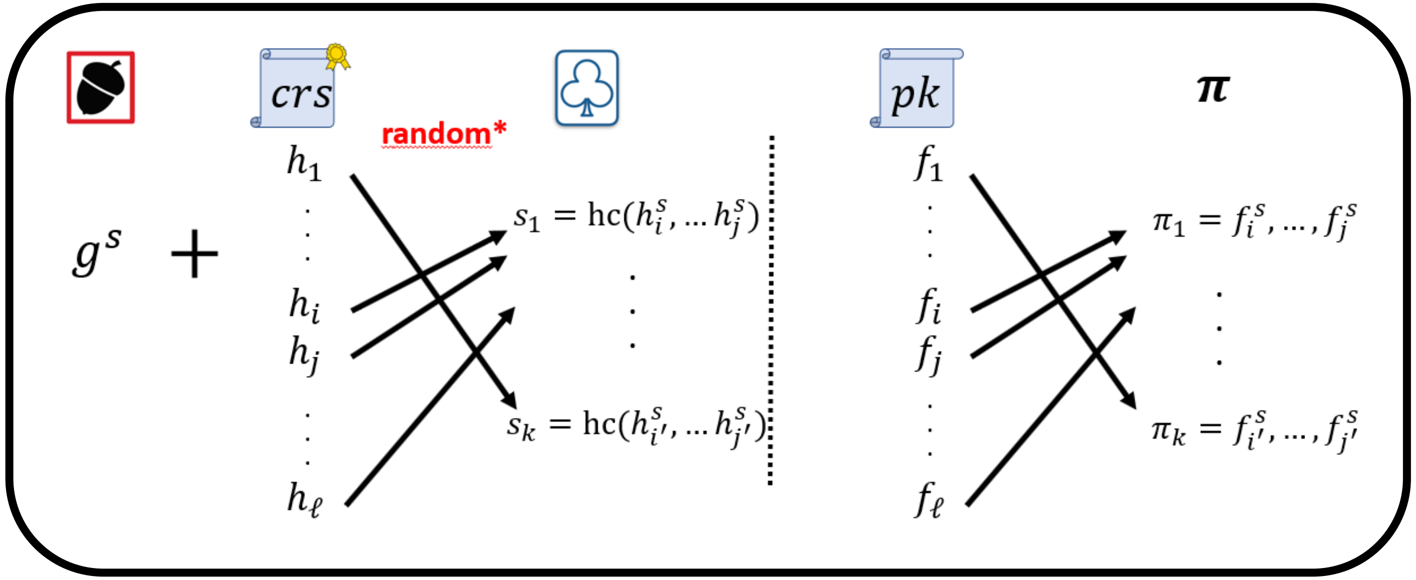
Prover
crs
pk
Verifier
k_V
, , π
, π_i



crs
pk
π
random*
g^s +
h_1
h_i
h_j
h_ℓ
$s_1 = \mathrm{hc}(h_i^s, \dots h_j^s)$
$s_k = \mathrm{hc}(h_{i'}^s, \dots h_{j'}^s)$
f_1
f_i
f_j
f_ℓ
$\pi_1 = f_i^s, \dots, f_j^s$
$\pi_k = f_{i'}^s, \dots, f_{j'}^s$

