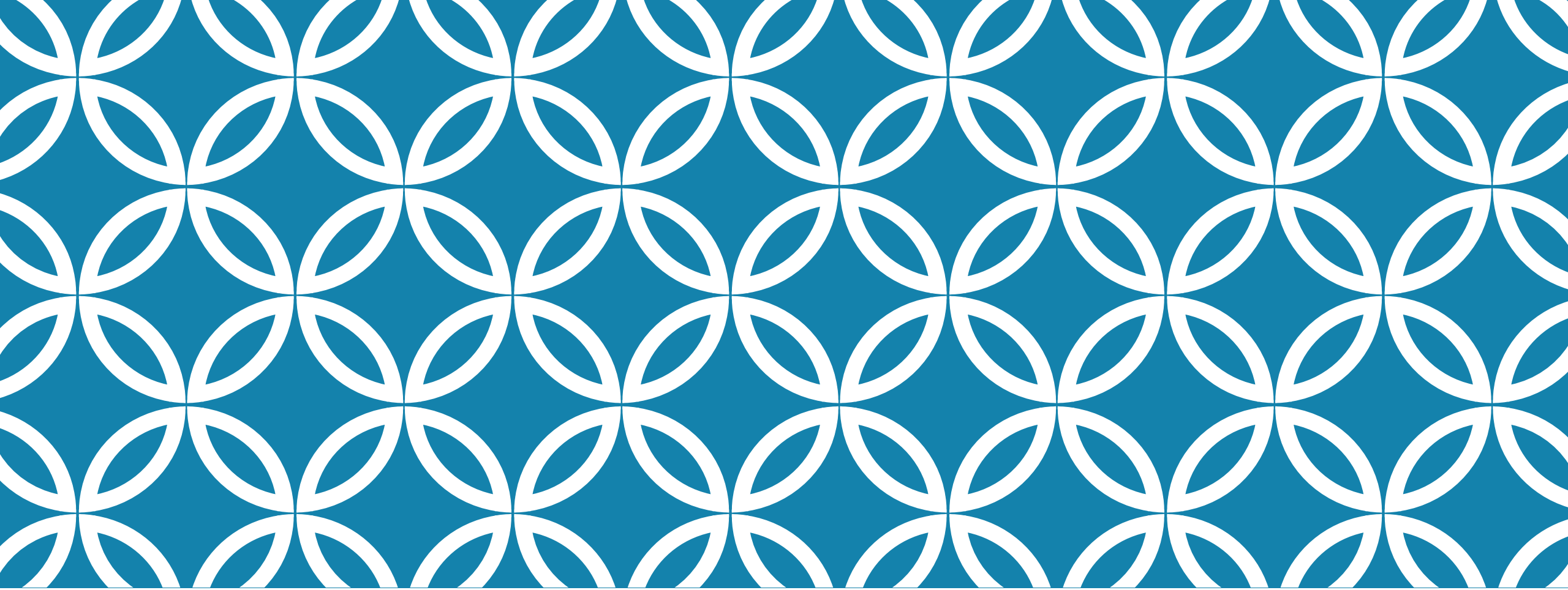


KEY-AND-ARGUMENT-UPDATABLE QA-NIZKS

Helger Lipmaa
Simula UiB, Bergen, Norway

REMINDER: NIZK

$inp = x,\ wit = w$

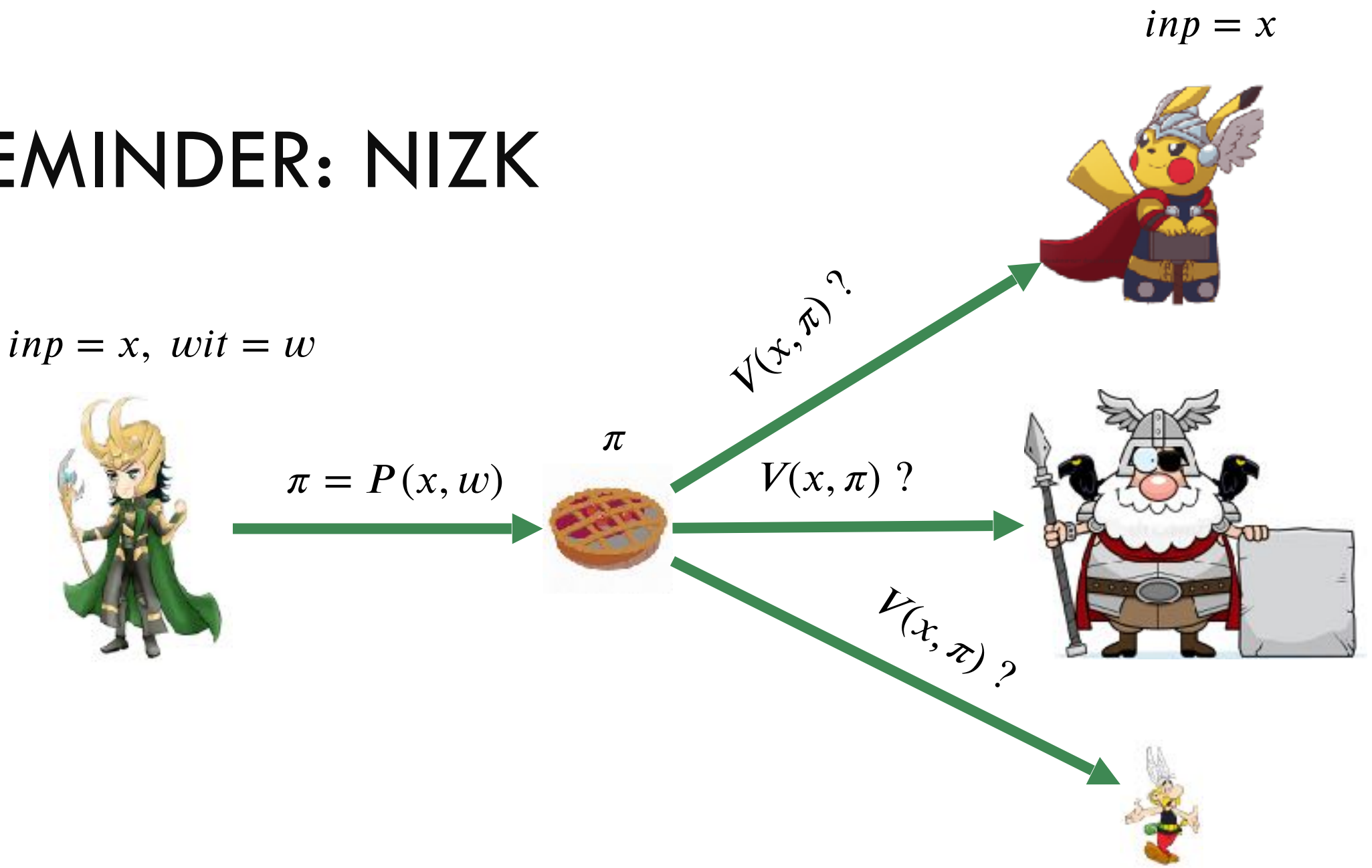
REMINDER: NIZK

$inp = x,\ wit = w$

$\pi = P(x, w)$

π

REMINDER: NIZK



REMINDER: NIZK

$inp = x$

$inp = x,\ wit = w$

$\pi = P(x, w)$

π

$V(x, \pi)$?

$V(x, \pi)$?

$V(x, \pi)$?

- Correctness
- Soundness
- Zero-knowledge

REMINDER: NIZK

$inp = x$

$inp = x,\ wit = w$

$\pi = P(x, w)$

π

$V(x, \pi)$?

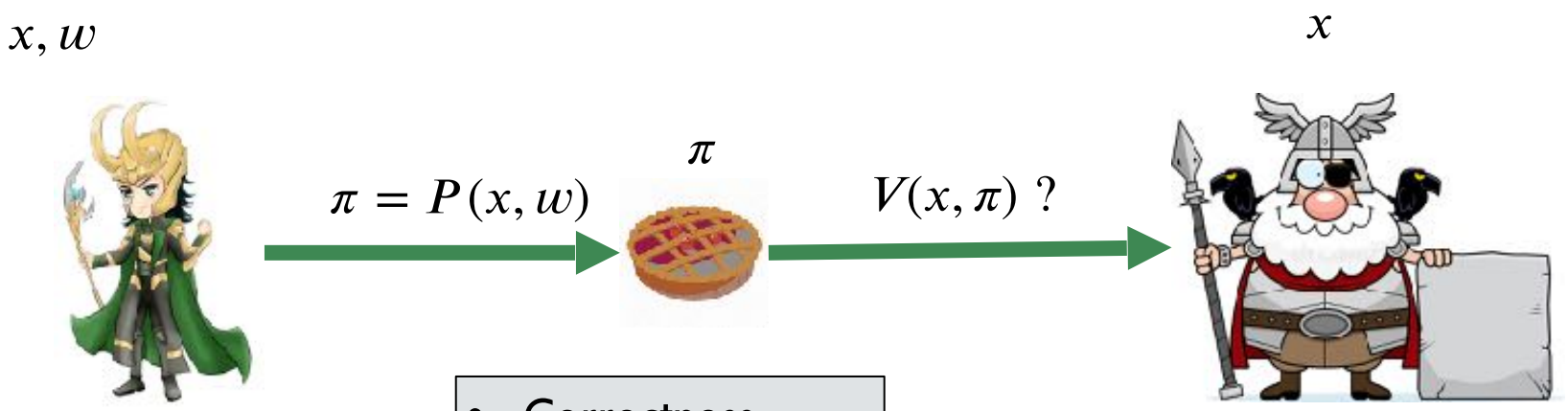
$V(x, \pi)$?

$V(x, \pi)$?

