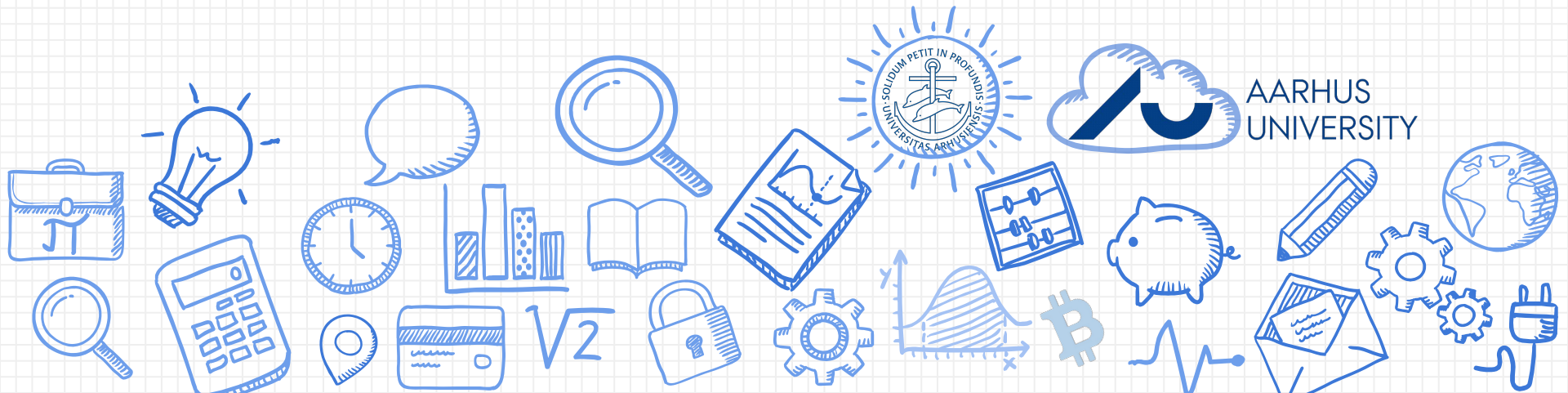
Lattice-Based zero-knowledge SNARGs for Arithmetic Circuits

Anca Nitulescu

AARHUS UNIVERSITY



Outline

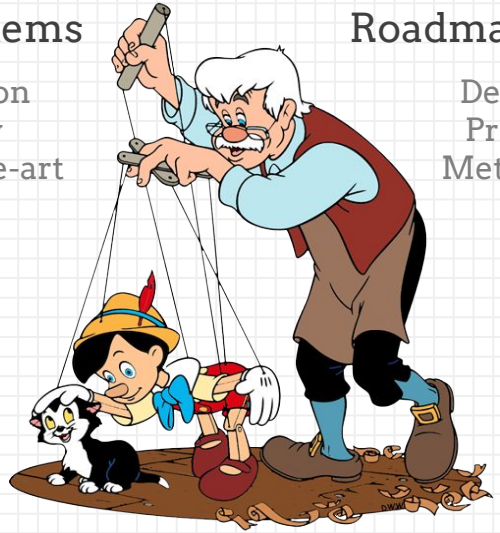
SNARG
Background

Definitions
for SNARGs

Construction
Security

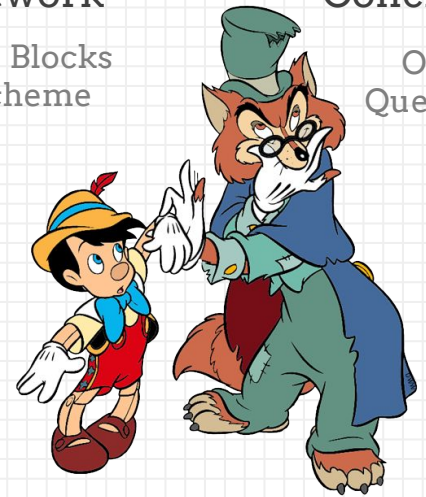
The END

Proof Systems
Motivation
History
State-of-the-art



Roadmap and Tools
Definitions
Properties
Methodology

Framework
Building Blocks
New Scheme



Conclusions
Open
Questions

SNARK

SNARG
Background

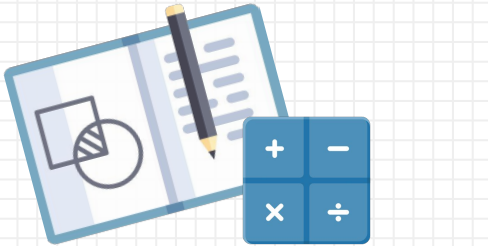
Definitions
for SNARGs

Construction
Security

The
END

Delegated Computation

Verifier



Task

computes
f(x)=**y**

Prover

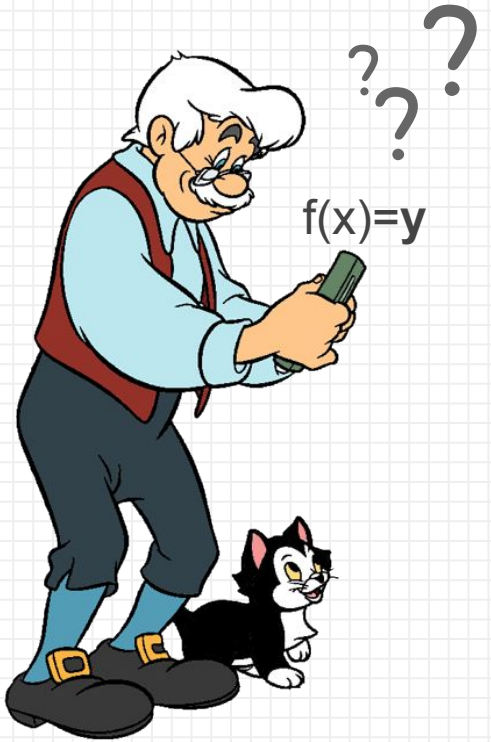
Prover claims a statement

Verifier

Claim
y=f(x)

Prover

Verifier does not trust

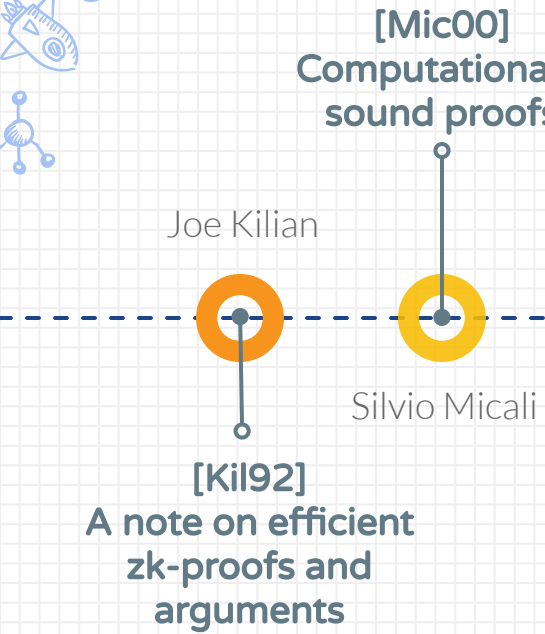


Verifier

y ≠ f(x)

Corrupted Prover

Proof Systems: Non-Interactive Arguments

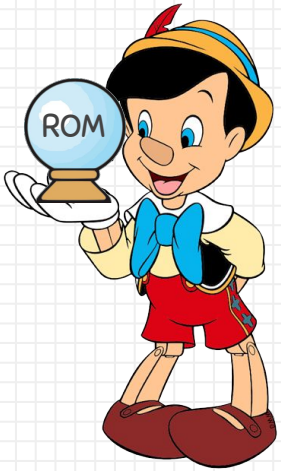


Non-Interactive Proof Protocol [Mic00]

Verifier

Claim
y=f(x)

Proof π



Prover

Pre-Processing for Efficient Arguments

[Mic00]
Computationally sound proofs

Joe Kilian

Silvio Micali

G. Di Crescenzo
Helger Lipmaa

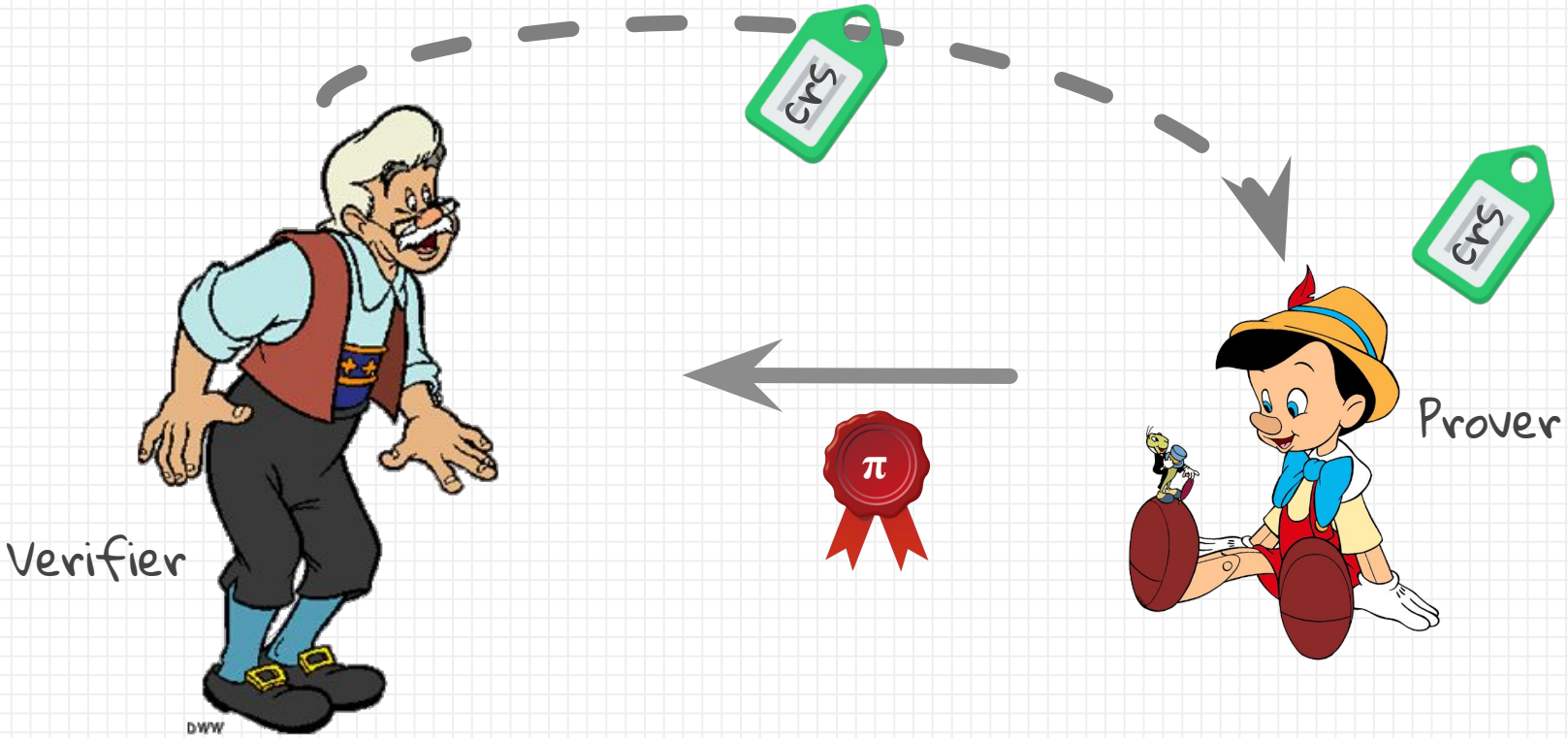
J. Groth

[Kil92]
A note on efficient zk-proofs and arguments

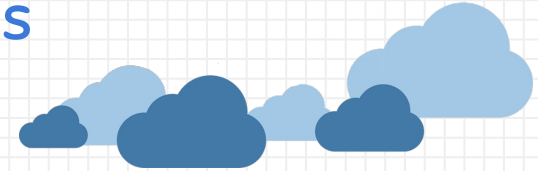
[DCL08]
Succinct NP proofs from an extractability assumption