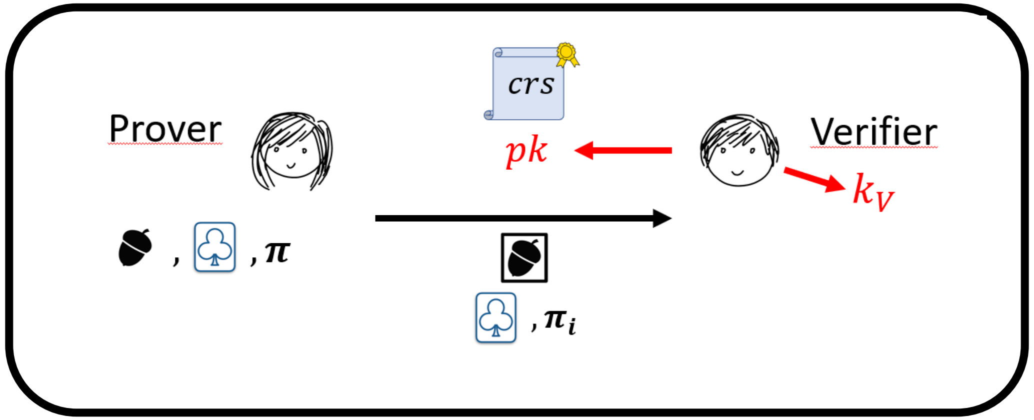
Our result

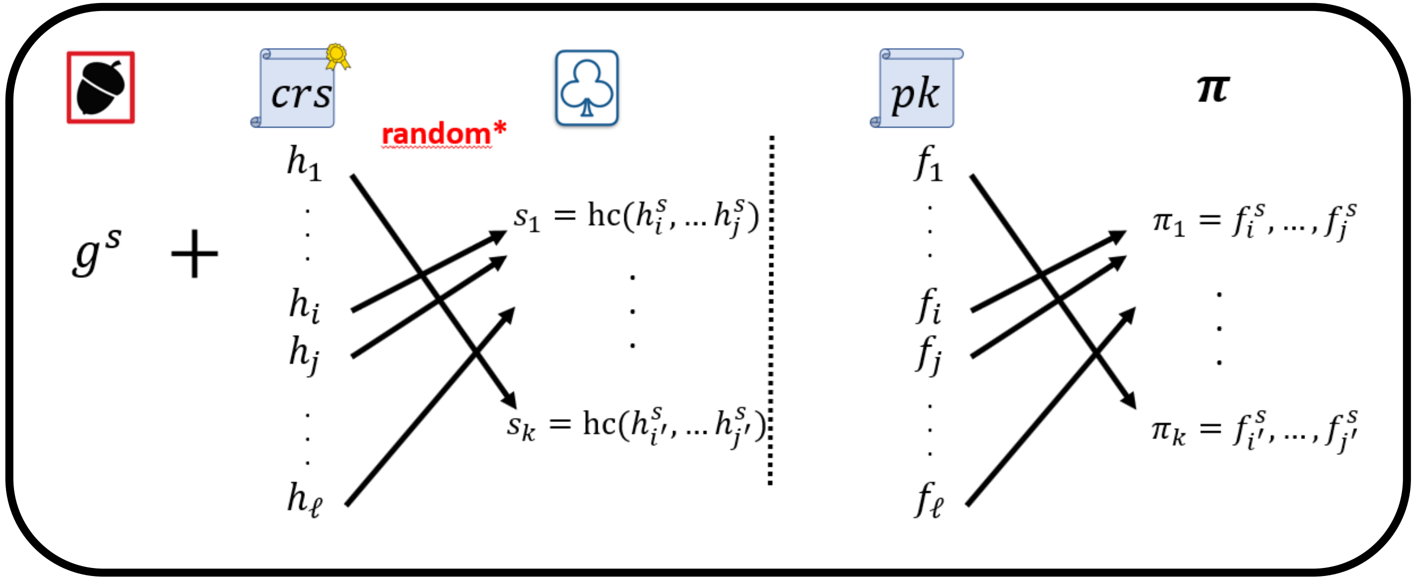




Theorem: MDV-PRG under ***One-More CDH***

Our result





Theorem: MDV-PRG under One-More CDH

Corollary: MDV-NIZK from One-More CDH

@EUROCRYPT'19

Part3: Designated Verifier/Prover Preprocessing NIZKs from Diffie-Hellman Assumptions

Shuichi Katsumata (AIST), Ryo Nishimaki (NTT),
Shota Yamada (AIST), Takashi Yamakawa (NTT).

Our Result

1. **DV**-NIZK from the **CDH** assumption (with "long" proof size).

2. **DP**-NIZK from **non-static DH-type** assumption over **pairing groups** with "short" proof size.

3. **PP**-NIZK from the **DDH** assumption with "short" proof size.

Our Result

1. ~~**DV**-NIZK from the **CDH** assumption (with "long" proof size).~~

DONE

2. **DP**-NIZK from **non-static DH-type** assumption over **pairing groups** with "short" proof size.

3. **PP**-NIZK from the **DDH** assumption with "short" proof size.

This Talk

Motivation

NIZK with $|\pi|$ independent of circuit C computing the NP relation is only known from strong assumptions:

(*)iO, FHE, knowledge assumptions, compact HomSig.

Motivation

NIZK with $|\pi|$ independent of circuit C computing the NP relation is only known from strong assumptions:

(*)iO, FHE, knowledge assumptions, compact HomSig.