- Correctness
- Soundness
- Zero-knowledge

- Efficiency

REMINDER: SIMULATION



- Correctness
- Soundness
- **Zero-knowledge**

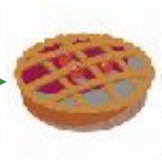
REMINDER: SIMULATION

x

x, w

$\pi = P(x, w)$

π

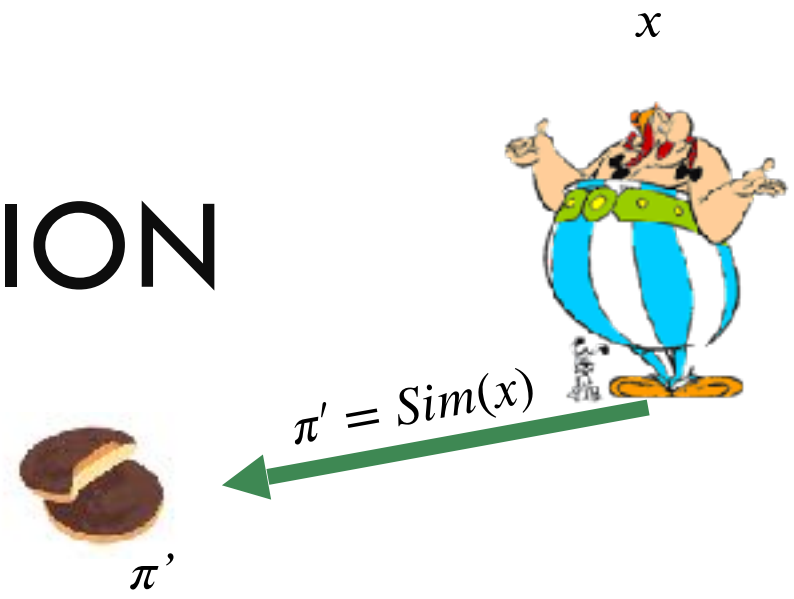


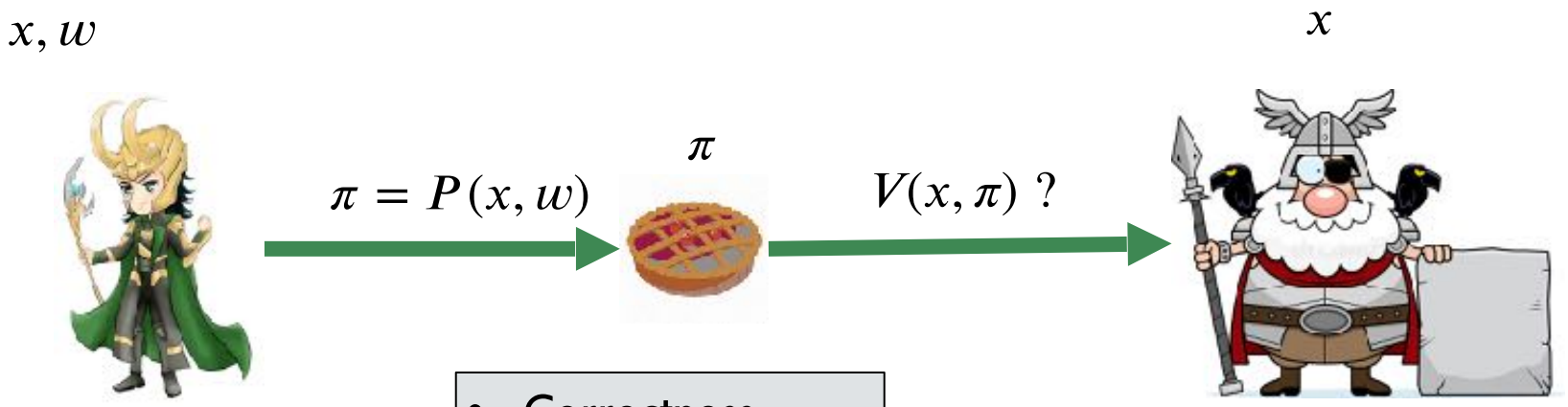
$V(x, \pi)$?

x

- Correctness
- Soundness
- **Zero-knowledge**

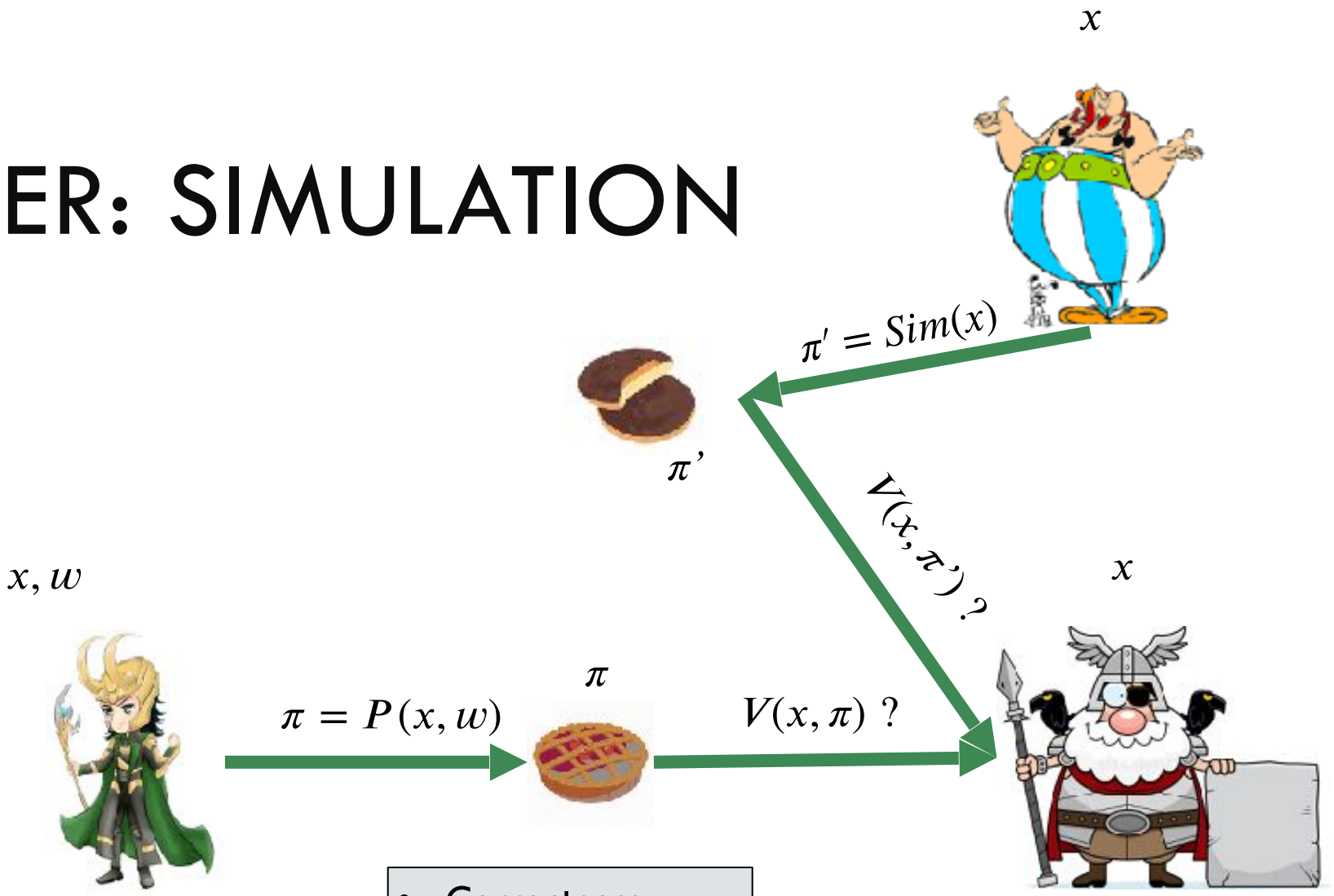
REMINDER: SIMULATION





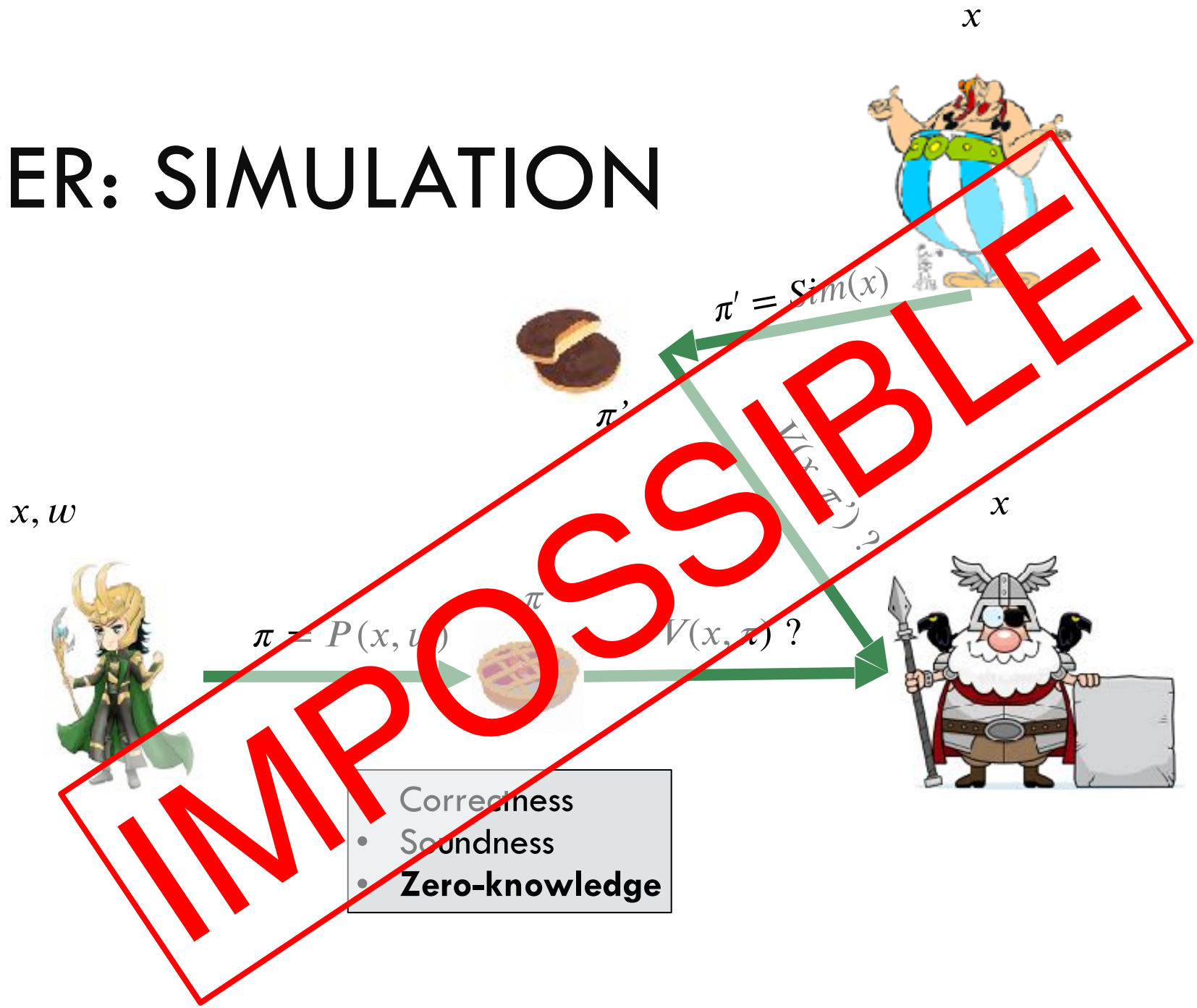
- Correctness
- Soundness
- **Zero-knowledge**

REMINDER: SIMULATION



- Correctness
- Soundness
- **Zero-knowledge**

REMINDER: SIMULATION



CRS MODEL

CRS MODEL

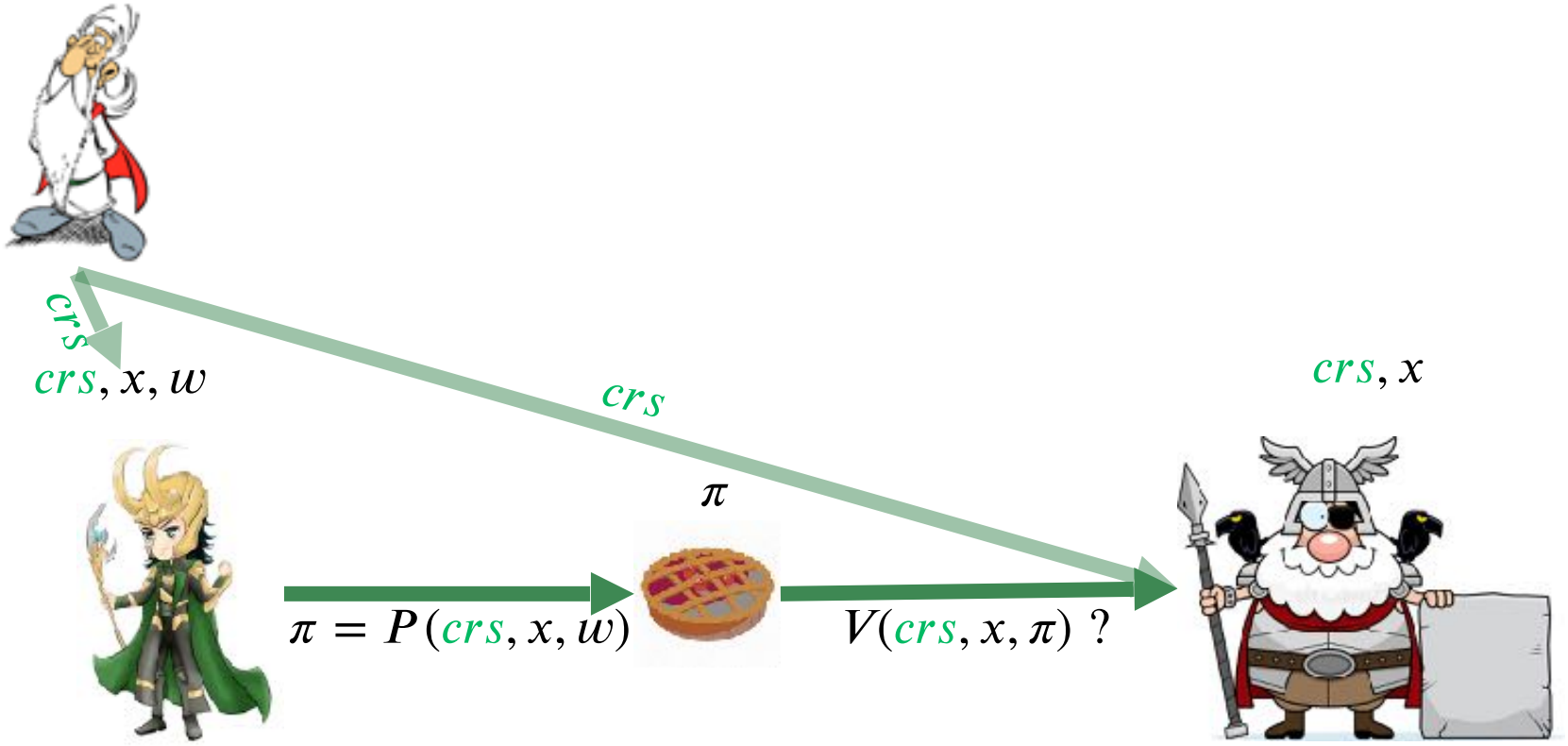
crs

crs, x, w

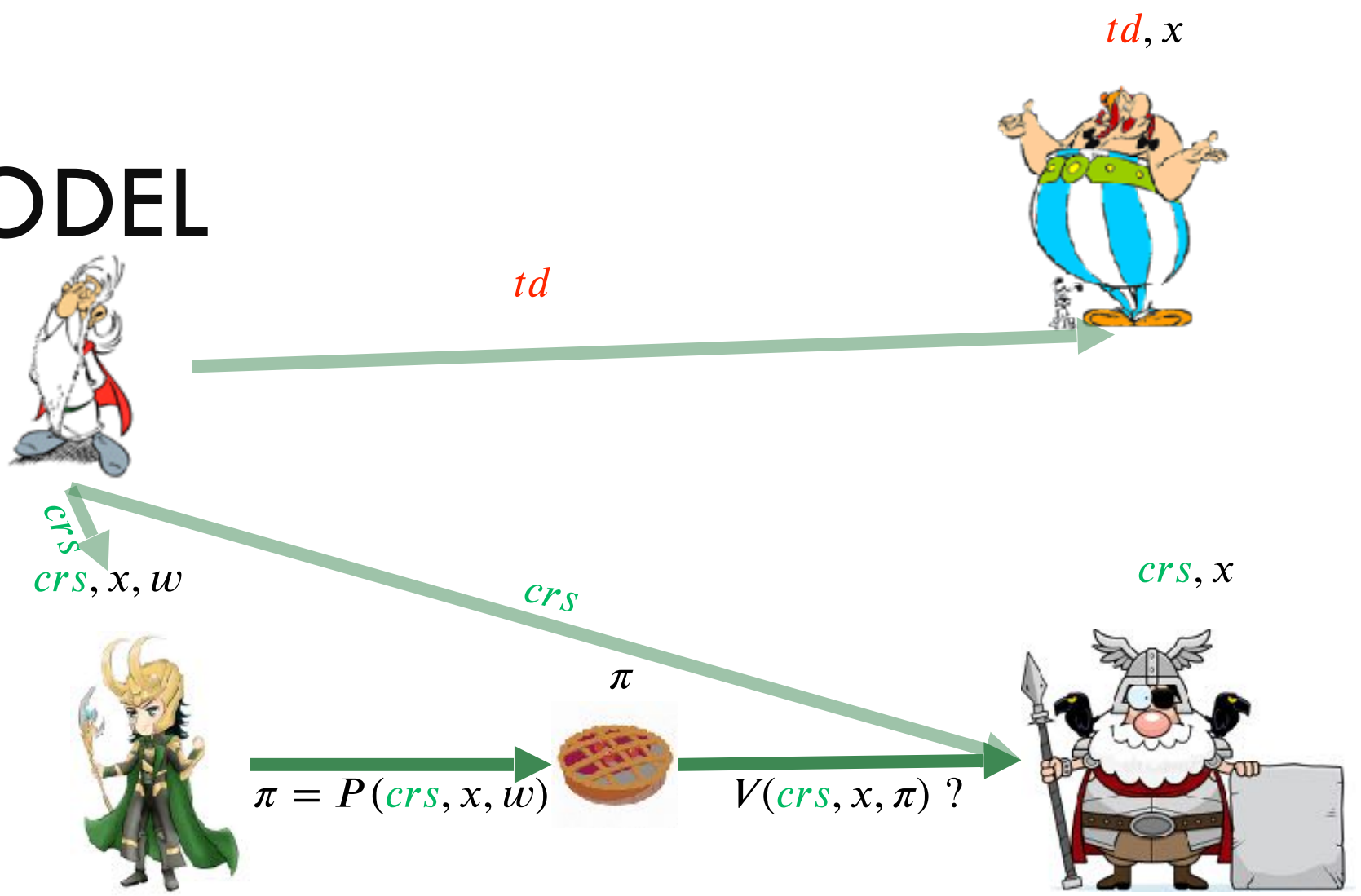
crs

crs, x

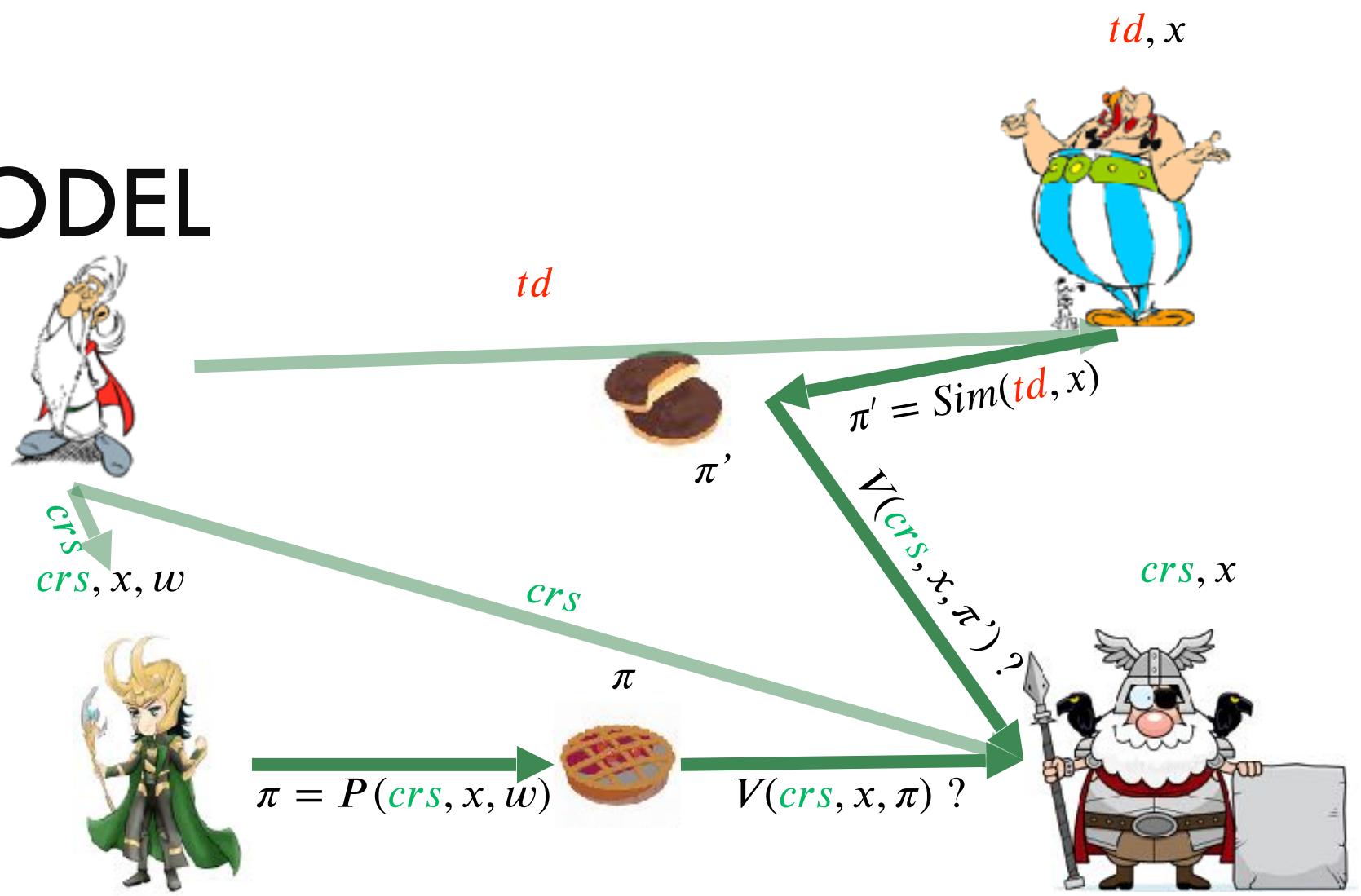
CRS MODEL



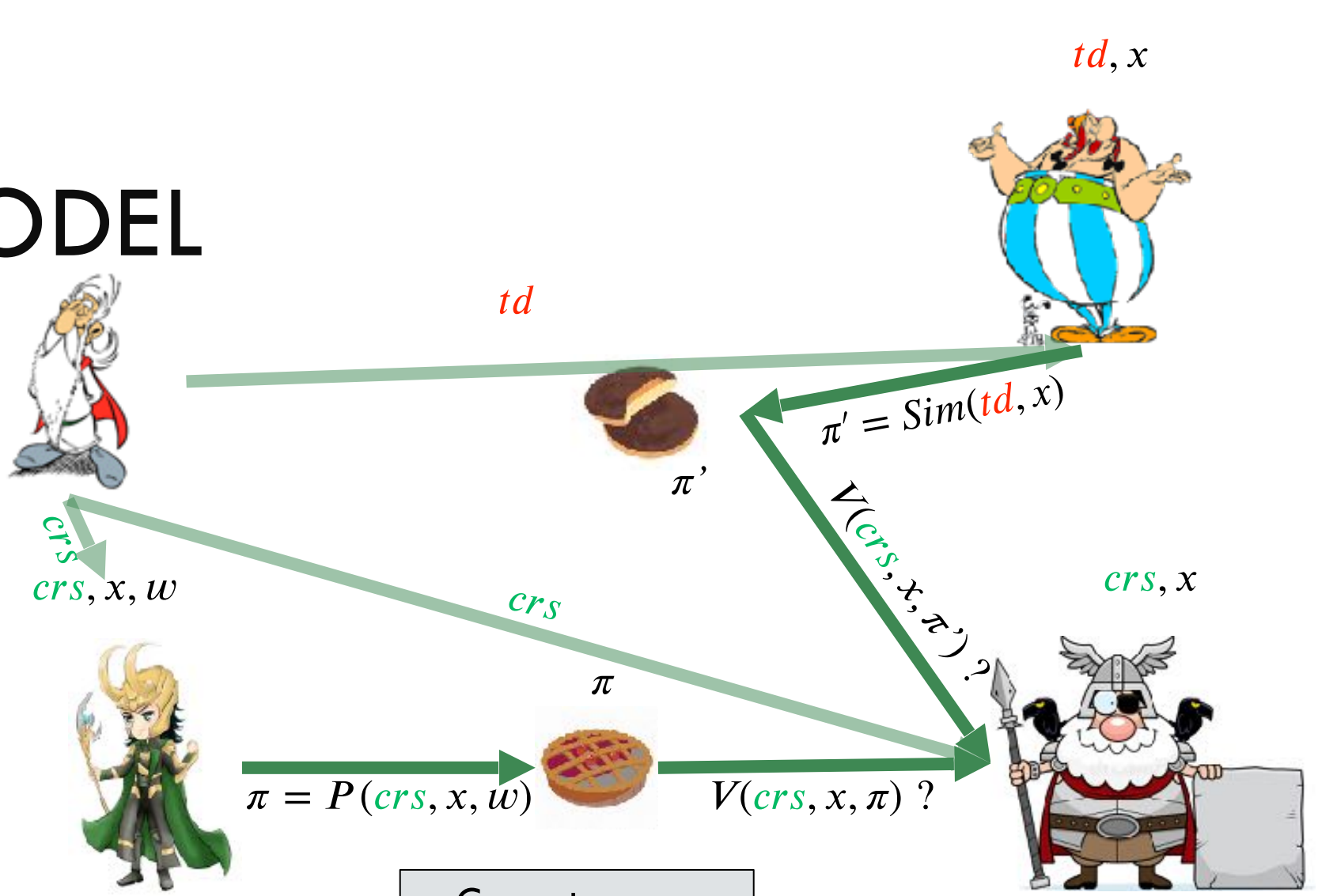
CRS MODEL



CRS MODEL



CRS MODEL



- Correctness
- Soundness
- **Zero-knowledge**

- Efficiency

CRS MODEL

- Universally trusted CRS creator [BluFelMic88]

CRS MODEL

- Universally trusted CRS creator [BluFelMic88]
- **CRS Model Benefits**

CRS MODEL

- Universally trusted CRS creator [BluFelMic88]
- **CRS Model Benefits**
  - Avoids the impossibility results

CRS MODEL

- Universally trusted CRS creator [BluFelMic88]
- **CRS Model Benefits**
  - Avoids the impossibility results
  - Sublinear-time verification: CRS encodes the circuit

CRS MODEL

- Universally trusted CRS creator [BluFelMic88]
- **CRS Model Benefits**
  - Avoids the impossibility results
  - Sublinear-time verification: CRS encodes the circuit
- **CRS Model Drawbacks**

CRS MODEL

- Universally trusted CRS creator [BluFelMic88]
- **CRS Model Benefits**
  - Avoids the impossibility results
  - Sublinear-time verification: CRS encodes the circuit
- **CRS Model Drawbacks**
  - Trust – who is the CRS creator?

EFFICIENT NIZKS (1): ZK-SNARKS

Model	Paper	Trust in CRS creator	Efficiency	Assumptions
CRS Model	Groth10	Everybody trusts	Efficient for NP [GGPR13,Gro16,…]	Non-black-box for soundness (needed)

EFFICIENT NIZKS (1): ZK-SNARKS

Model	Paper	Trust in CRS creator	Efficiency	Assumptions