[Gro10]
Short Pairing-based Non-interactive Zero-Knowledge Arguments

One round Interaction



Strong Assumptions



Joe Kilian

[Kil92]
A note on efficient zk-proofs and arguments

[Mic00]
Computationally sound proofs

Silvio Micali

G. Di Crescenzo
Helger Lipmaa

[DCL08]
Succinct NP proofs from an extractability assumption

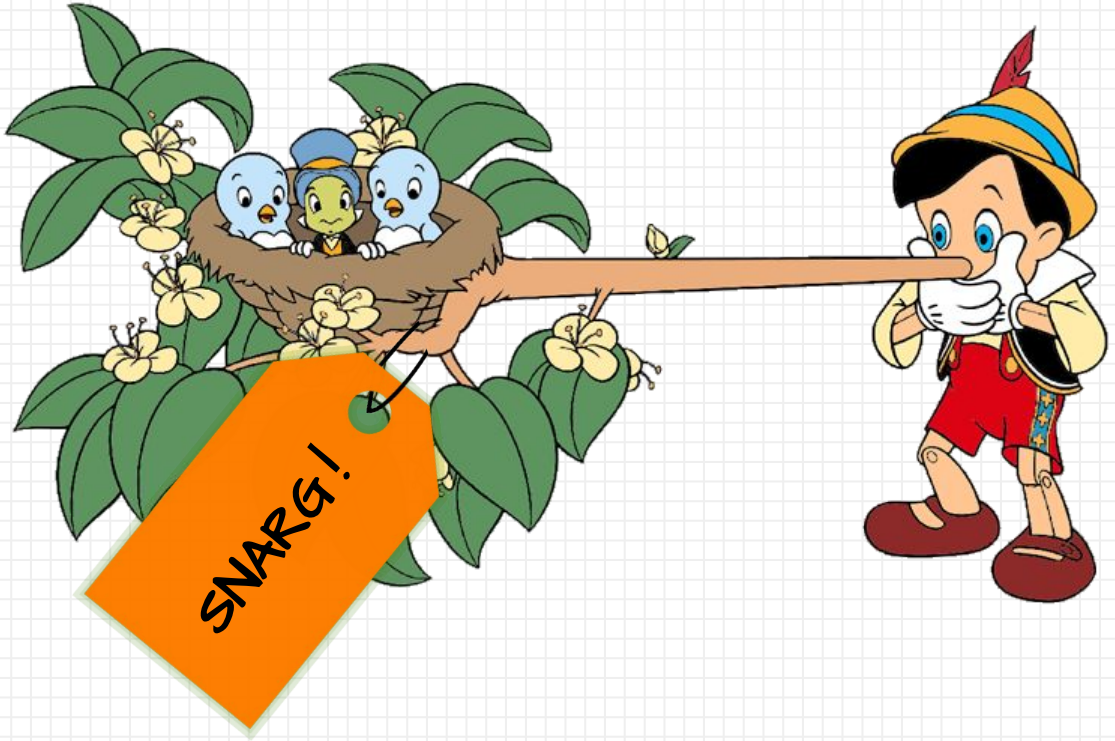
J. Groth

[Gro10]
Short Pairing-based Non-interactive Zero-Knowledge Arguments

[GW11]
Separating succinct non-interactive arguments from all falsifiable assumptions

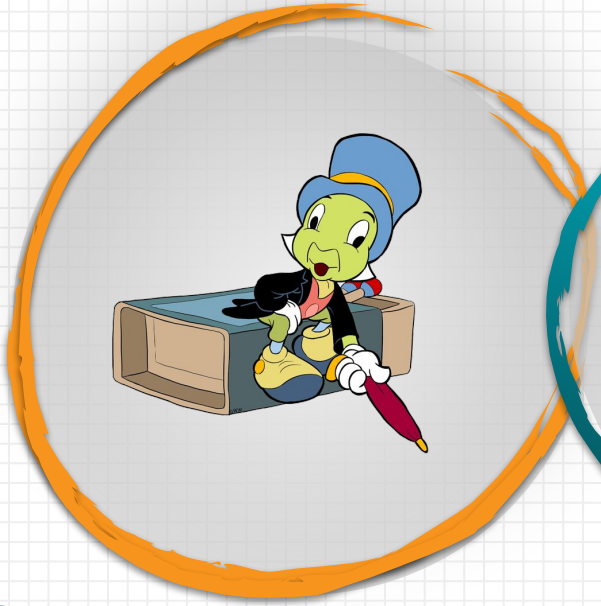
Craig Gentry
Daniel Wichs

Succinct Non-interactive ARGument

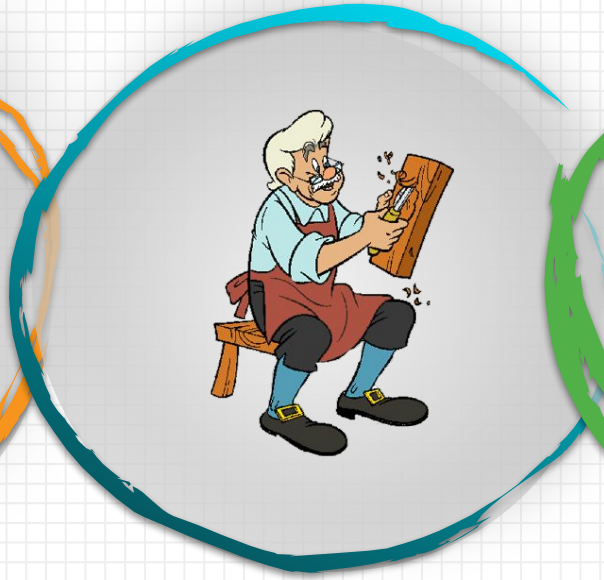


Properties of a SNARG

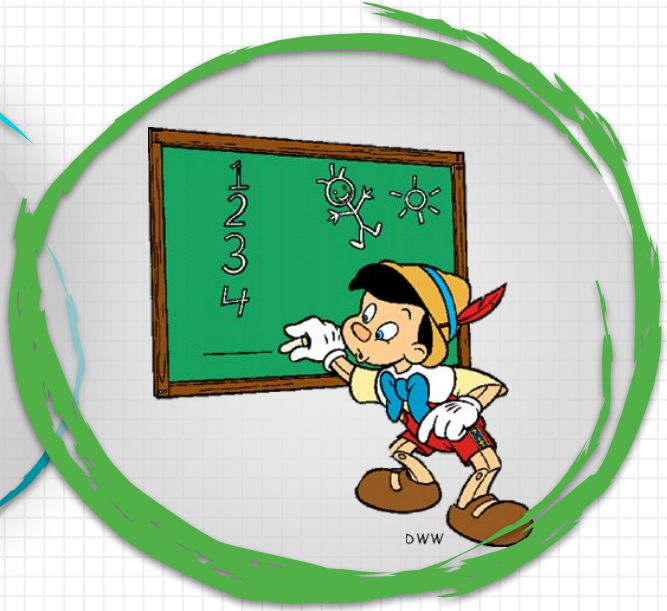
Succinct Proof



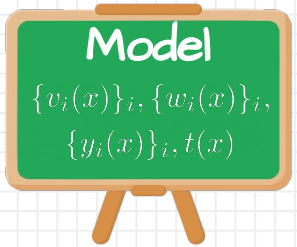
Efficient Verification



Computational Soundness



SNARG: Methodology



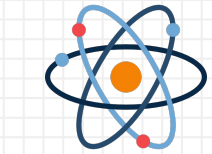
Target Statement
R(y,w)=1

Computational Model
(Representation)

SNARG
under
Non-Falsifiable Assumptions

Computation y=F(x)	**PCP:** Probabilistically Checkable Proofs	**ECRH:** Extractable Collision-Resistant Hash
Boolean Circuit SAT	**QSP / SSP:** Quadratic / Square Span Programs	**PKE:** Power Knowledge of Exponent
Arithmetic Circuit SAT	**QAP / SAP:** Q / S Arithmetic Programs	**PKE:** Power Knowledge of Exponent

State-of-the-art



PKE

ECRH

[BCI+13]
SNARGs via linear interactive proofs

R. Gennaro, C. Gentry, B. Parno, M. Raykova

B. Parno, J.Howell, C. Gentry, M. Raykova

N.Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, O Paneth

[GGPR13]
QSP and succinct NIZKs without PCPs

[PHGR13]
Pinocchio: Nearly practical verifiable computation

Post-Quantum Succinct Arguments

[PHGR13]
Pinocchio: Nearly practical verifiable computation

B. Parno, J.Howell, C. Gentry, M. Raykova

[BCI+13]
SNARGs via linear interactive proofs

N.Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, O Paneth

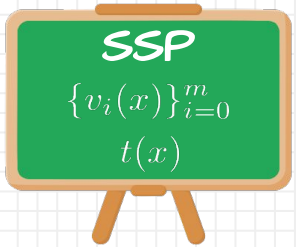
[BISW17]
Lattice-based SNARGs and their application to more efficient obfuscation

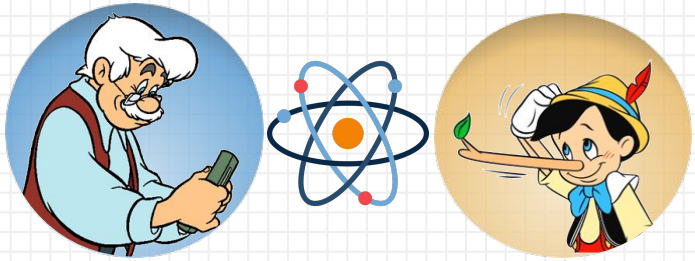
D. Boneh, Y. Ishai, A. Sahai, D.J. Wu

[GMNO18]
Lattice-based zk-SNARKs from SSP

R. Gennaro, M. Minelli, **Anca Nitulescu,** M. Orrù

Post-Quantum SNARGs





Target Statement R(y,w)=1	Computational Model (Representation)	SNARG under Post-Quantum Assumptions
[BISW17]	**PCP:** Probabilistically Checkable Proofs	(Strong) Vector Linear-Only Encryption
[GMNO18] Boolean Circuit SAT	**QSP / SSP:** Quadratic / Square Span Programs	PKE on Lattice Encodings
Arithmetic Circuit SAT	**QAP / SAP:** Q / Square Arithmetic Programs	?