Without (*):

- DV-NIZK from CDH has proof size $\mathrm{poly}(\lambda, |C|)$.
- Famous GOS CRS-NIZK has proof size $O(\lambda|C|)$.
- <u>Shortest know</u> is CRS-NIZK of [Gro10@AC] based on Naccache-Stern PKE has proof size $\mathrm{polylog}(\lambda)|C|$.

Motivation

NIZK with $|\pi|$ independent of circuit C computing the NP relation is only known from strong assumptions:

(*)iO, FHE, knowledge assumptions, compact HomSig.

Without (*):

- DV-NIZK from CDH has proof size $\mathrm{poly}(\lambda, |\mathrm{C}|)$.
- Famous GOS CRS-NIZK has proof size $\mathrm{O}(\lambda|\mathrm{C}|)$.
- Shortest know is CRS-NIZK of [Gro10@AC] based on Naccache-Stern PKE has proof size $\mathrm{polylog}(\lambda)|\mathrm{C}|$.

→ ***Multiplicative overhead in*** $|\mathbf{C}|$***…***

Motivation

NIZK with
NP relatio

(*)iO, FH

This Work

(DP, PP)-NIZKs based on **falsifiable pairing/paring-free group assumptions** with proof size $|\mathrm{C}| + \mathbf{poly}(\lambda)$.

Without (

- DV-NIZK from CDH has proof size $\mathrm{poly}(\lambda, |\mathrm{C}|)$.
- Famous GOS CRS-NIZK has proof size $O(\lambda|\mathrm{C}|)$.
- <u>Shortest know</u> is CRS-NIZK of [Gro10@AC] based on Naccache-Stern PKE has proof size $\mathrm{polylog}(\lambda)|\mathrm{C}|$.

➡ ***Multiplicative overhead in*** $|\mathrm{C}|$***…***

Recap: (DP, PP)-NIZKs

Designated-Prover NIZKs

Prover (x, w)

π

Verifier x

Proving Key k_P

*Opposite to DV-NIZKs

Recap: (DP, PP)-NIZKs

PreProcessing NIZKs

Prover (x, w)

π →

Verifier x

Proving Key $\mathrm{k_P}$

Verifying Key $\mathrm{k_V}$

*Relaxation of DP and DV-NIZKs

Recap: (DP, PP)-NIZKs

PreProcessing NIZKs

Prover (x, w)

π →

Verifier x

Proving Key $\mathrm{k_P}$

Verifying Key $\mathrm{k_V}$

- Result of [KimWu18@Crypto]

Any **context-hiding homomorphic signatures/MACs** (HomSig/MAC) can be converted into **DP/PP-NIZKs**.

HomSig/MAC in a Nutshell

Signer

Signs on many messages

$\mathbf{w} = (w_1, \ldots, w_k)$

➡ $\{(w_i, \sigma_i)\}_{i \in [k]}$

Cirucit C

$\{(w_i, \sigma_i)\}_{i \in [k]}$

(Public) Evaluator

"Evaluated" Signature on message $C(\mathbf{w})$

$(C(\mathbf{w}), \sigma_C)$

HomSig/MAC in a Nutshell

Signer

Signs on many messages

$\mathbf{w} = (w_1, \dots, w_k)$

➡ $\{(w_i, \sigma_i)\}_{i\in[k]}$

Cirucit C

$\{(w_i, \sigma_i)\}_{i\in[k]}$

(Public) Evaluator

"Evaluated" Signature on message $C(\mathbf{w})$

$(C(\mathbf{w}), \sigma_C)$

- **Unforgeability** → For **soundness**.
- **Context-Hiding:** Evaluated signature $(C(\mathbf{w}), \sigma_C)$ leaks no information of the original message $\mathbf{w}$. → For **zero-knowledge**.

HomSig/MAC in a Nutshell

Signer

(Public) Evaluator

Cirucit C

$\{(w_i, \sigma_i)\}_{i\in[k]}$

"Evaluated" Signature on message $C(\mathbf{w})$

$(C(\mathbf{w}), \sigma_C)$

Signs on many messages

$\mathbf{w} = (w_1, \dots, w_k)$

➡ $\{(w_i, \sigma_i)\}_{i\in[k]}$

If $|\sigma_C|$=poly(λ) for $\forall C \in \mathbf{NC^1}$, then $|\pi| = |C|$ +poly(λ) by [KimWu18].