CRS Model	Groth10	Everybody trusts	Efficient for NP [GGPR13,Gro16,…]	Non-black-box for soundness (needed)
Sub-ZK (BPK model)	ABLZ17, Fuc18	P does not trust, V trusts	-..-	Non-black-box for both soundness and ZK (needed)

EFFICIENT NIZKS (1): ZK-SNARKS

Model	Paper	Trust in CRS creator	Efficiency	Assumptions
CRS Model	Groth10	Everybody trusts	Efficient for NP [GGPR13,Gro16,…]	Non-black-box for soundness (needed)
Sub-ZK (BPK model)	ABLZ17, Fuc18	P does not trust, V trusts	-..-	Non-black-box for both soundness and ZK (needed)
Updatable	GKM+18	P does not trust, V trusts one updater	-..-	-..-

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions
- **Cons**

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions
- **Cons**
  - CRS depends on the language parameter $lpar \leftarrow_{\$} \mathscr{D}_p$

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions
- **Cons**
  - CRS depends on the language parameter $lpar \leftarrow_{\$} \mathscr{D}_p$
  - For wider class of languages, not so efficient

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions
- **Cons**
  - CRS depends on the language parameter $lpar \leftarrow_{\$} \mathscr{D}_p$
  - For wider class of languages, not so efficient
- **Many applications**

ANOTHER TYPE OF NIZK: QA-NIZK

- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions
- **Cons**
  - CRS depends on the language parameter $lpar \leftarrow_{\$} \mathscr{D}_p$
  - For wider class of languages, not so efficient
- **Many applications**
  - separate applications (encryption, signatures, …)

ANOTHER TYPE OF NIZK: QA-NIZK

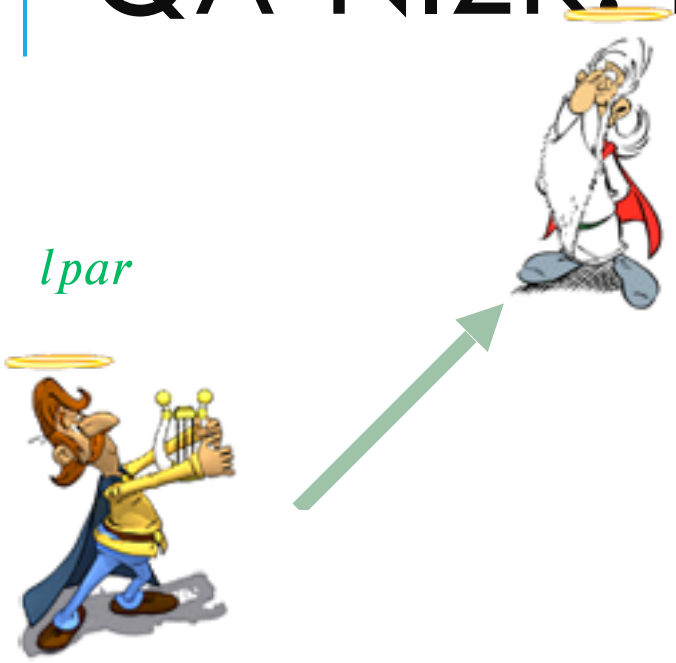
- **Pros**
  - Super efficient for a restricted class of languages
  - Based on standard assumptions
- **Cons**
  - CRS depends on the language parameter $lpar \leftarrow_{\$} \mathscr{D}_p$
  - For wider class of languages, not so efficient
- **Many applications**
  - separate applications (encryption, signatures, …)
  - building block of SNARK variations

QA-NIZK: DEFINITION

$lpar$

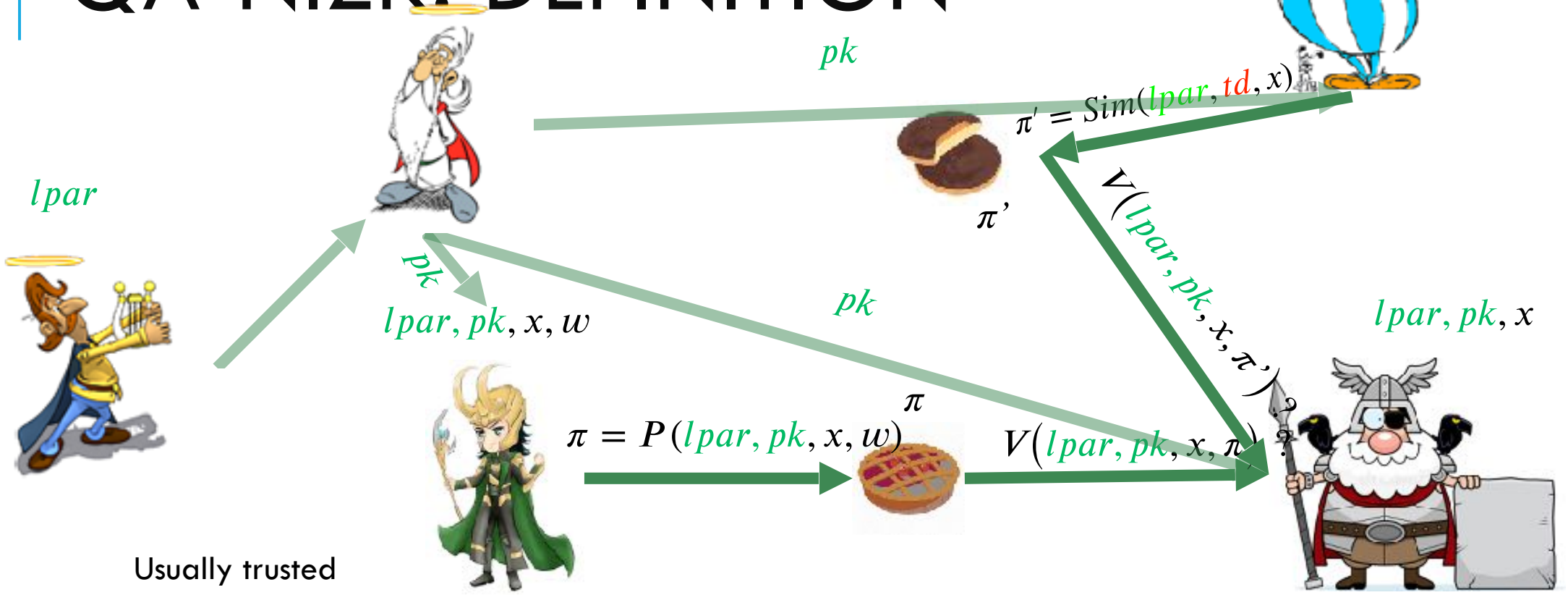
Usually trusted

QA-NIZK: DEFINITION



Usually trusted

QA-NIZK: DEFINITION



Usually trusted

quasi-adaptiveness enables to get improved efficiency
Subspace QA-NIZKs where $|\pi| = 1$ group element

EFFICIENT NIZKS (2): QUASI-ADAPTIVE NIZKS

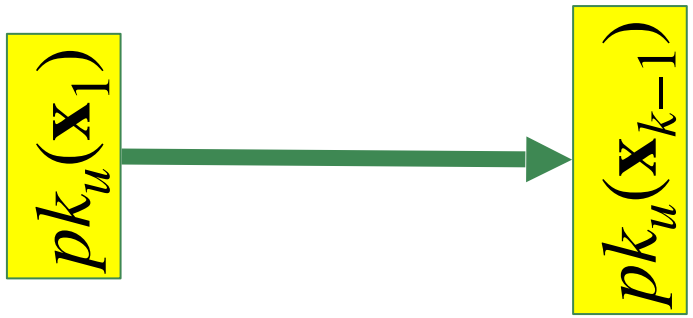
Model	Year	Trust in lpar &/ pk	Efficiency	Assumptions
CRS Model	JR13	Everybody trusts both	Efficient for linear languages, less efficient for NP	Black-box
Sub-ZK (BPK model)	ALSZ20	V trusts pk generator	-..-	NBB for ZK (needed)
Updatable	This work	V trusts one pk updater	-..-	-..-

UPDATABLE SNARKS: IDEA

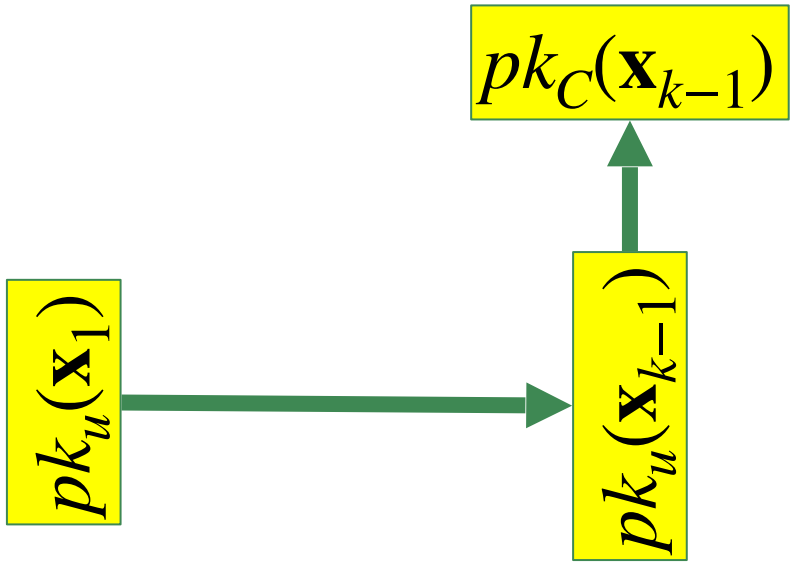
- **Non-updatable zk-SNARKs**
  - $sk = \mathbf{x}$ — a vector of uniformly random integers
  - $pk_C(\mathbf{x}) = \{[f(\mathbf{x})] : f \in S_C\}$ for a C**-dependent** function set S_C
- **Updatable SNARKs**
  - $pk_u(\mathbf{x}) := \{[f(\mathbf{x})] : f \in S_n\}$ for a universal set S_n
    - …universal for all circuits of size $\leq n$
  - verifiably specialize $pk_u(\mathbf{x}) \Rightarrow pk_C(\mathbf{x})$ to concrete circuit C

UPDATABLE SNARKS: DETAILS

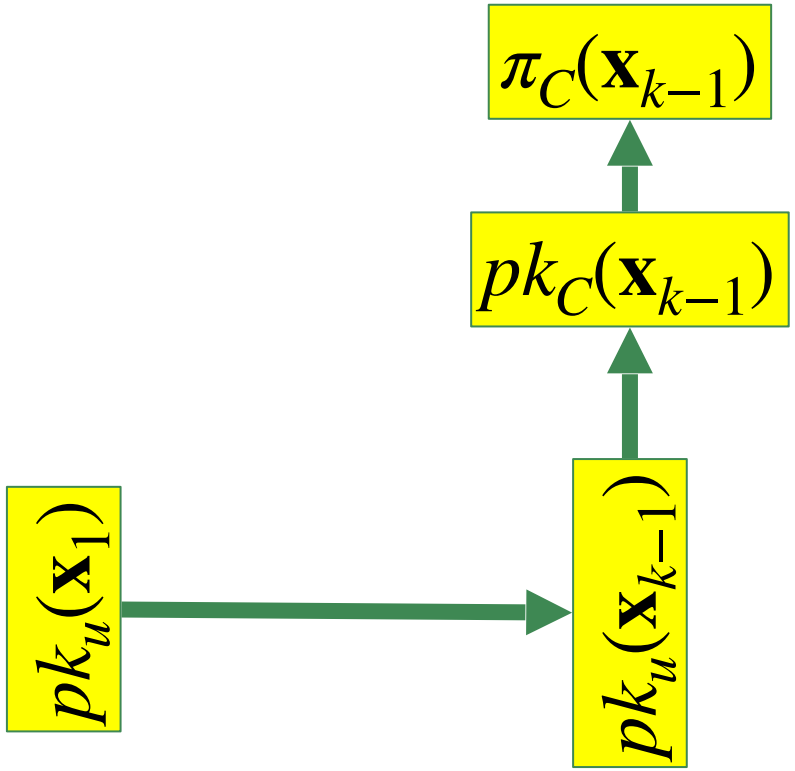
UPDATABLE SNARKS: DETAILS



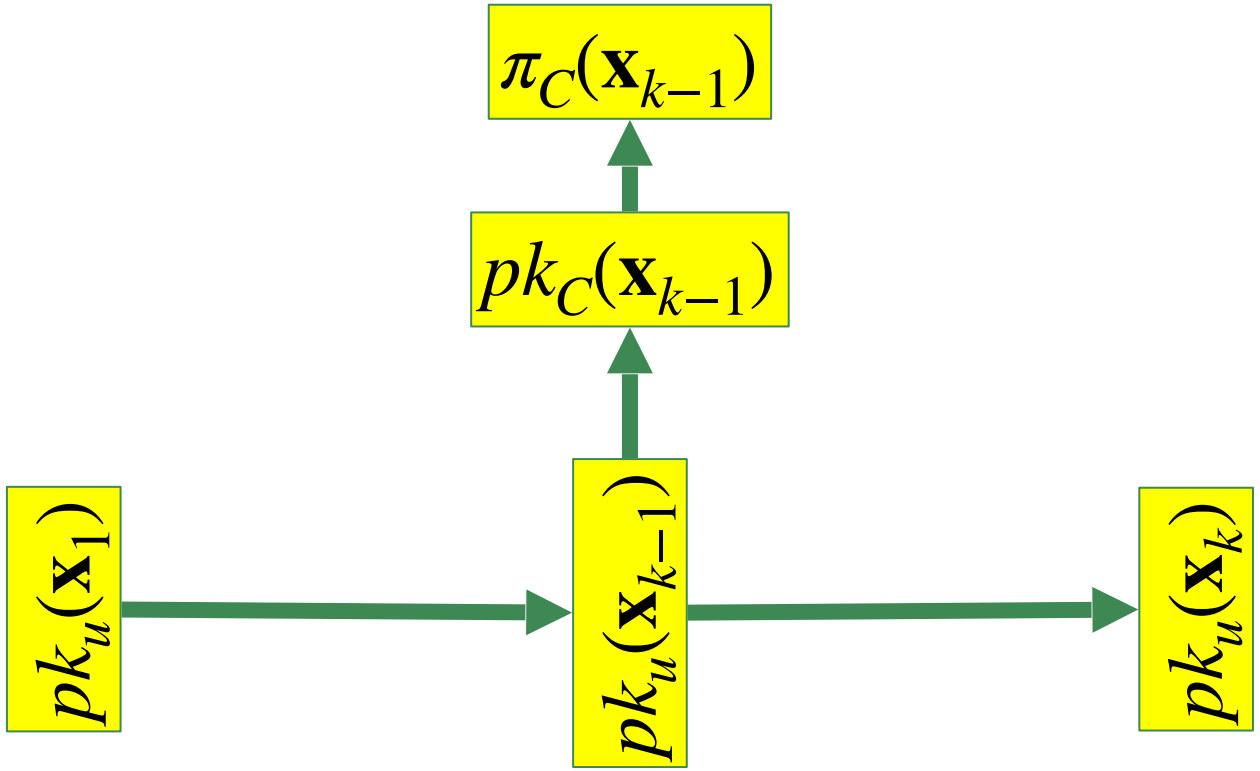
UPDATABLE SNARKS: DETAILS



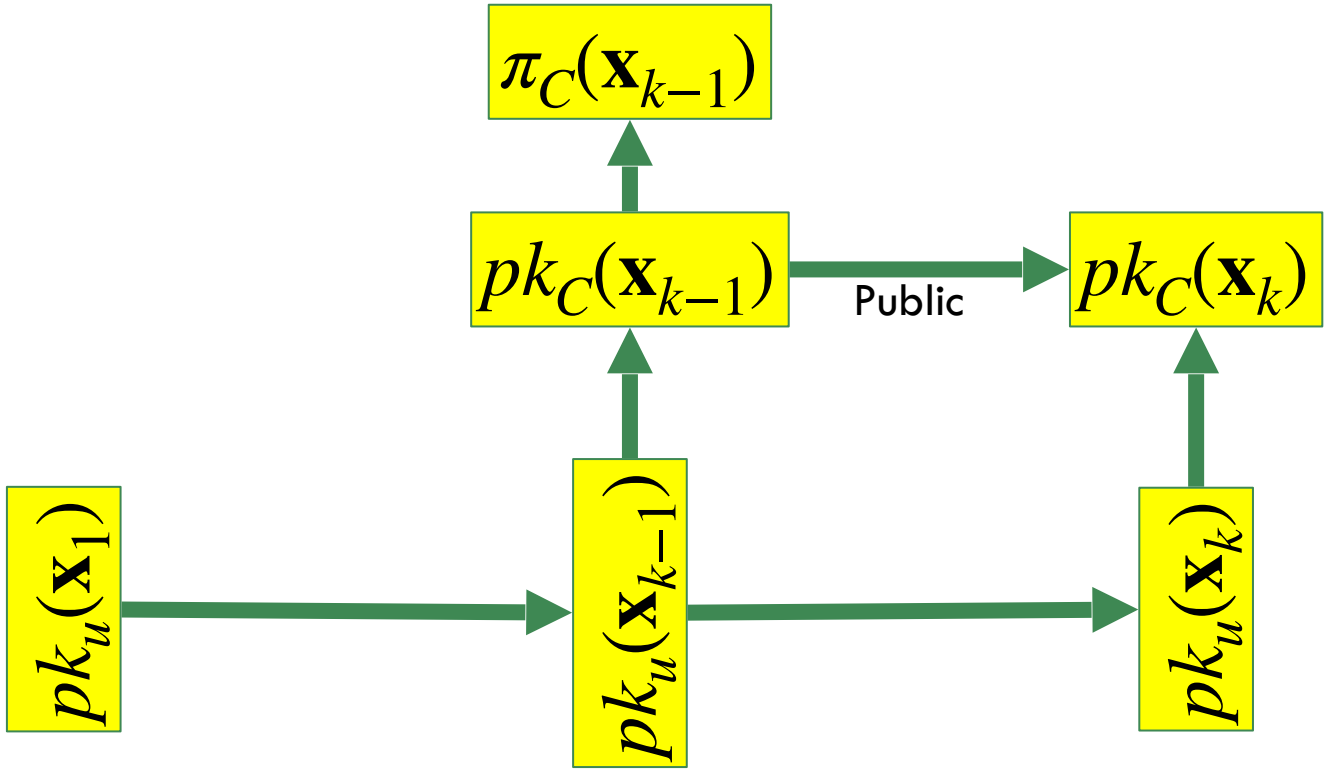
UPDATABLE SNARKS: DETAILS



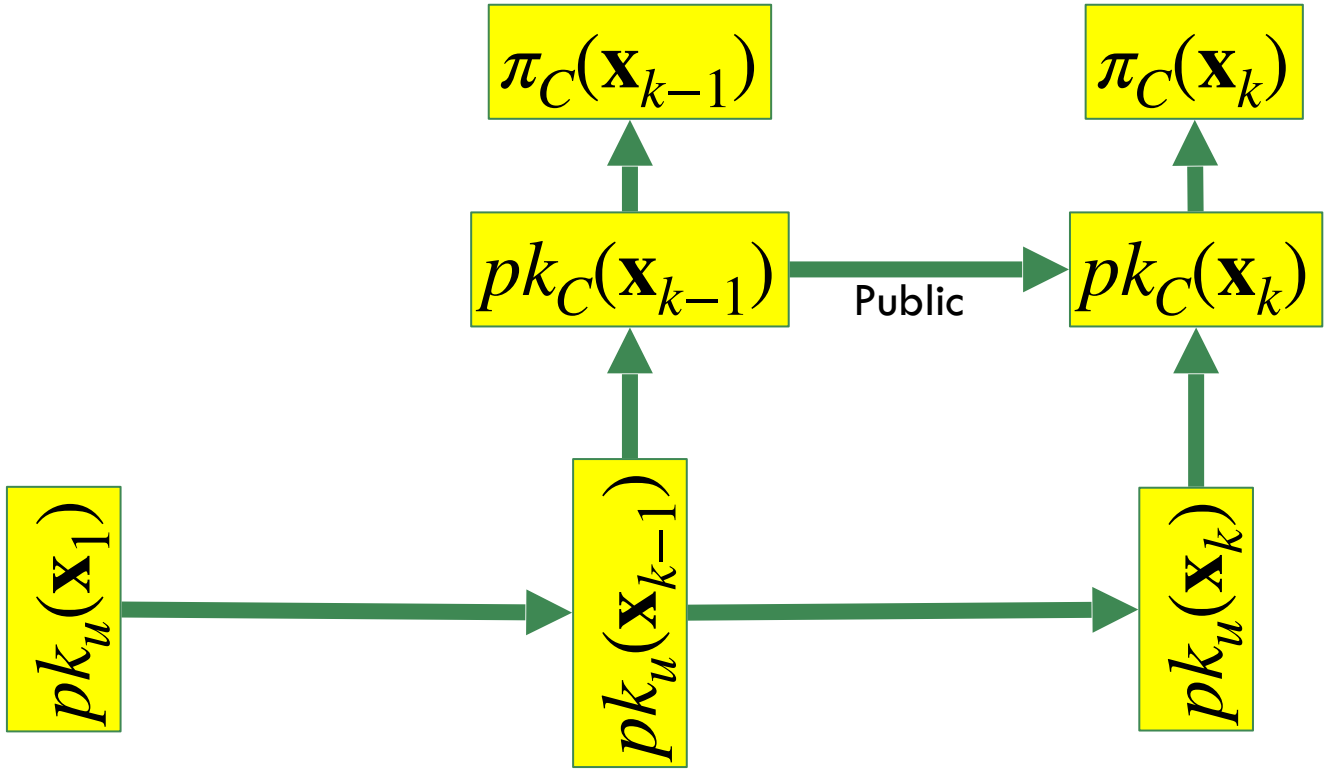
UPDATABLE SNARKS: DETAILS



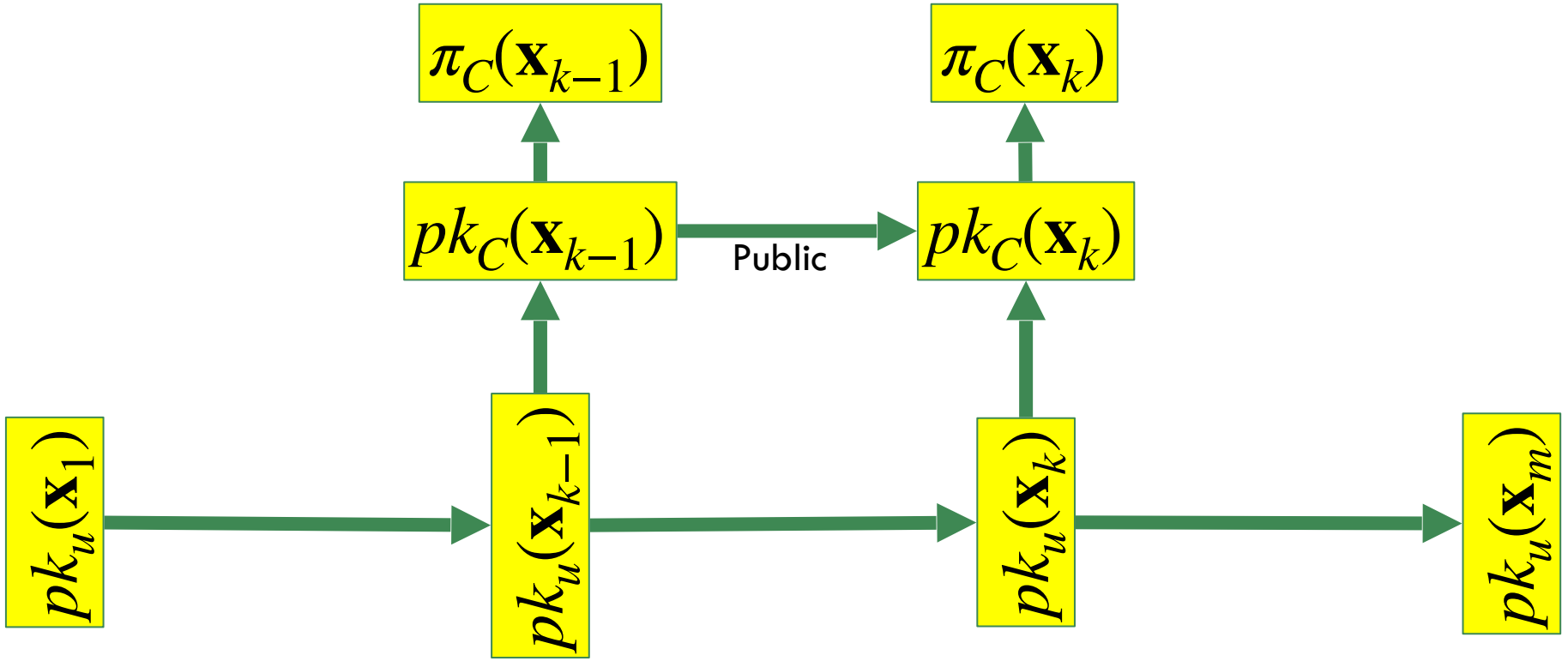
UPDATABLE SNARKS: DETAILS



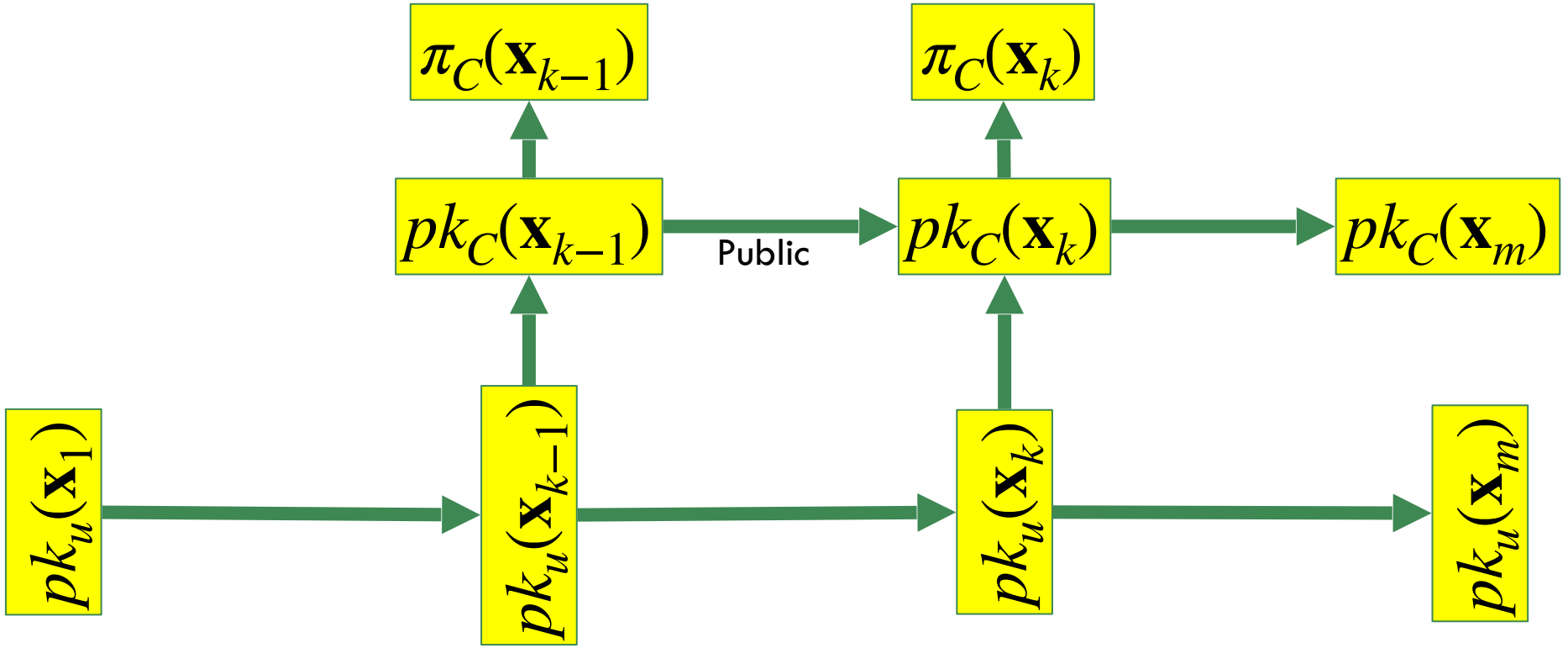
UPDATABLE SNARKS: DETAILS



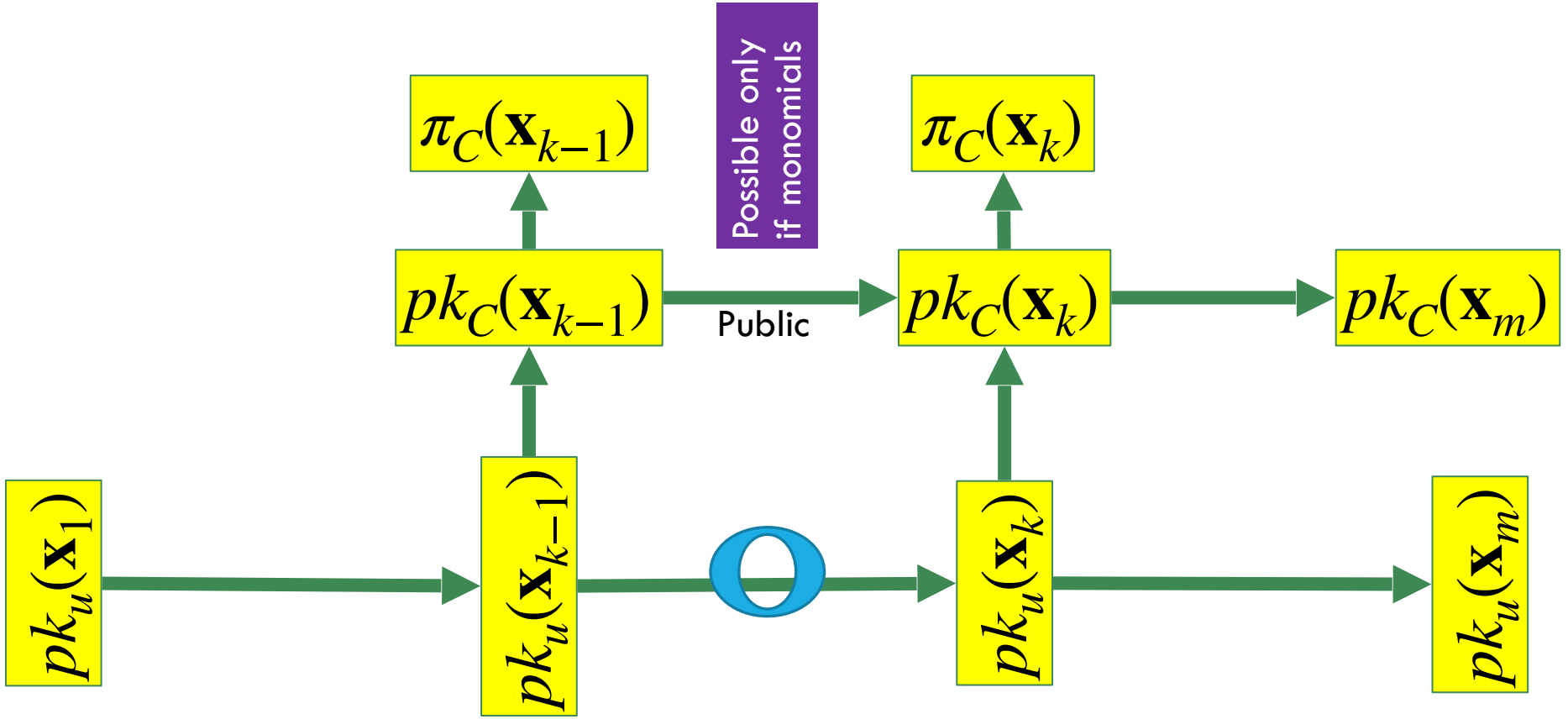
UPDATABLE SNARKS: DETAILS



UPDATABLE SNARKS: DETAILS



UPDATABLE SNARKS: DETAILS



SECURITY GOALS

- **Specialized pk updatable publicly**

SECURITY GOALS

- **Specialized pk updatable publicly**
- **Universal pk creation/update efficiently verifiable**

SECURITY GOALS

- **Specialized pk updatable publicly**
- **Universal pk creation/update efficiently verifiable**
  - Creation corresponds to some choice of $\mathbf{x}_1$

SECURITY GOALS

- **Specialized pk updatable publicly**
- **Universal pk creation/update efficiently verifiable**
  - Creation corresponds to some choice of $\mathbf{x}_1$
  - Update corresponds to some choice of $\delta\mathbf{x}_k$

SECURITY GOALS

- **Specialized pk updatable publicly**
- **Universal pk creation/update efficiently verifiable**
  - Creation corresponds to some choice of $\mathbf{x}_1$
  - Update corresponds to some choice of $\delta\mathbf{x}_k$
  - No trust needed!

SECURITY GOALS

- **Specialized pk updatable publicly**
- **Universal pk creation/update efficiently verifiable**
  - Creation corresponds to <u>some</u> choice of $\mathbf{x}_1$
  - Update corresponds to <u>some</u> choice of $\delta\mathbf{x}_k$
  - No trust needed!
- **If one update is honest then $\mathbf{x}_k$ is uniformly random**

SECURITY GOALS

- **Specialized pk updatable publicly**
- **Universal pk creation/update efficiently verifiable**
  - Creation corresponds to some choice of $\mathbf{x}_1$
  - Update corresponds to some choice of $\delta\mathbf{x}_k$
  - No trust needed!
- **If one update is honest then $\mathbf{x}_k$ is uniformly random**
  - Our observation: for this we need the distribution of sk belongs to a generalized ideal of a convolution semigroup ("nice distribution")

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i \in \{1,2\}}$ for well-chosen polynomials sets S_{Ci}

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i \in \{1,2\}}$ for well-chosen polynomials sets S_{Ci}
- **Multiplicative update:** $\mathbf{x}_k := \mathbf{x}_{k-1} \circ \delta\mathbf{x}_k$

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i \in \{1,2\}}$ for well-chosen polynomials sets S_{Ci}
- **Multiplicative update:** $\mathbf{x}_k := \mathbf{x}_{k-1} \circ \delta\mathbf{x}_k$
- Can be used to efficiently update [monomials]:

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i \in \{1,2\}}$ for well-chosen polynomials sets S_{Ci}
- **Multiplicative update:** $\mathbf{x}_k := \mathbf{x}_{k-1} \circ \delta\mathbf{x}_k$
- Can be used to efficiently update [monomials]:
  - For example: $[x^2_{k-1,1} x^3_{k-1,2} x_{k-1,3}]_1 \cdot \delta x^2_{k,1} \delta x^3_{k,2} \delta x_{k,3} = [x^2_{k,1} x^3_{k,2} x_{k,3}]_1$

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i \in \{1,2\}}$ for well-chosen polynomials sets S_{Ci}
- **Multiplicative update:** $\mathbf{x}_k := \mathbf{x}_{k-1} \circ \delta\mathbf{x}_k$
- Can be used to efficiently update [monomials]:
  - For example: $[x^2_{k-1,1} x^3_{k-1,2} x_{k-1,3}]_1 \cdot \delta x^2_{k,1} \delta x^3_{k,2} \delta x_{k,3} = [x^2_{k,1} x^3_{k,2} x_{k,3}]_1$
- [GKM+18]: non-monomials **cannot be updated** by using M.U.

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i\in\{1,2\}}$ for well-chosen polynomials sets S_{Ci}
- **Multiplicative update:** $\mathbf{x}_k := \mathbf{x}_{k-1} \circ \delta\mathbf{x}_k$
- Can be used to efficiently update [monomials]:
  - For example: $[x^2_{k-1,1}x^3_{k-1,2}x_{k-1,3}]_1 \cdot \delta x^2_{k,1}\delta x^3_{k,2}\delta x_{k,3} = [x^2_{k,1}x^3_{k,2}x_{k,3}]_1$
- [GKM+18]: non-monomials **cannot be updated** by using M.U.
  - A flurry in research in SNARKs where pk consists of only monomials

GKM+18: MULTIPLICATIVE UPDATES

- In efficient (non-updatable) zk-SNARKs:
  - $td = \mathbf{x}$ a vector of uniformly random integers
  - $pk = ([f(\mathbf{x}) : f \in S_{Ci}]_i)_{i\in\{1,2\}}$ for well-chosen polynomials sets S_{Ci}
- **Multiplicative update:** $\mathbf{x}_k := \mathbf{x}_{k-1} \circ \delta\mathbf{x}_k$
- Can be used to efficiently update [monomials]:
  - For example: $[x^2_{k-1,1}x^3_{k-1,2}x_{k-1,3}]_1 \cdot \delta x^2_{k,1}\delta x^3_{k,2}\delta x_{k,3} = [x^2_{k,1}x^3_{k,2}x_{k,3}]_1$
- [GKM+18]: non-monomials **cannot be updated** by using M.U.
  - A flurry in research in SNARKs where pk consists of only monomials
  - … cannot reuse existing techniques of [GGPR13,Gro16,…]

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**
- **Different updating strategy** // KW CRS does not consist of monomials

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**
- **Different updating strategy** // KW CRS does not consist of monomials
  - Divide KW pk into two parts: $[\mathbf{A}]_2$ (universal KerMDH matrix), C-dependent pk

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**
- **Different updating strategy** // KW CRS does not consist of monomials
  - Divide KW pk into two parts: $[\mathbf{A}]_2$ (universal KerMDH matrix), C-dependent pk
  - Update $[\mathbf{A}]_2$ *multiplicatively*; update C-dependent pk *additively*

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**
- **Different updating strategy** // KW CRS does not consist of monomials
  - Divide KW pk into two parts: $[\mathbf{A}]_2$ (universal KerMDH matrix), C-dependent pk
  - Update $[\mathbf{A}]_2$ *multiplicatively*; update C-dependent pk *additively*
- **Generalization:**

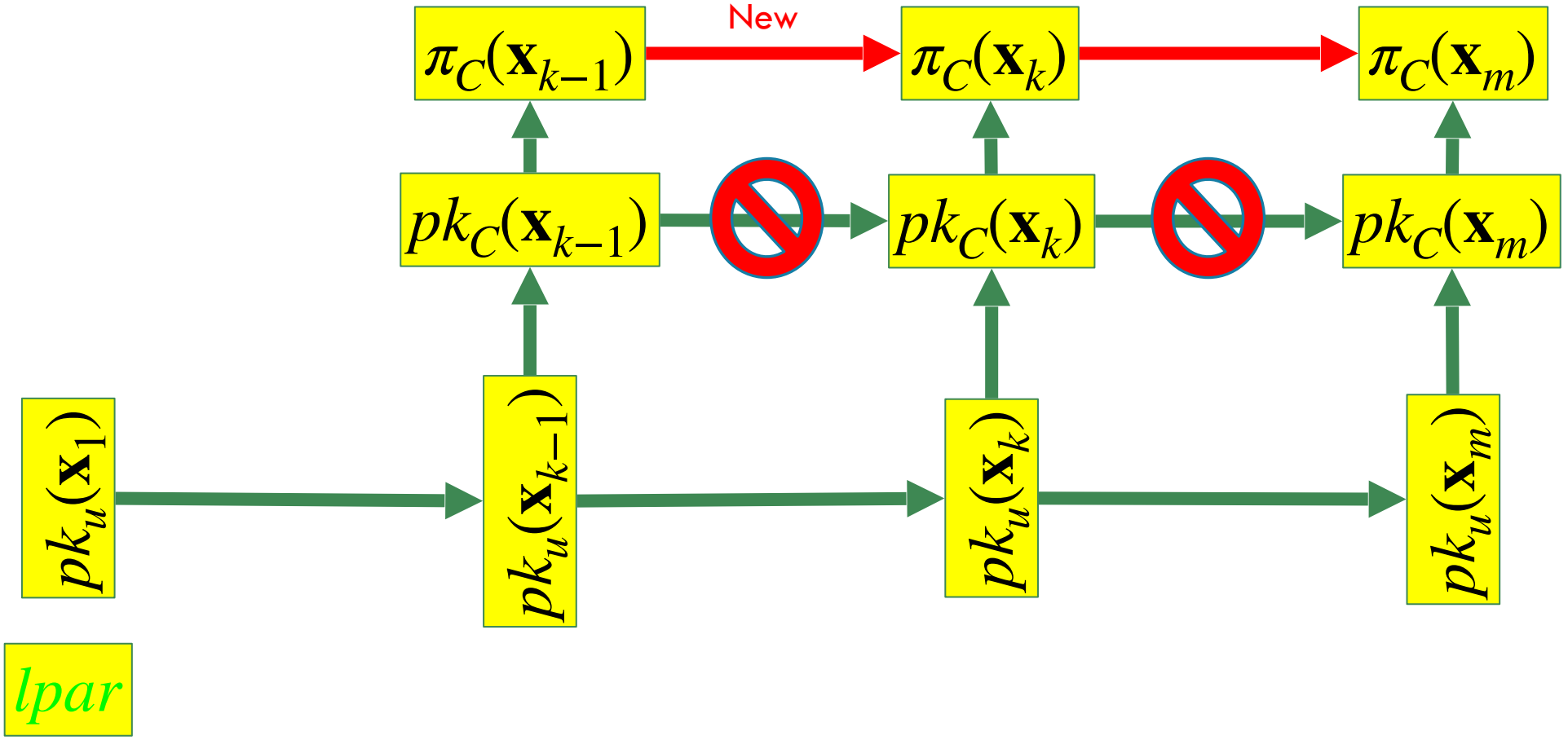
OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**
- **Different updating strategy** // KW CRS does not consist of monomials
  - Divide KW pk into two parts: $[\mathbf{A}]_2$ (universal KerMDH matrix), C-dependent pk
  - Update $[\mathbf{A}]_2$ *multiplicatively*; update C-dependent pk *additively*
- **Generalization:**
  - Use/update $[\mathbf{A}]_2$ in different applications that use KerMDH

OUR RESULTS

- **Construction:** updatable version of Kiltz-Wee QA-NIZK
- **Argument-updatability:** // previous updatable constructions did not have it
  - We update π together with pk
  - … it looks like fresh argument with new pk
- **Corresponding security definitions**
- **Different updating strategy** // KW CRS does not consist of monomials
  - Divide KW pk into two parts: $[\mathbf{A}]_2$ (universal KerMDH matrix), C-dependent pk
  - Update $[\mathbf{A}]_2$ *multiplicatively*; update C-dependent pk *additively*
- **Generalization:**
  - Use/update $[\mathbf{A}]_2$ in different applications that use KerMDH
  - Update the rest of the pk separately in each application

K&A UPDATABLE QA-NIZKS



K&A UPDATABLE QA-NIZKS

Caveat: updated π is as sound as original π