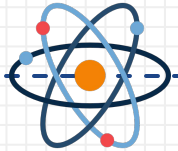
Post-Quantum Succinct Arguments

[PHGR13]
Pinocchio: Nearly practical verifiable computation

B. Parno, J.Howell, C. Gentry, M. Raykova

[BCI+13]
SNARGs via linear interactive proofs

N.Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, O Paneth



[BISW17]
Lattice-based SNARGs and their application to more efficient obfuscation

D. Boneh, Y. Ishai, A. Sahai, D.J. Wu

[GMNO18]
Lattice-based zk-SNARKs from SSP

R. Gennaro, M. Minelli, **Anca Nitulescu,** M. Orrù

[this work]
Lattice-based zk-SNARGs from SAP

Anca Nitulescu

Defining SNARGs

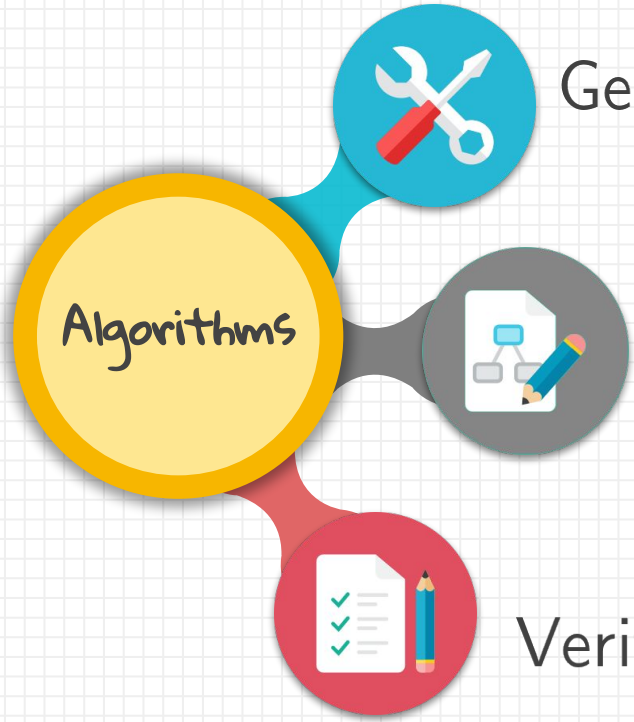
SNARK
Background

Definitions
for SNARGs

Construction
Security

The
END

SNARG with Preprocessing

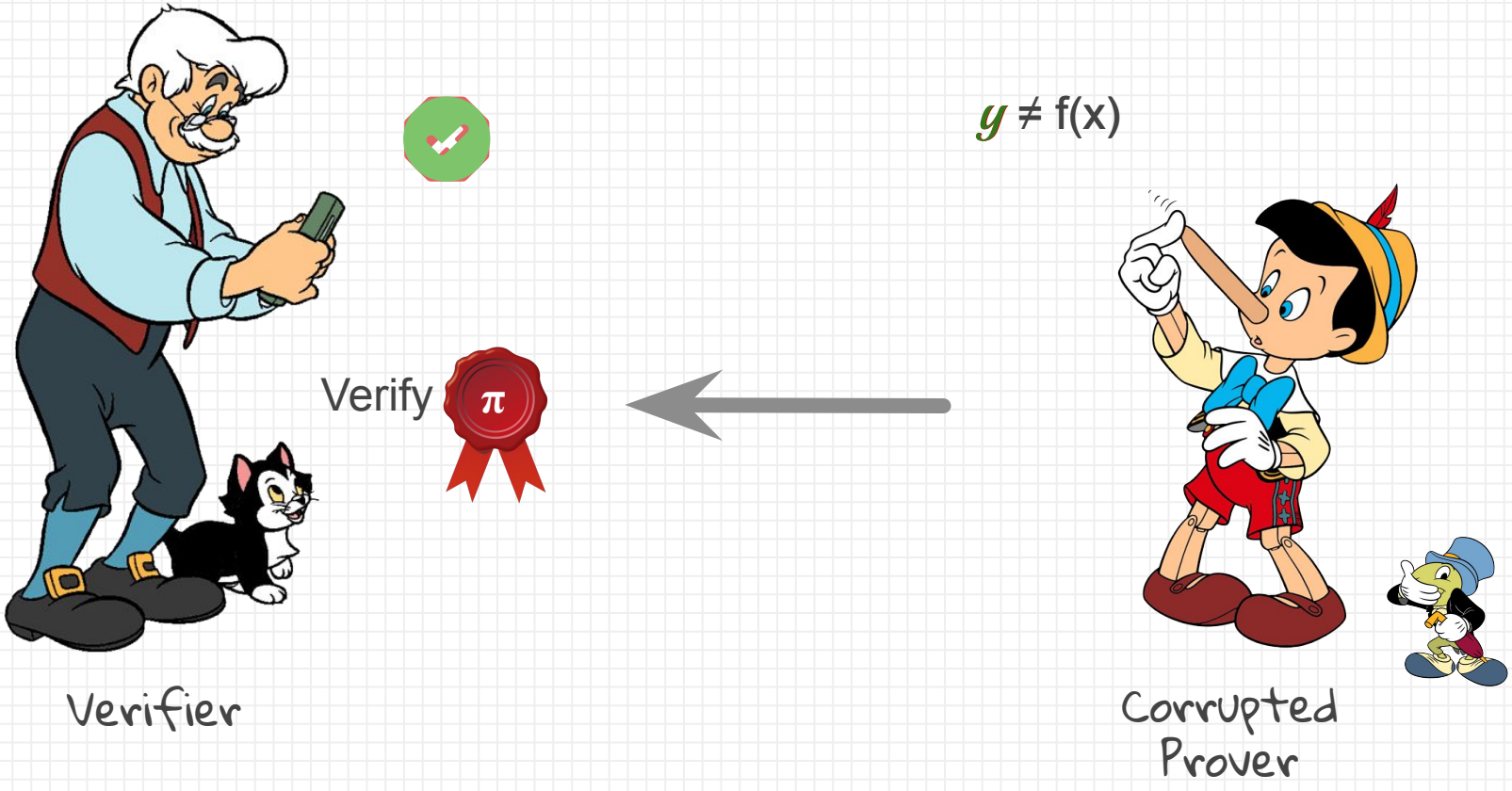


$\mathsf{Gen}(1^\lambda, \mathcal{R}) \to (\mathsf{crs}, \mathsf{vk})$

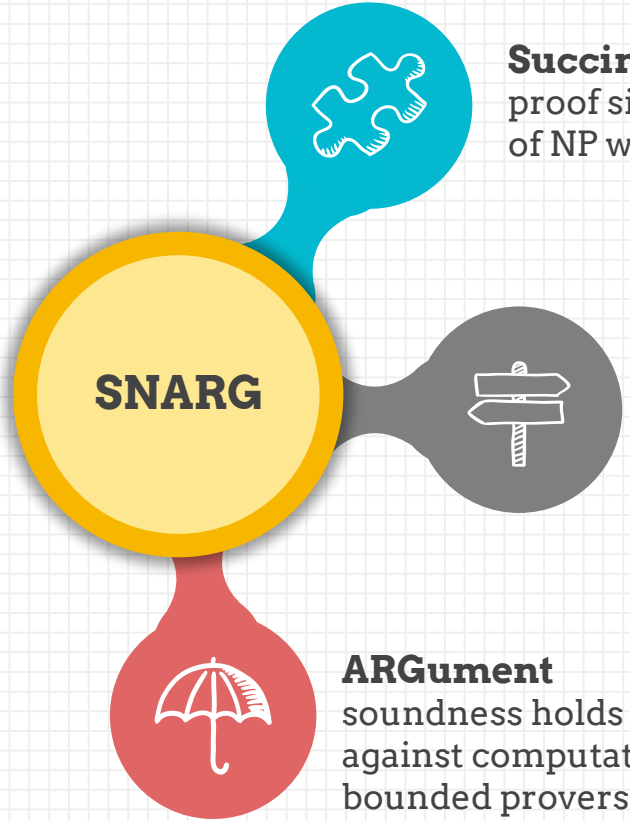
$\mathsf{Prove}(\mathsf{crs}, y, w) \to \pi : (y, w) \in \mathcal{R}$

$\mathsf{Verify}(\mathsf{vk}, y, \pi) \to 0/1$

Correctness and Soundness



SNARG: Succinct Non-Interactive ARGument

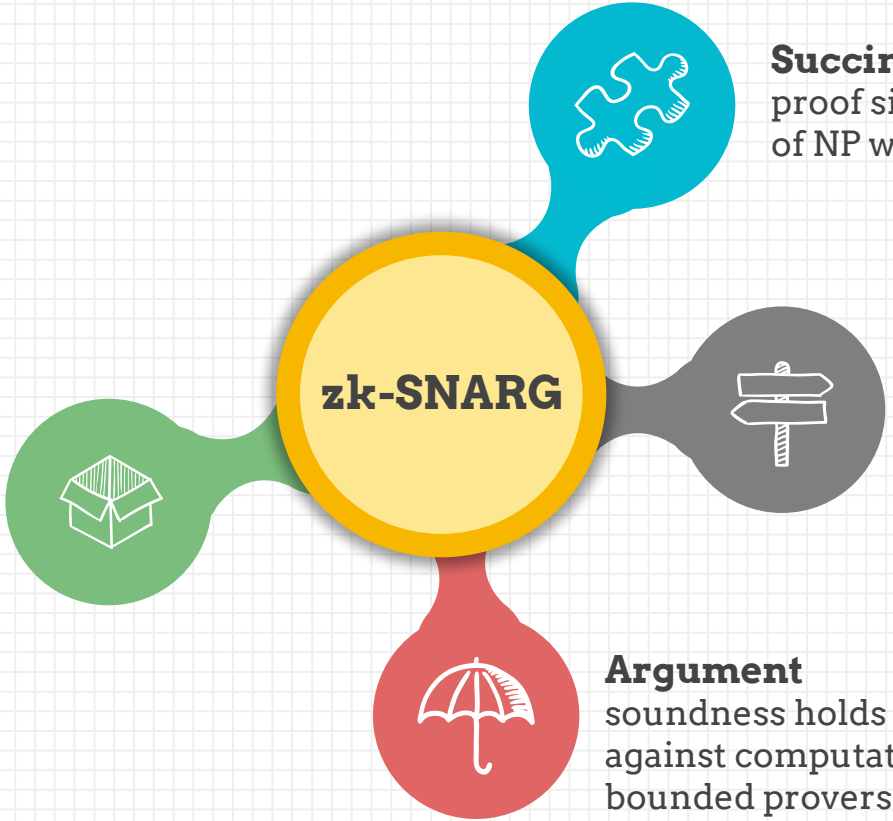


Succinctness
proof size independent of NP witness size

Non-Interactivity
no exchange between prover and verifier

ARGument
soundness holds only against computationally bounded provers

Zero-Knowledge SNARG



Succinctness
proof size independent
of NP witness size

Non-Interactivity
no exchange between
prover and verifier

Argument
soundness holds only
against computationally
bounded provers

Zero-Knowledge
does not leak anything
about the witness

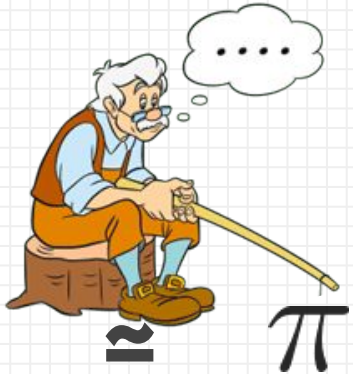
Zero-Knowledge

(crs, vk)
trapdoor

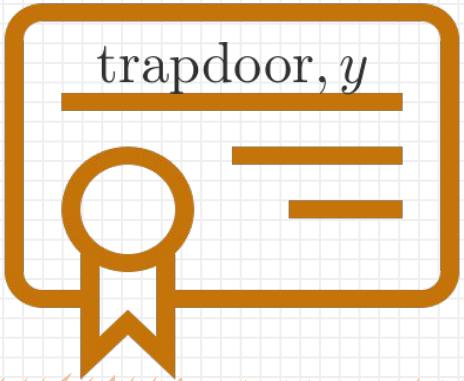
Simulator

(crs, vk)

Prover

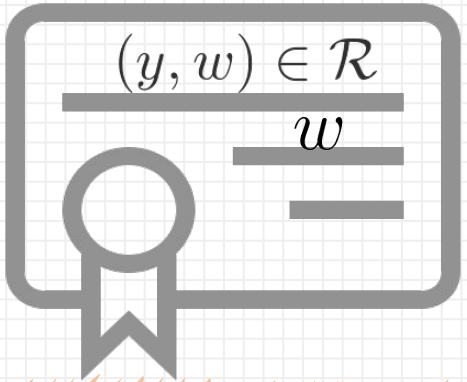


π'



$\approx$

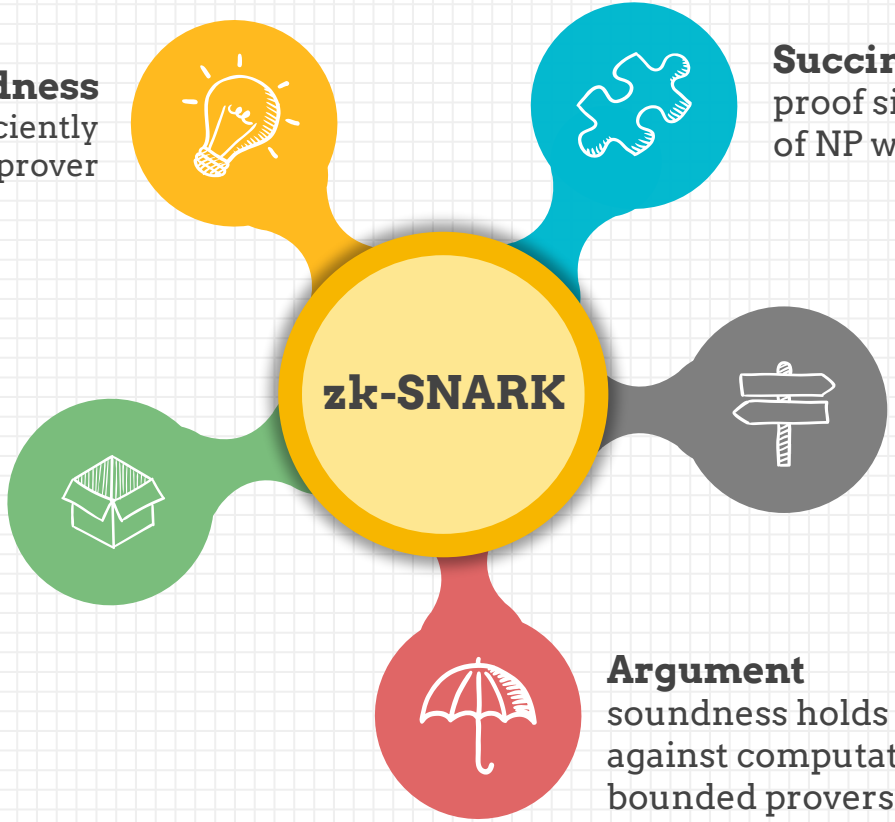
π



SNARK: Succinct Non-Interactive ARgument of Knowledge

Knowledge Soundness
a witness can be efficiently extracted from the prover

Succinctness
proof size independent of NP witness size



Zero-Knowledge
does not leak anything about the witness

Non-Interactivity
no exchange between prover and verifier

Argument
soundness holds only against computationally bounded provers

SNARG comparison

	BISW17 Lattice-based SNARG from PCP	GMNO18 Lattice-Based zk-SNARK from SSP	This work Lattice-Based zk-SNARG from SAP
computational model	PCP	SSP	SAP
assumption	strong vector linear-only	lattice PKE	linear-only
proof size	1 vector of ciphertexts	5 ciphertexts	2 ciphertexts
zero-knowledge	✗	✓	✓
knowledge soundness	✗	✓	✗
arithmetic circuit	✗	✗	✓
quantum resilient	✗	✓	✓

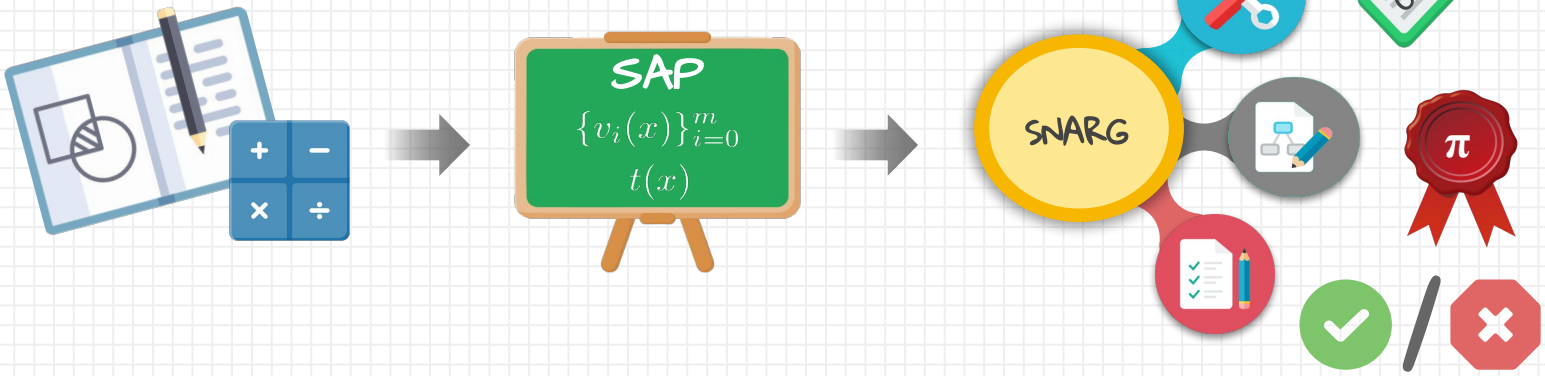
Framework intuition

SNARG
Background

Framework
for SNARGs

Construction
Security

The
END



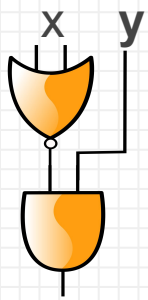
Computation: Circuit SAT

Verifier

Claim f(x)=y

NP statement

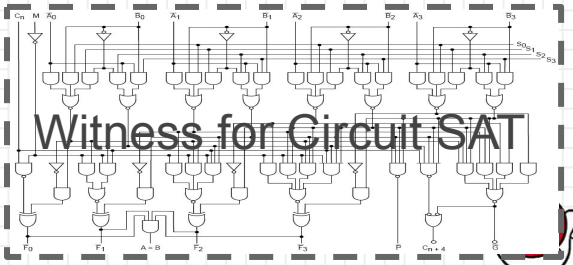
f(x)=y

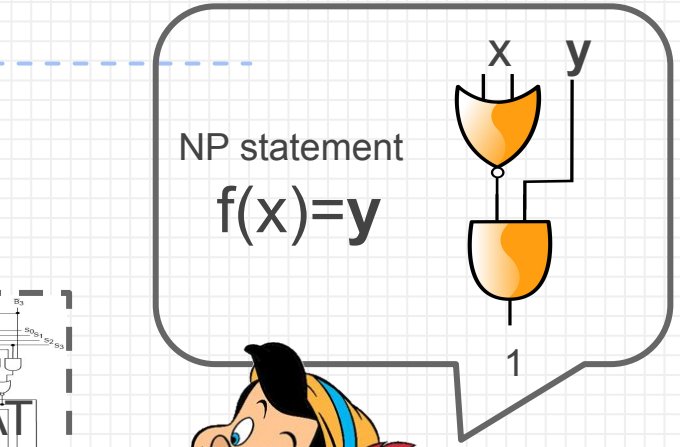


Prover

NP witness: Too long!

Verifier

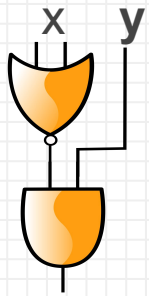


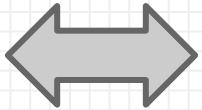


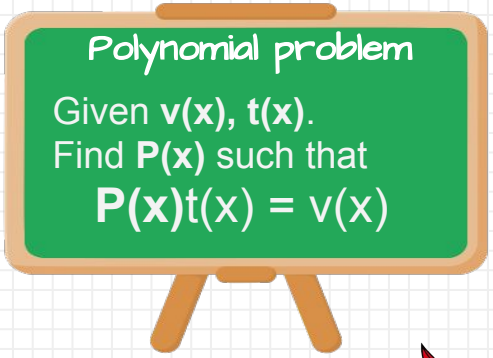
Prover

Solve equivalent problem instead

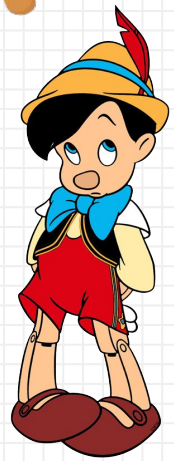
Circuit SAT solution







Verifier



Prover

Solve equivalent problem instead

Polynomial problem

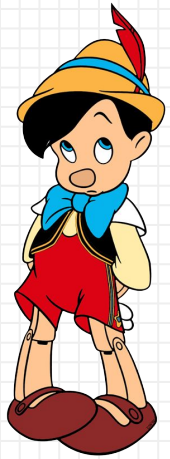
Given **v(x)**, **t(x)**.
Find **P(x)** such that
P(x)t(x) = v(x)

$$P(x) = \sum p_i x^i$$

Verifier

Coefficients of solution P(x)

$p_0, p_1, p_2, \ldots p_d$

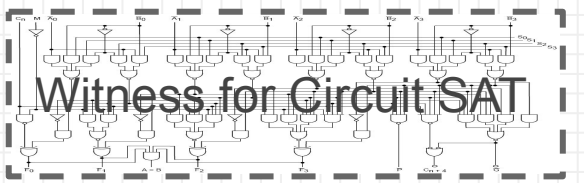


Prover

Solution as big as witness for Circuit SAT

Not Succinct

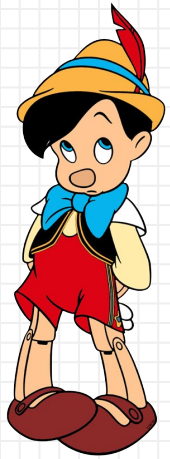
Verifier



Coefficients of solution P(x)

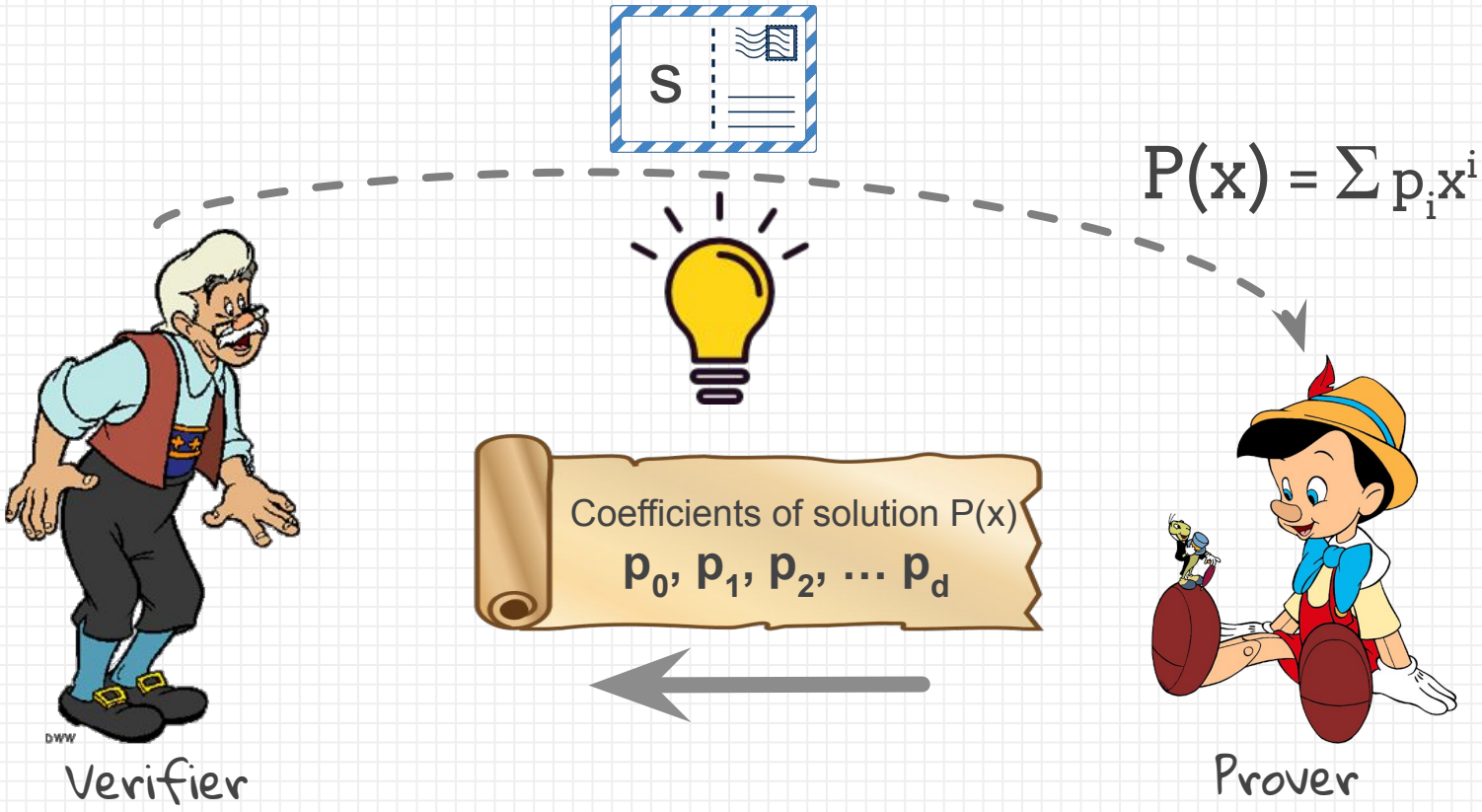
$p_0, p_1, p_2, \ldots p_d$

$$P(x) = \Sigma p_i x^i$$

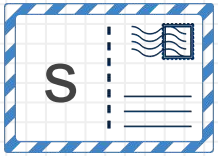


Prover

Evaluate polynomial in one point s



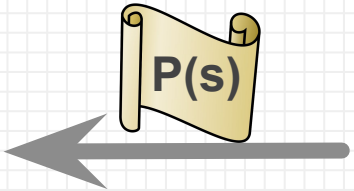
Evaluate polynomial in one point s

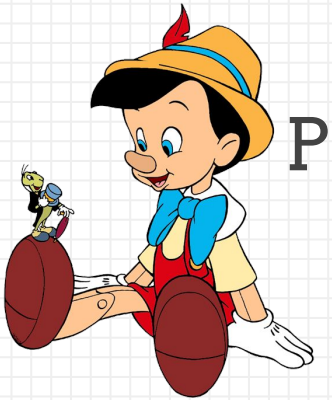


$$P(s) = \sum p_i s^i$$

P(x)t(x) = v(x) ⇒ **P(s)**t(s) = v(s)

Verifier

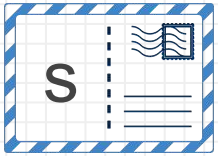




P(x)

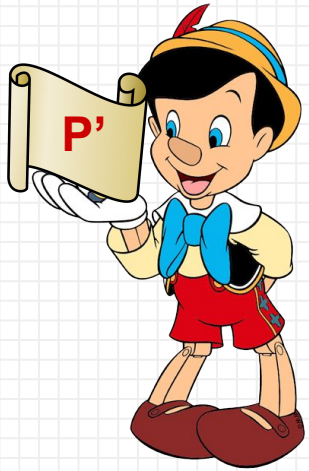
Prover

The evaluation point should be **hidden**



P' ≠ P(x)

Verifier

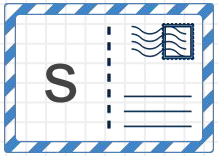


P(x)

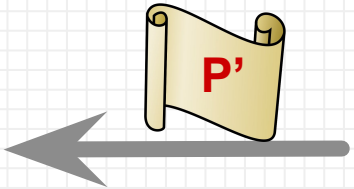
Prover

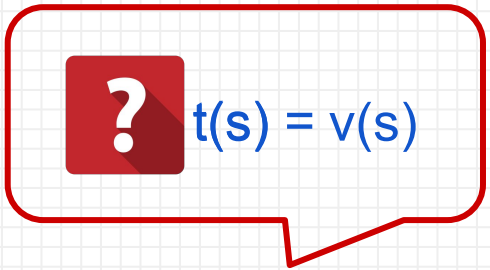
The evaluation point should be **hidden**

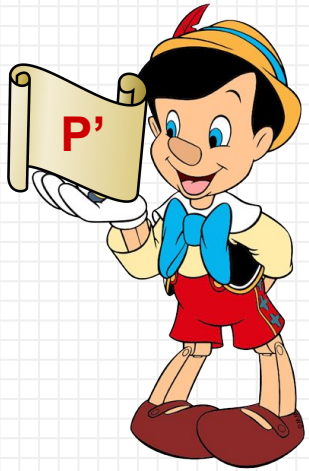
Verifier



Enc(**s**)



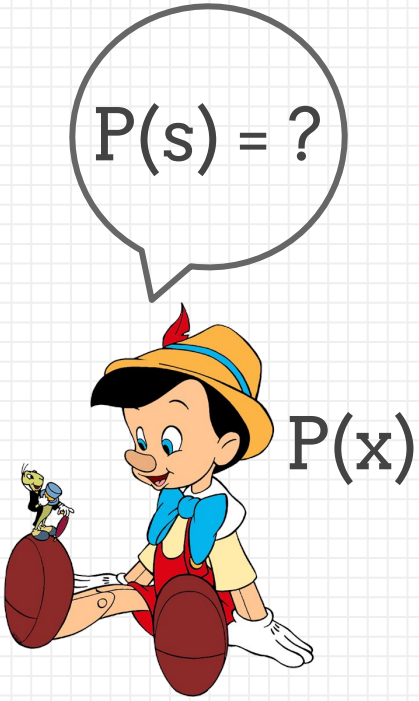




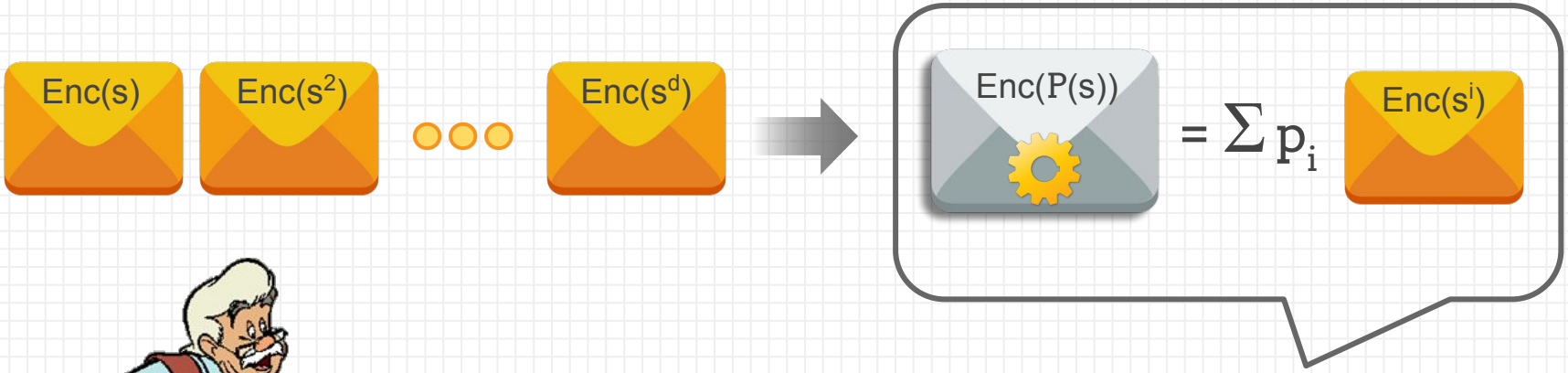
P(x)

Prover

Encoding of evaluation point s



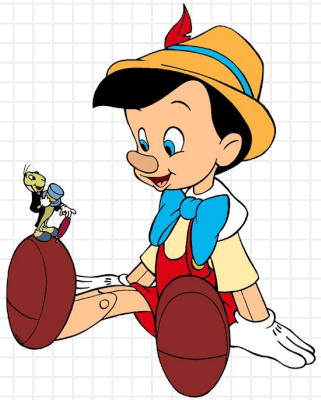
Encoding Properties



Verifier

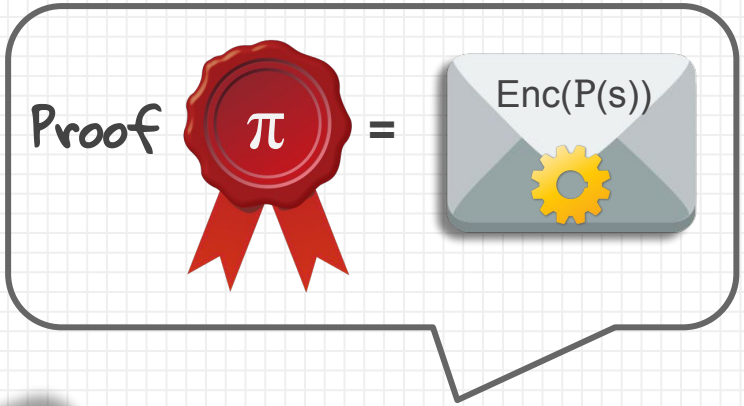
Encoding:

- ***linearly*** homomorphic



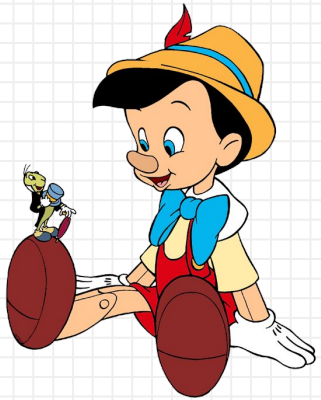
Prover

Succinct Proof



Verifier

Constant size Proof



Prover

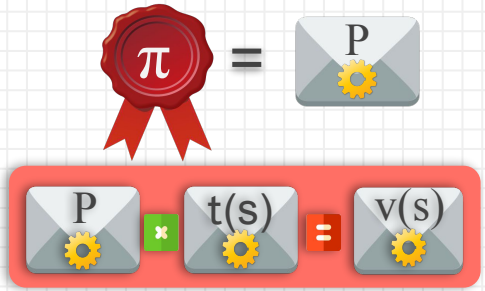
Verification

Polynomial problem

Given **v(x)**, **t(x)**.
Find **P(x)** such that

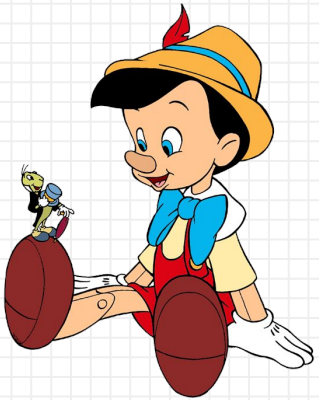
$$\mathbf{P(x)}t(x) = v(x)$$

Verifier



Encoding:

- *linearly* homomorphic
- quadratic root detection
- image verification



Prover

Security

Non-falsifiable Assumption: Linear-Only

L-O

$Enc(m_1)$ $Enc(m_2)$ ●●● $Enc(m_n)$

$Enc(M)$

$$M = a_1 m_1 + a_2 m_2 + \bullet\bullet\bullet + a_d m_d$$

Our SNARG

SNARG
Background

Definitions
for SNARGs

Construction
Security

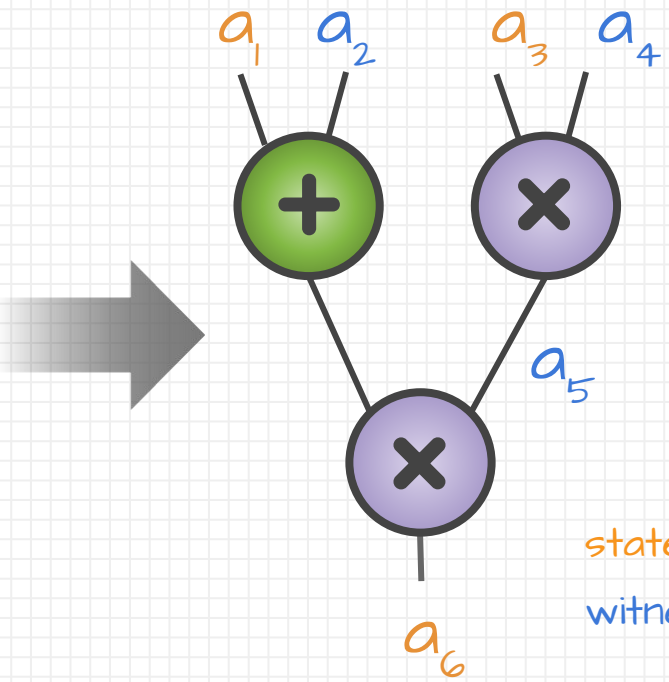
The
END

Square Arithmetic Programs

Computational Model
For Arithmetic Circuits

Arithmetic Circuit Satisfiability Problem

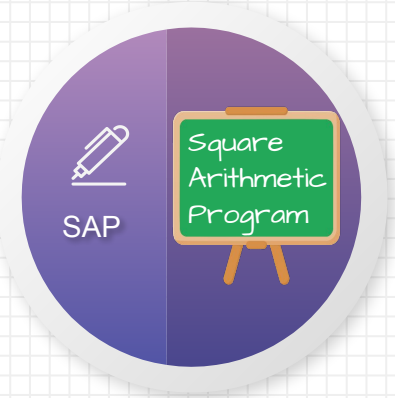
$f(a_1, a_3) = a_6$

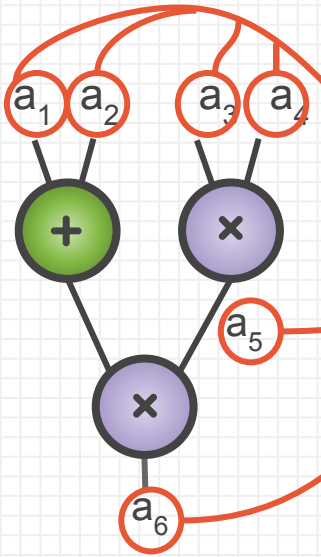


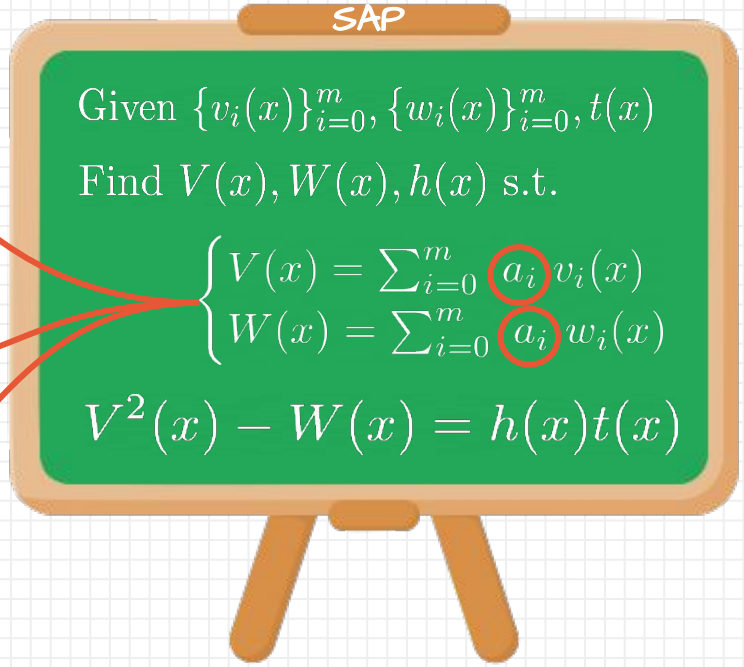
statement: a_1, a_3, a_6

witness: a_2, a_4, a_5

NEW Representation: Square Arithmetic Program







Encodings

Lattice-Based Assumptions

Encodings Instantiations: Discrete Log

DLog Group $\mathbb{G}$

$$Enc(s) = g^s$$

$$\langle g \rangle = \mathbb{G}, \langle \tilde{g} \rangle = \tilde{\mathbb{G}}$$

$$e : \mathbb{G} \times \mathbb{G} \to \tilde{\mathbb{G}}$$

$$e(g^a, g^b) = \tilde{g}^{ab}$$

Linearly homomorphic:

$$g^{\sum_i p_i s^i} = \prod (g^{s^i})^{p_i}$$

Quadratic root detection (**public**)

$$t(s)h(s) \stackrel{?}{=} p(s)$$

$$e(g^{t(s)}, g^{h(s)}) \stackrel{?}{=} e(g^{p(s)}, g)$$

Post-Quantum: Encryption Scheme

Encryption scheme

$$Enc(p(s)) = E_{pk}(p(s))$$

$$E_{pk}(m) = c$$

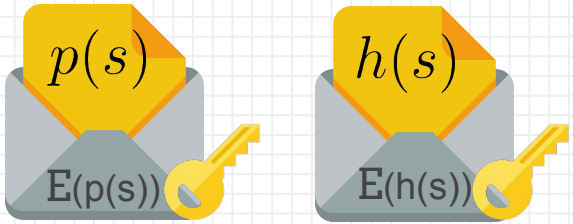
$$D_{sk}(c) = m$$

Linearly homomorphic:

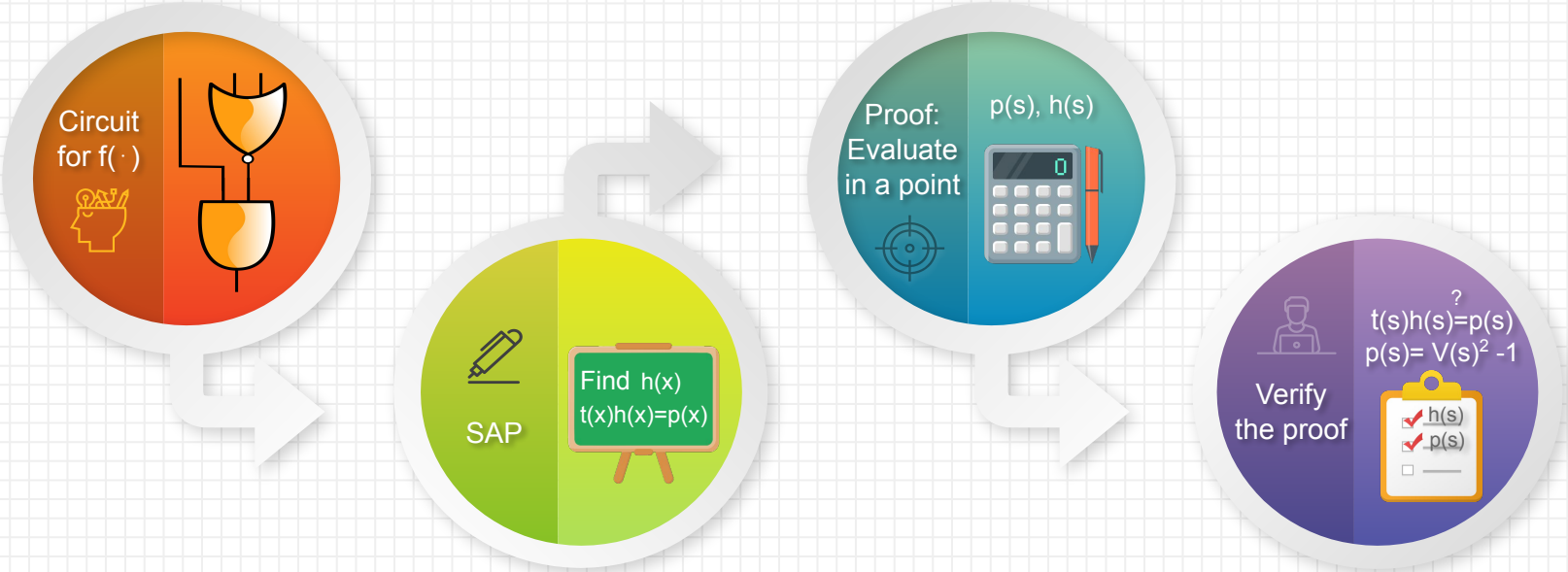
$$E_{pk}(\sum p_i s^i) = \sum p_i E_{pk}(s^i)$$

Quadratic root detection needs sk

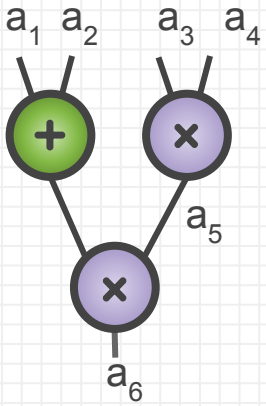
$$t(s)h(s) \stackrel{?}{=} p(s)$$



SNARK from SAP



Proof: Evaluate solution in s



Given $\{v_i(x)\}_{i=0}^m, \{w_i(x)\}_{i=0}^m, t(x)$

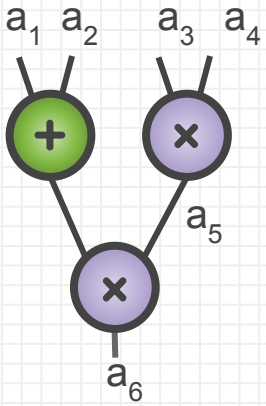
Find $V(x), W(x), h(x)$ s.t.

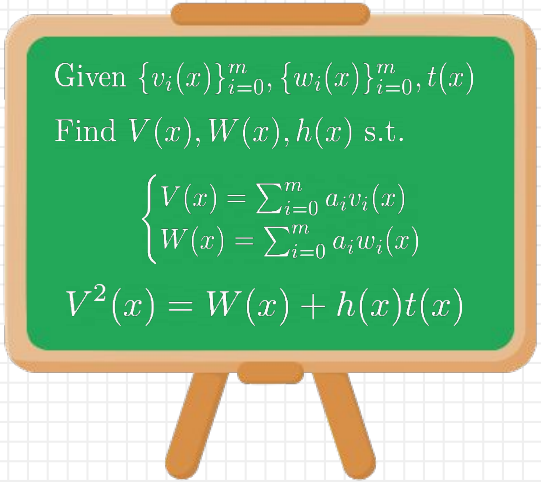
$$\begin{cases} V(x) = \sum_{i=0}^m a_i v_i(x) \\ W(x) = \sum_{i=0}^m a_i w_i(x) \end{cases}$$

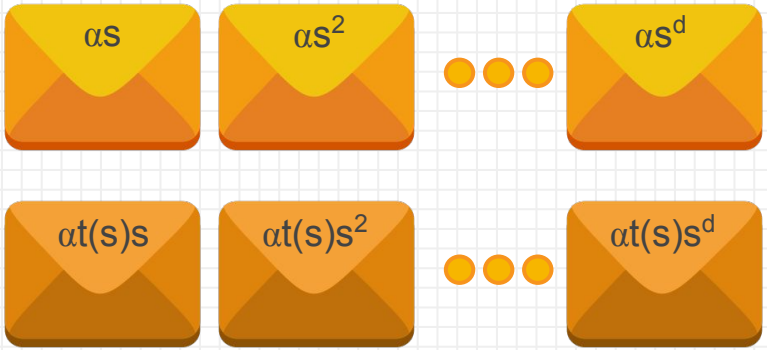
$$V^2(x) = W(x) + h(x)t(x)$$

= E(αV(s))

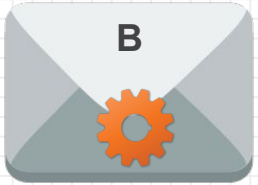
Proof: Division Term $A^2 = \alpha B$





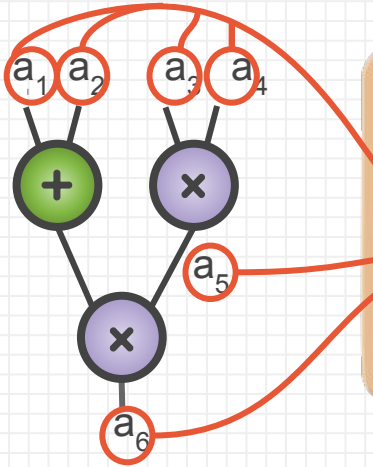


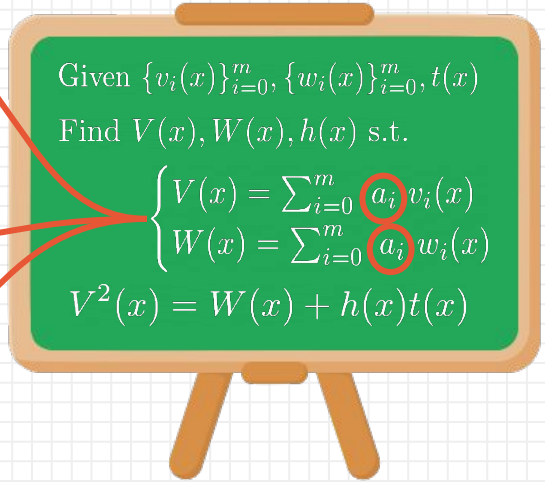
= E(αV(s))