- **Unforgeability**
- **Context-Hiding:** Evaluated signature $(C(\mathbf{w}), \sigma_C)$ leaks no information of the original message $\mathbf{w}$.

→ For **zero-knowledge**.

Result 1: New HomSig (=>DP-NIZK)

Compact HomSig for $\mathbf{NC}^1$ based on a **non-static Diffie-Hellman** type assumption.

Core Idea:

- View the simulator used in certain Key-Policy ABE security proofs as HomSigs.
- Construct Key-Policy ABE with constant-sized secret-keys from non-static DH type assumptions building on [RW13, AC16, AC17].

High Level Overview of Result 1

Proof of ***selective security*** of an ABE scheme…

ABE Simulator S
(= Adversary for some hard problem )

x^* ← Target attribute

Adversary 

High Level Overview of Result 1

Proof of *selective security* of an ABE scheme…

ABE Simulator S

(= Adversary for some hard problem 

Adversary 

$\leftarrow x^*$ — Target attribute

Generate **sim. trapdoor td_{x^*}** along with pp.

$\xrightarrow{pp}$

High Level Overview of Result 1

Proof of *selective security* of an ABE scheme…

ABE Simulator S

(= Adversary for some hard problem 

x^*

Target attribute

Adversary 

Generate **sim. trapdoor td_{x^*}** along with pp.

pp

C s.t. $C(x^*) = 0$

Secret key query

Use td_{x^*} to **simulate secret key sk_C**

sk_C

High Level Overview of Result 1

Proof of ***selective security*** of an ABE scheme…

ABE Simulator S

(= Adversary for some hard problem )

Adversary 

x^* ←

Target attribute

View $\mathrm{td}_{\mathrm{x}^*}$ as a "signature" for msg x^*.

Generate **sim. trapdoor** $\mathbf{td}_{\mathbf{x}^*}$ along with pp.

pp →

C s.t. $\mathrm{C}(\mathrm{x}^*) = 0$ ←

Secret key query

Use $\mathrm{td}_{\mathrm{x}^*}$ to **simulate secret key** $\mathbf{sk}_{\mathbf{C}}$

sk_{C} →

High Level Overview of Result 1

Proof of *selective security* of an ABE scheme…

ABE Simulator S

(= Adversary for some hard problem 

Adversary 

x^* ← Target attribute

Generate **sim. trapdoor td_{x^*}** along with pp.

pp →

View td_{x^*} as a "signature" for msg x^*.

C s.t. $C(x^*)$ ←

Use td_{x^*} to **simulate secret key sk_C**

sk_C →

View this process as "evaluating" td_{x^*} on circuit C. Then, sk_C is the "evaluated signature" for message $C(x^*)=0$.

Result 2: New HomMAC (=>PP-NIZK)

Compact HomMAC for **arithmetic circuits of poly. bounded degree** based on **DDH**.

*Includes $\mathbf{NC^1}$!!

Core Idea:

- Transform the **non-context-hiding** HomMAC by [CatFio18@JoC] into a **context-hiding** HomMAC using (extractable) **FE for inner prodoucts (IPFE)**.

- Instantiate with **DDH-based** (extractable) **IPFE** by [AgrLibSte16@Crypto]

* Since we need the "extractable" feature, the LWE-based IPFE of [AgrLibSte16] cannot be used.

High Level Overview of Result 2

Non-context-hiding HomMAC by [CatFio18]

- KeyGen(): sk = $(s, \boldsymbol{r}) \leftarrow \mathbb{Z}_p^{k+1}$
- Sign(sk, $w_i \in \mathbb{Z}_p$): σ_i such that $r_i = w_i + \sigma_i s$

High Level Overview of Result 2

Non-context-hiding HomMAC by [CatFio18]

- KeyGen(): sk = $(s, \boldsymbol{r}) \leftarrow \mathbb{Z}_p^{k+1}$
- Sign(sk, $w_i \in \mathbb{Z}_p$): σ_i such that $r_i = w_i + \sigma_i s$
- SigEval(poly. f s.t. deg(f)= D, $\{(w_i, \sigma_i)\}_{i \in [k]}$):
  $\sigma_f = (c_1, \ldots, c_D) \in \mathbb{Z}_p^{D+1}$ s.t. $f(\boldsymbol{r}) = f(\boldsymbol{w}) + \sum_{j=1}^{D} c_j s^j$

*Can be computed w/o knowledge of $s, \boldsymbol{r}$!!