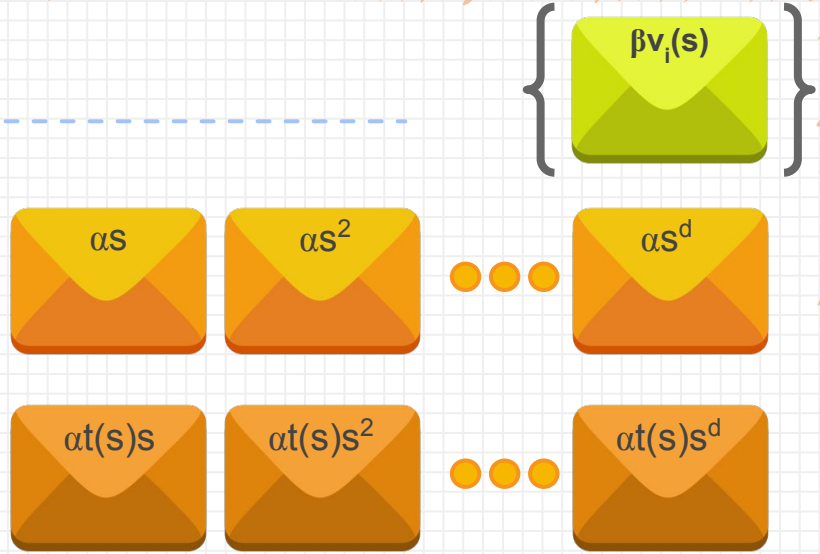


= E(αW(s) + αt(s)h(s))

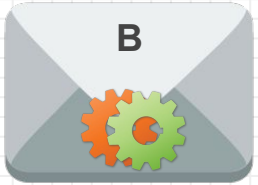
Proof: Linear Span





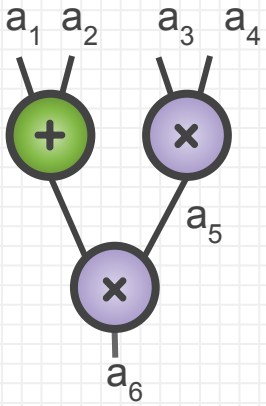


= E(αV(s))



= E(αW(s) + **βV(s)** + αt(s)h(s))

Proof: Same Span for V, W

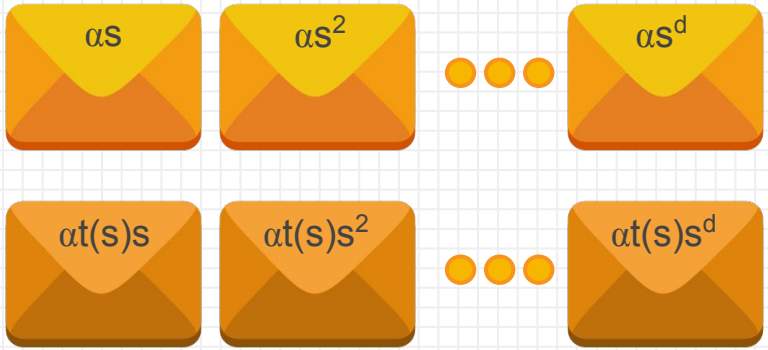


Given $\{v_i(x)\}_{i=0}^m, \{w_i(x)\}_{i=0}^m, t(x)$

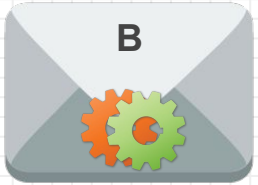
Find $V(x), W(x), h(x)$ s.t.

$$\begin{cases} V(x) = \sum_{i=0}^m a_i v_i(x) \\ W(x) = \sum_{i=0}^m a_i w_i(x) \end{cases}$$

$$V^2(x) = W(x) + h(x)t(x)$$

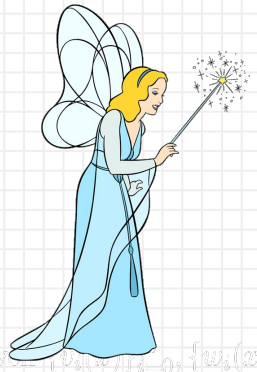


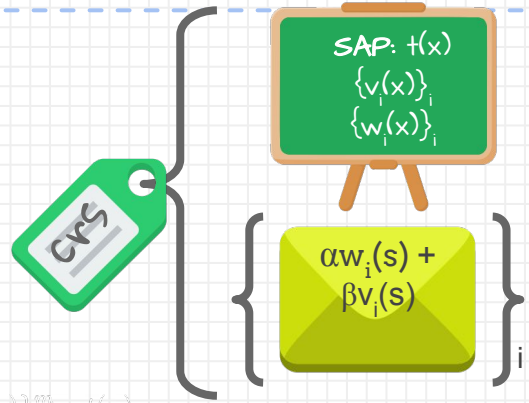
= E(αV(s))

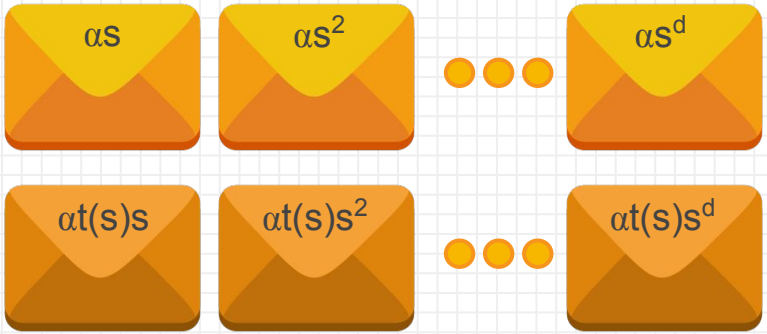


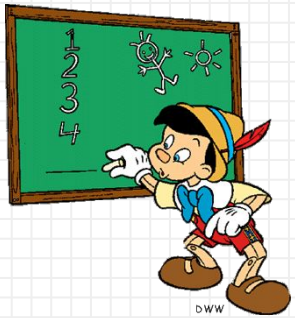
= E(α**W(s)** + β**V(s)** + α**t(s)h(s)**)

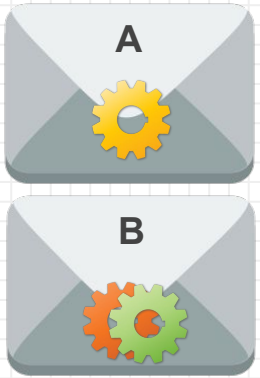
Protocol







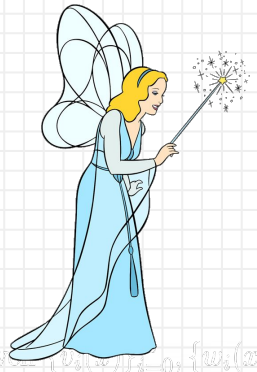


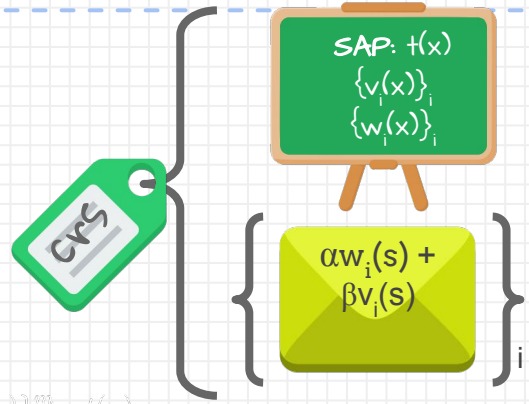


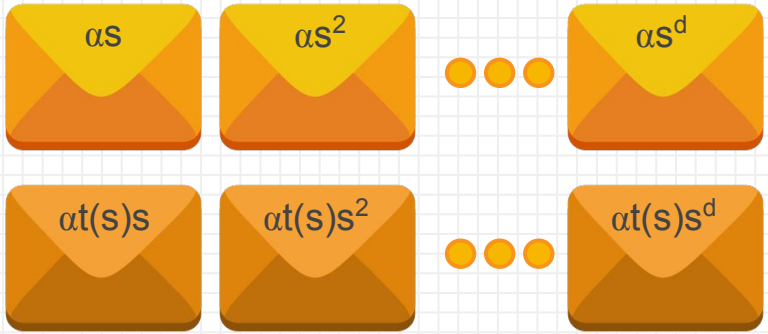
$= E(\alpha V(s))$

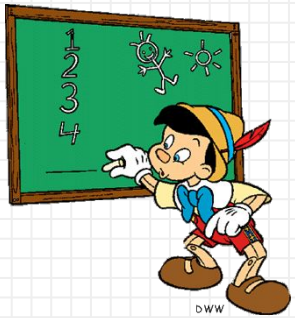
$= E(\alpha W(s) + \beta V(s) + \alpha t(s)h(s))$

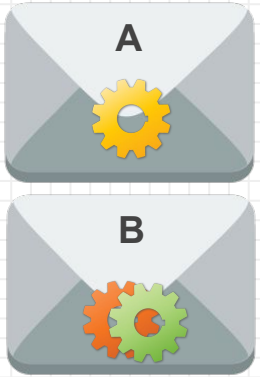
Setup and Proof











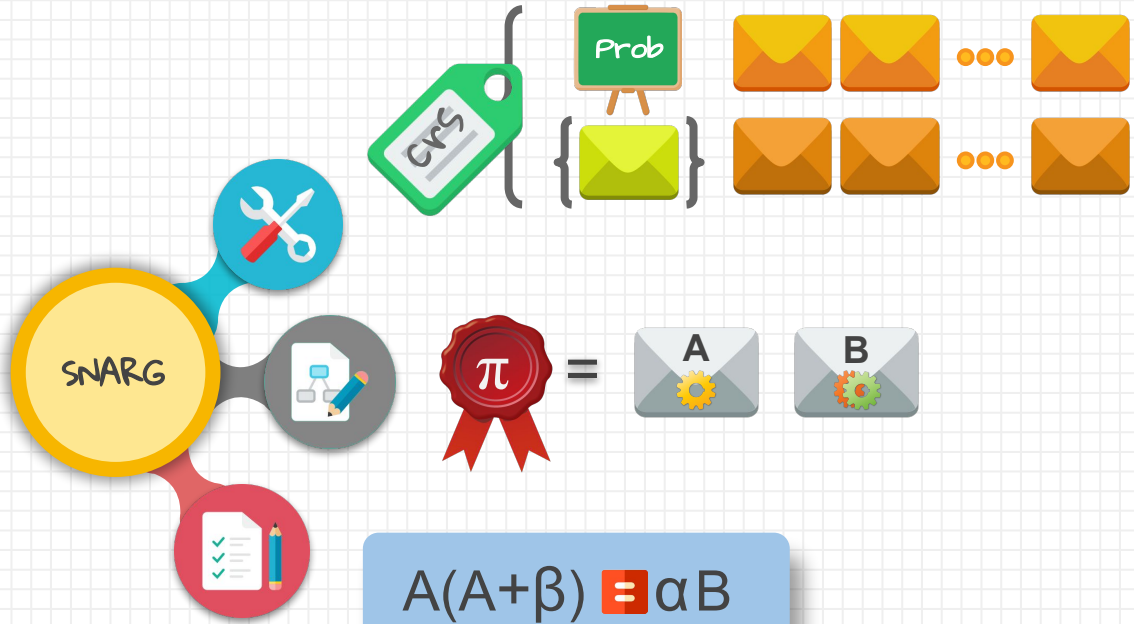
$$A = E(\alpha V(s))$$

$$B = E(\alpha W(s) + \beta V(s) + \alpha t(s) h(s))$$

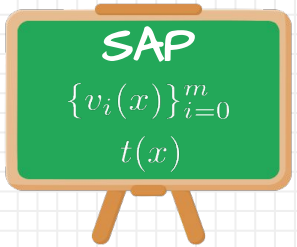
$$A(A+\beta) = \alpha B$$

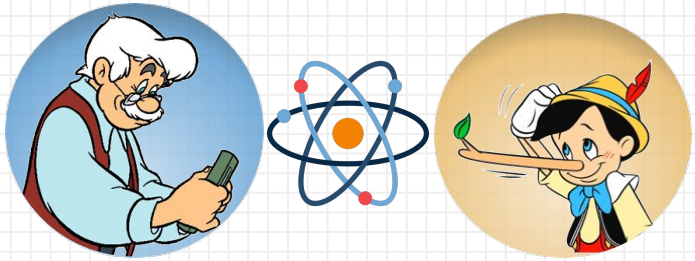
$$V^2(x) = W(x) + h(x)t(x)$$

Review of the Protocol (Algorithms)



zk-SNARG





Target Statement R(y,w)=1	Computational Model (Representation)	SNARG under Post-Quantum Assumptions
Computation y=F(x)	**PCP:** Probabilistically Checkable Proofs	(Strong) Vector Linear-Only Encryption
Boolean Circuit SAT	**QSP / SSP:** Quadratic / Square Span Programs	PKE on Lattice Encodings
Arithmetic Circuit SAT	**SAP:** Square Arithmetic Programs	Linear-Only Encodings

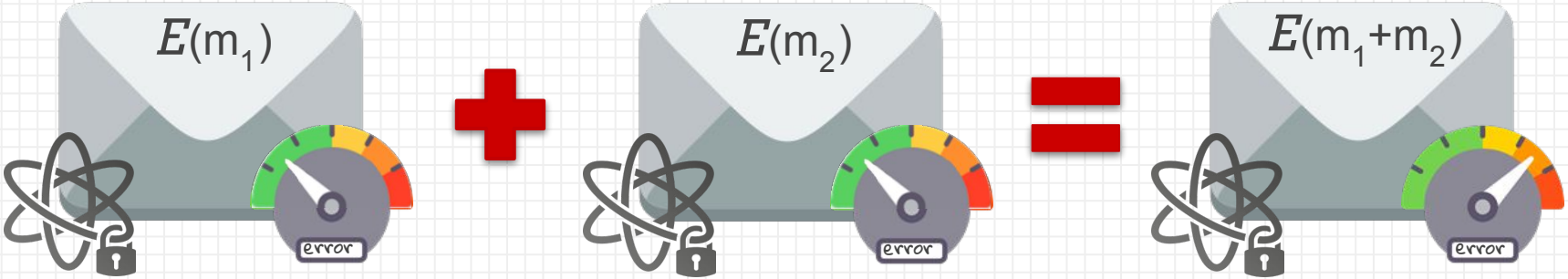
Post-Quantum: Lattice-Based Encryption Scheme

error

Encryption: $E_{\vec{s}}(m) = (-\vec{a}, \vec{a}\vec{s} + pe + m),\ e \in \chi$

Decryption: $D_{\vec{s}}((\vec{c_0}, c_1)) = \vec{c_0} \cdot \vec{s} + c_1 \pmod{p}$

$$E_{\vec{s}}(m_1) + E_{\vec{s}}(m_2) = (-\vec{a}_1 - \vec{a}_2, (\vec{a}_1 + \vec{a}_1)\vec{s} + p(e_1 + e_2) + m_1 + m_2)$$

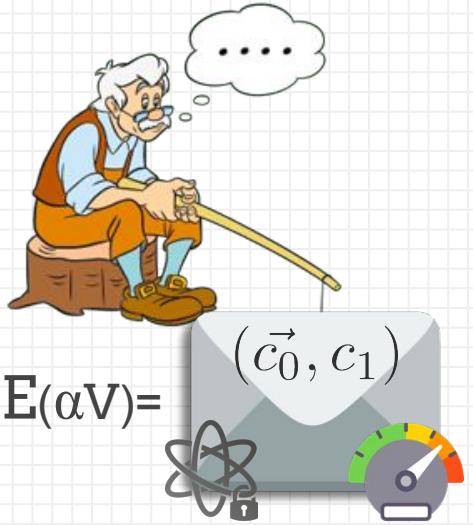


Challenge: Adding Zero-Knowledge

- ✘ randomize polynomials V(x), W(x) to hide witness

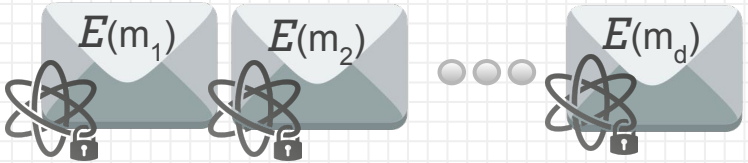
$$V(x) = v_0(x) + \sum_{i=1}^{m} a_i v_i(x) + \gamma t(x)$$

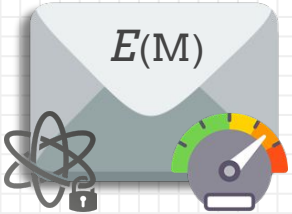
- ✘ add a **smudging term** to the noise of the encoding → distribution of the final noise independent of the coefficients a_i
- ✘ vector $\vec{c_0}$ is statistically indistinguishable from uniformly random from **leftover hash lemma**



Linear-Only Assumption [BISW17]

Linearly-Only
L-O



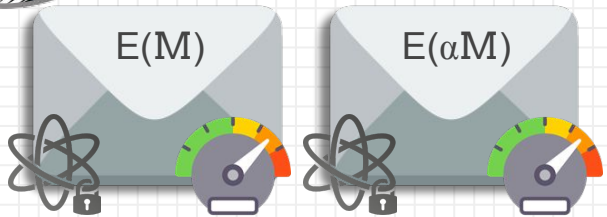


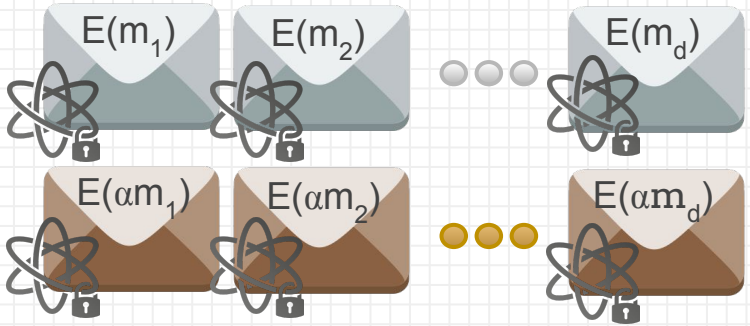
$= E(\Sigma a_i m_i)$

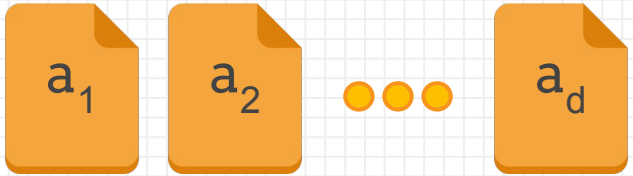
$M = a_1 m_1 + a_2 m_2 + \cdots + a_d m_d$

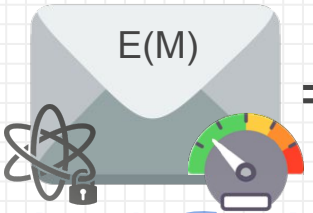
Extractable Linear-Only Assumption

Extractable L-O









$= E\left(\sum a_i m_i\right)$

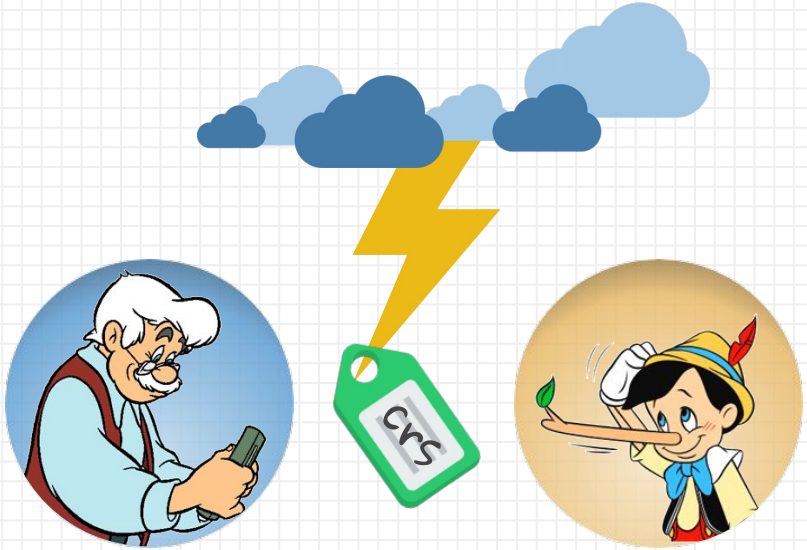
Conclusions

SNARK
Background

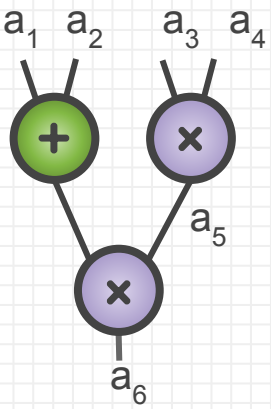
Framework
for SNARKs

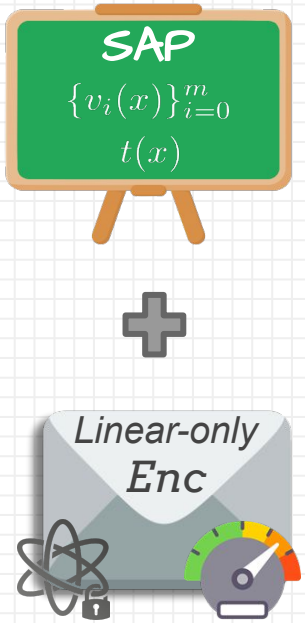
Construction
Security

The
END



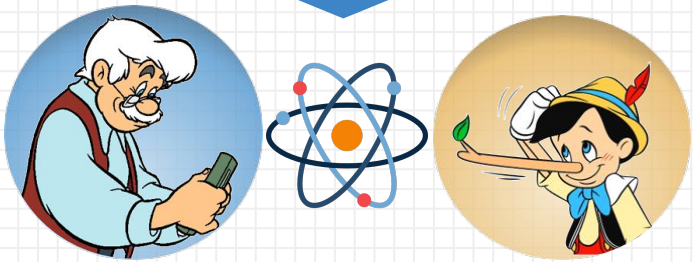
Review of Our Result





SNARG

Lattice-Based
zk-SNARG

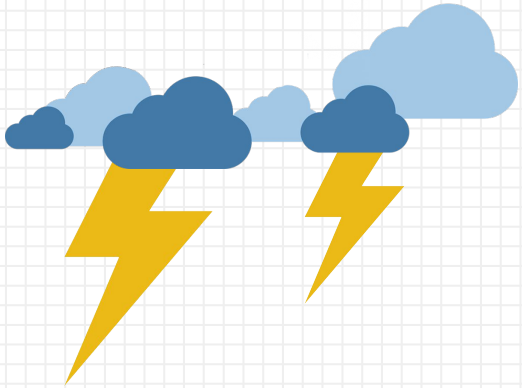


2 ciphertexts

zero-knowledge

designated-verifier

Further directions



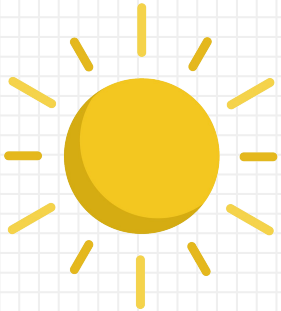
Pre-Processing:

(**crs**: common reference string)

- ✘ Secret coins
- ✘ Expensive
- ✘ Subversion

Designated Verifier:

- ✘ Secret Key **sk**



Subversion-Resistant Protocols

- ✘ Updatable **crs**
- ✘ Verifiable **crs**

Public Verification ?

Thank you

www.di.ens.fr/~nitulesc