High Level Overview of Result 2

Non-context-hiding HomMAC by [CatFio18]

- KeyGen(): sk = $(s, \boldsymbol{r}) \leftarrow \mathbb{Z}_p^{k+1}$
- Sign(sk, $w_i \in \mathbb{Z}_p$): σ_i such that $r_i = w_i + \sigma_i s$
- SigEval(poly. f s.t. deg(f)= D, $\{(w_i, \sigma_i)\}_{i \in [k]}$):

  $\sigma_f = (c_1, \dots, c_D) \in \mathbb{Z}_p^{D+1}$ s.t. $f(\boldsymbol{r}) = f(\boldsymbol{w}) + \sum_{j=1}^{D} c_j s^j$

  *Can be computed w/o knowledge of $s, \boldsymbol{r}$!!
- VerifyEvaled(sk, f, (z, σ_f)):

  Compute $f(\boldsymbol{r})$ and check if $f(\boldsymbol{r}) = z + \sum_{j=1}^{D} c_j s^j$

High Level Overview of Result 2

Non-context-hiding HomMAC by [CatFio18]

- KeyGen(): sk = $(s, \boldsymbol{r}) \leftarrow \mathbb{Z}_p^{k+1}$
- Sign(sk, $w_i \in \mathbb{Z}_p$): σ_i such that $r_i = w_i + \sigma_i s$
- SigEval(poly. f s.t. deg(f)= D, $\{(w_i, \sigma_i)\}_{i \in [k]}$):

  $\sigma_f = (c_1, \ldots, c_D) \in \mathbb{Z}_p^{D+1}$ s.t. $f(\boldsymbol{r}) = f(\boldsymbol{w}) + \sum_{j=1}^{D} c_j s^j$

  *Can be computed w/o knowledge of $s, \boldsymbol{r}$!!
- VerifyEvaled(sk, f, (z, σ_f)):

  Compute $f(\boldsymbol{r})$ and check if $f(\boldsymbol{r}) = z + \sum_{j=1}^{D} c_j s^j$

Not context-hiding since $\sigma_f = (c_1, \ldots, c_D)$ may leak information of the original msg. $\boldsymbol{w}$!

High Level Overview of Result 2

Main Observation

- VerifyEvaled$(\mathrm{sk}, f, (z, \sigma_f))$:

  Compute $f(\boldsymbol{r})$ and check if $f(\boldsymbol{r}) = z + \sum_{j=1}^{D} c_j s^j$

Verification does **not** need to know $\sigma_f = (c_1, \ldots, c_D)$, but only the value of $\sum_{j=1}^{D} c_j s^j$!!

High Level Overview of Result 2

Main Observation

- VerifyEvaled$(\mathrm{sk}, f, (z, \sigma_f))$:
  Compute $f(\boldsymbol{r})$ and check if $f(\boldsymbol{r}) = z + \sum_{j=1}^{D} c_j s^j$

Verification does **not** need to know $\sigma_f = (c_1, \dots, c_D)$, but only the value of $\sum_{j=1}^{D} c_j s^j$!!

Use FE for inner products!

① Modify SigEval to output an encryption:

$$\mathrm{ct} \leftarrow \mathrm{IPFE.Enc}(\mathrm{mpk}, (c_1, \dots, c_D))$$

② Include $\mathrm{sk}_{\mathrm{IP}} \leftarrow \mathrm{IPFE.KeyGen}(\mathrm{msk}, (s, \dots, s^D))$ in secret key and change VerifyEvaled to check:

$$f(\boldsymbol{r}) \stackrel{?}{=} z + \mathrm{IPFE.Dec}(\mathrm{sk}_{\mathrm{IP}}, \mathrm{ct})$$

Questions??

Designated-Verifier Pseudorandom Generators, and their Applications

Geoffroy Couteau, Dennis Hofheinz

Karlsruher Institut für Technologie

Reusable Designated-Verifier NIZKs for all NP from CDH

Willy Quach, Ron D. Rothblum, and Daniel Wichs

Designated Verifier/Prover and Preprocessing NIZKs from Diffie-Hellman Assumptions

Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa