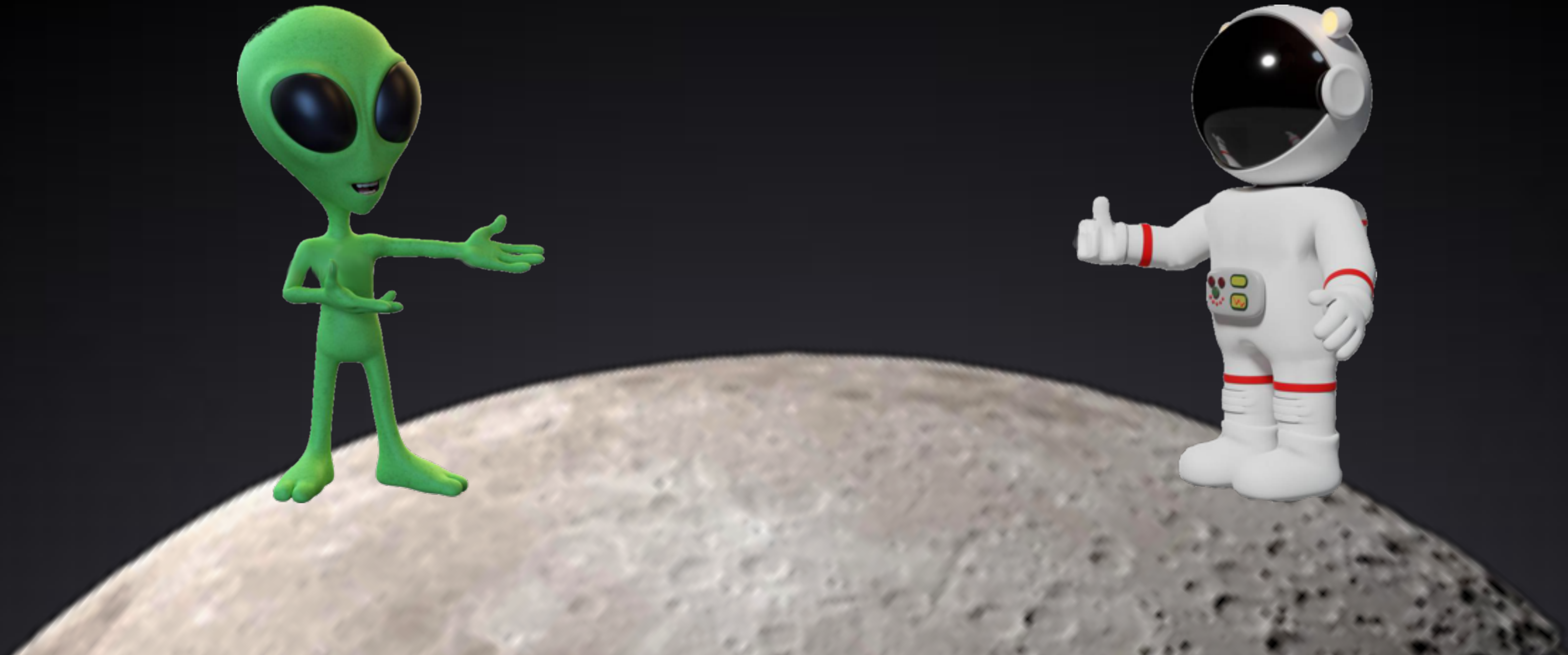
Lunar

a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions

SNARKs

Succinct Non Interactive ARguments of Knowledge



zero knowledge SNARKs

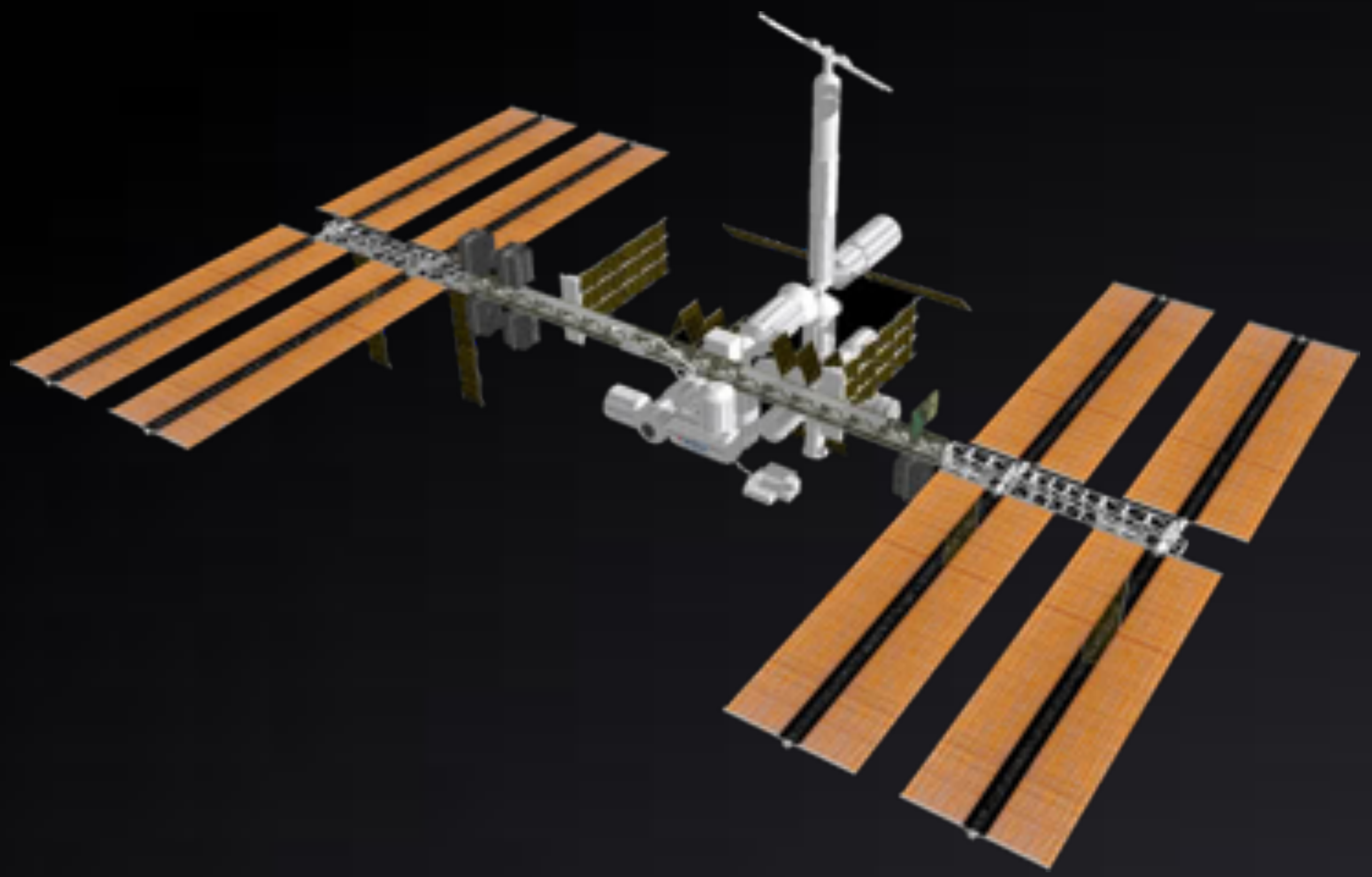
only learn claim is true

R (x , 👽) = 1

trusted setup zk SNARKs

third party creates keys for each relation

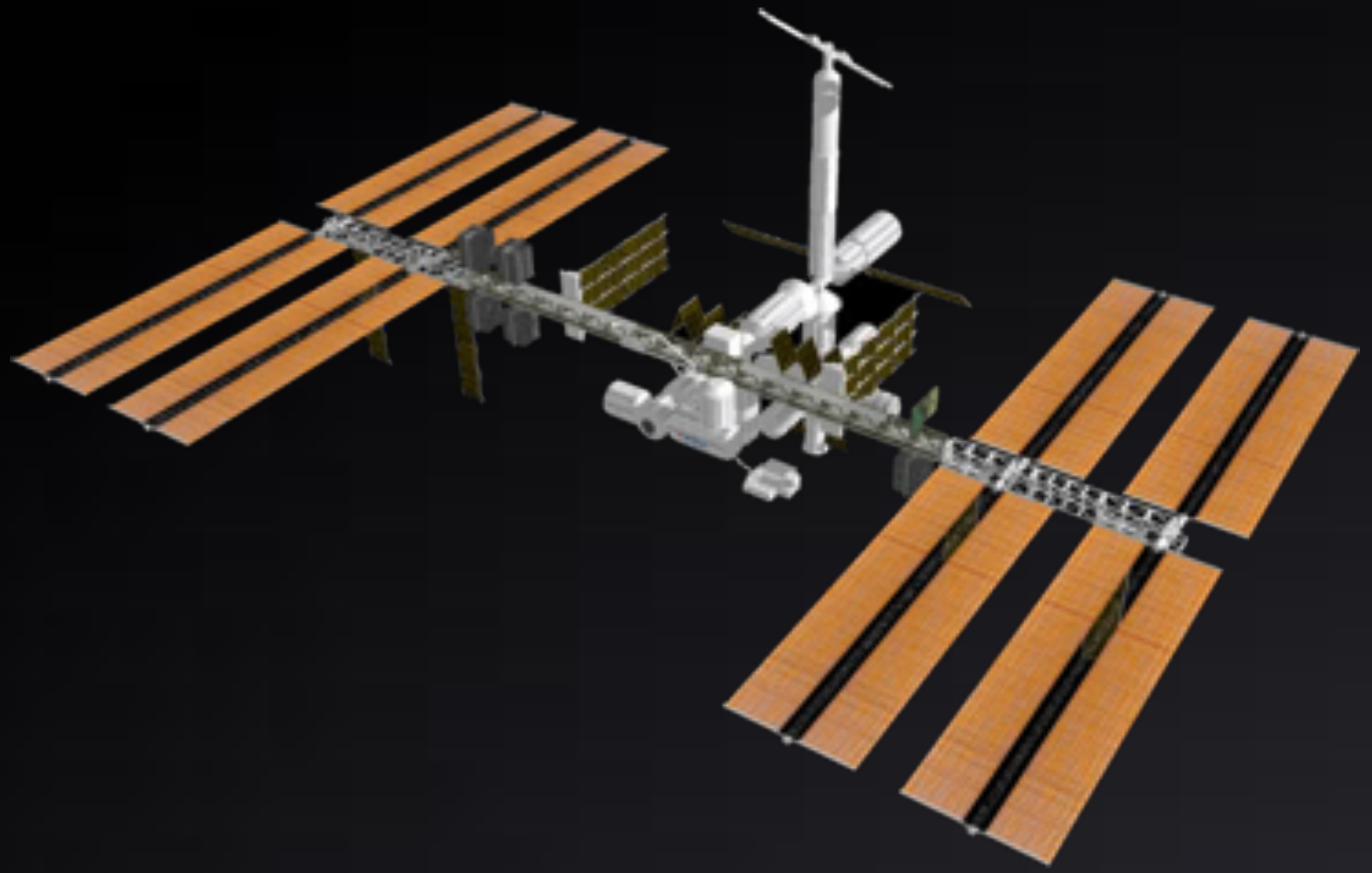
$$R(x, w) = 1$$

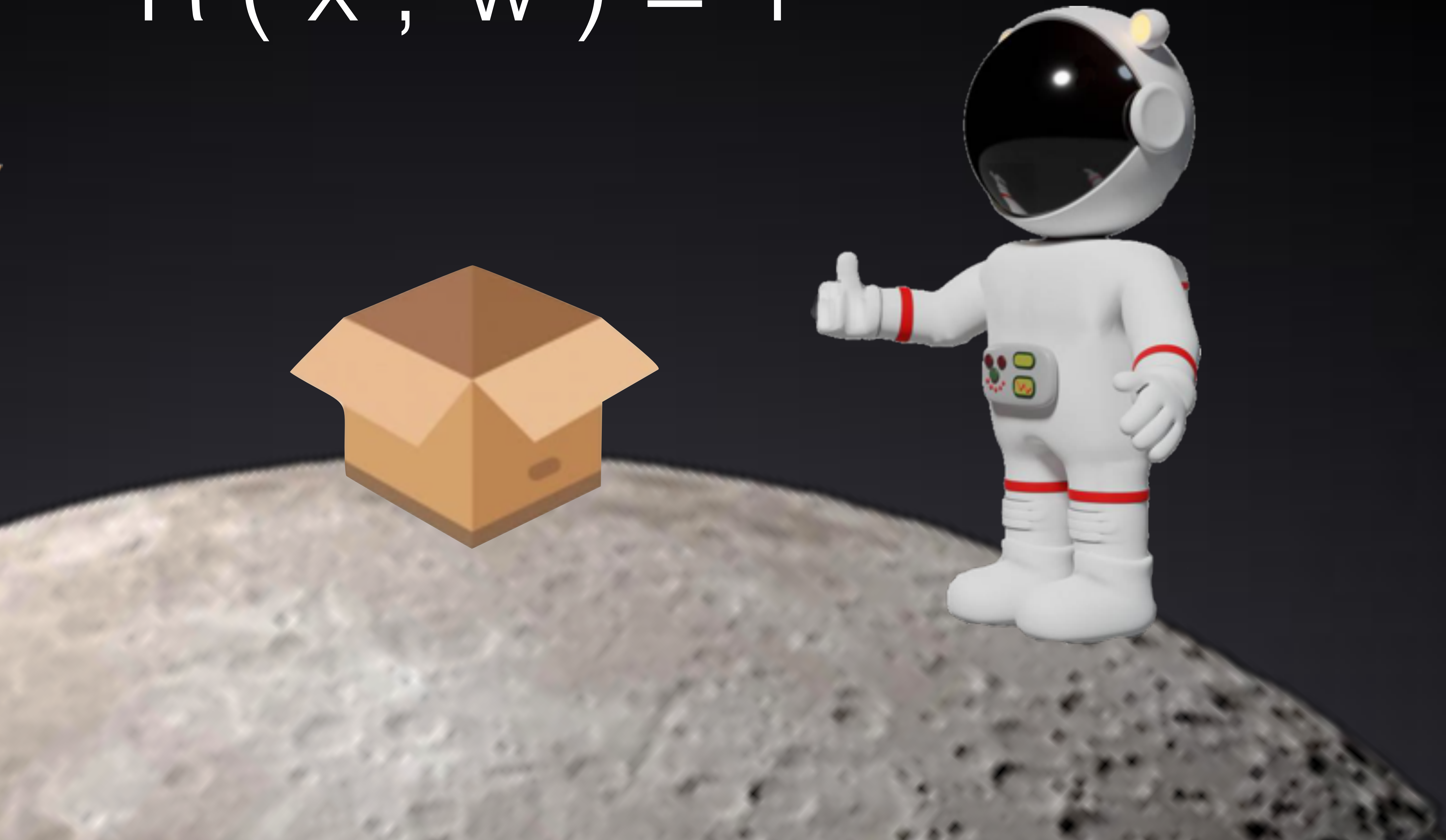


universal zk SNARKs

one time setup for any bounded relation

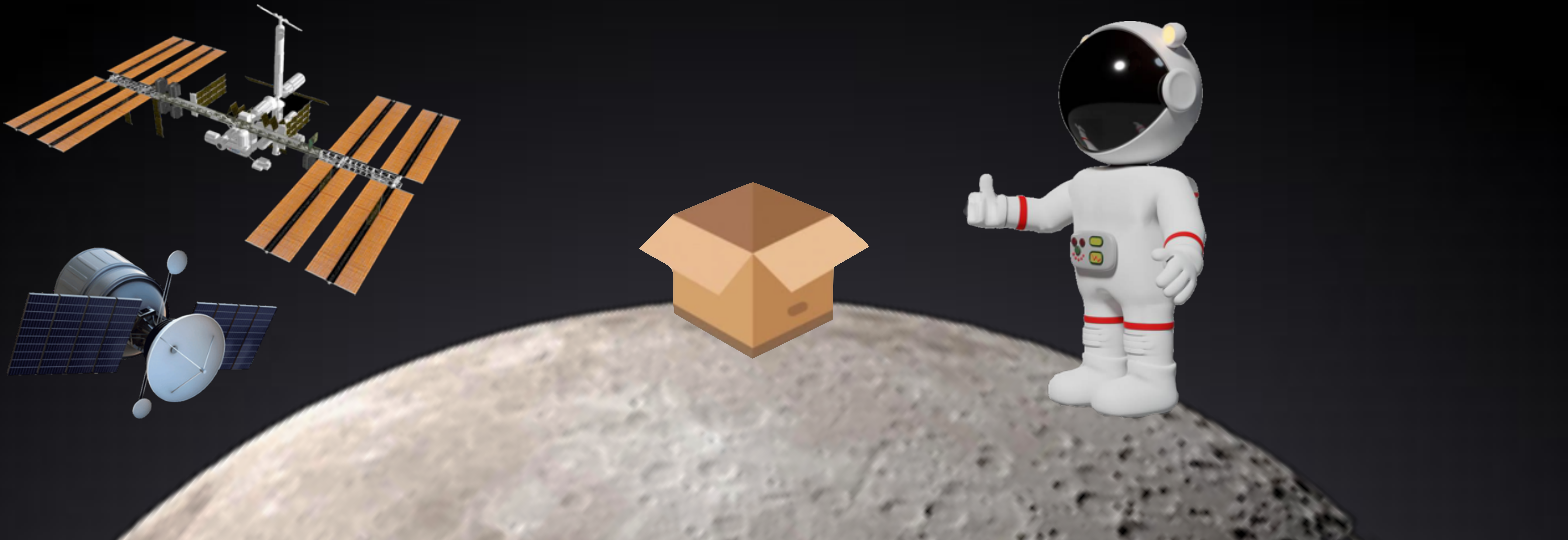
$$R(x, w) = 1$$





universal and updatable zk SNARKs

participate in the randomness



- first universal and updatable zkSNARK
- quadratic size SRS

'18

GKM+18

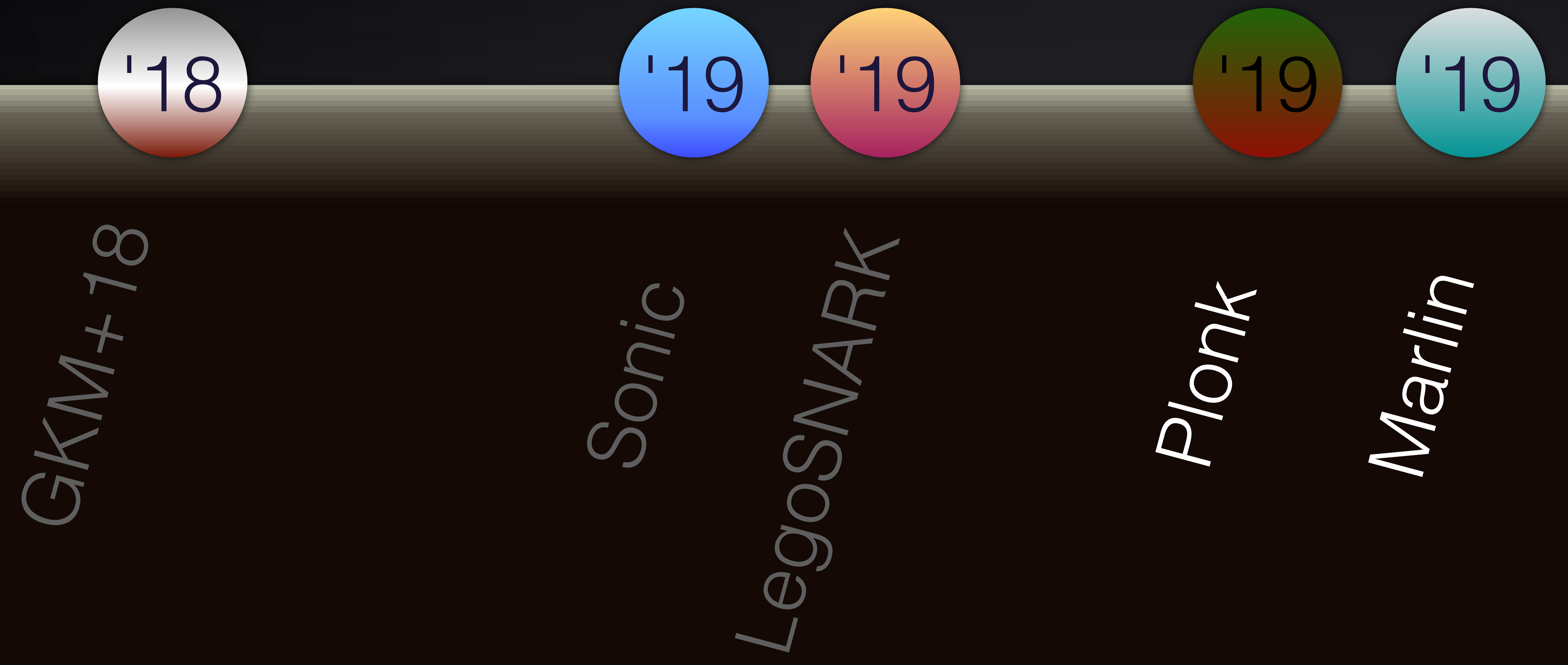
- first linear SRS universal and updatable zkSNARKs
- constant size proof and quasilinear prover
- polylogarithmic proof and linear prover

'18 GKM+18

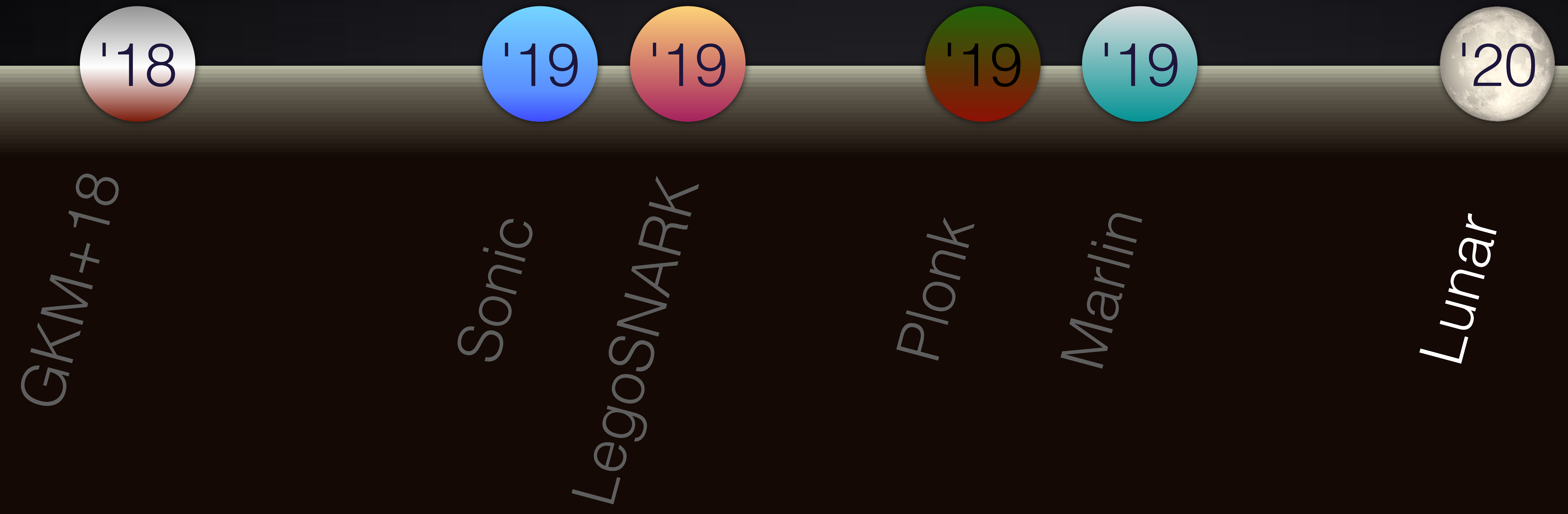
'19 Sonic

'19 LegoSNARK

- linear SRS universal and updatable zkSNARKs
- (shorter) constant size proof and (faster) quasilinear prover
- IOP-like + polynomial commitments

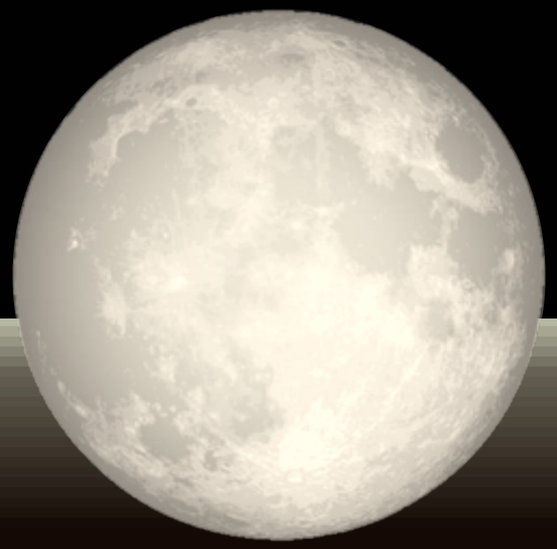


- family of linear SRS universal and updatable zkSNARKs
- more efficiency, shorter proofs, efficient CP variants
- more general IOP-like + CP-SNARKs



PLONK

MARLIN

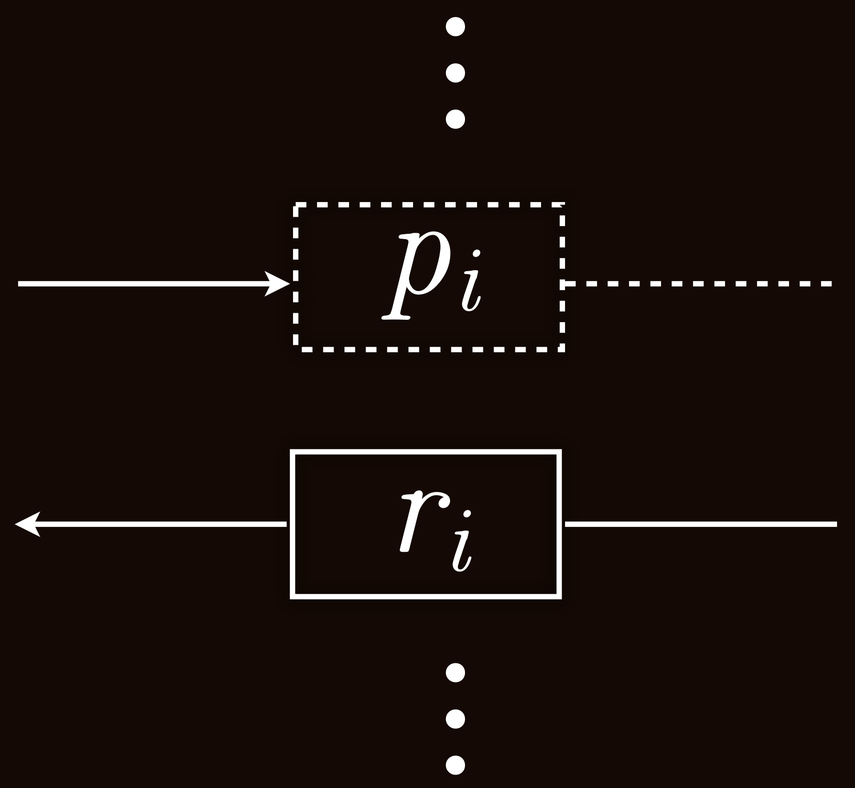


ILDP or AHP

+

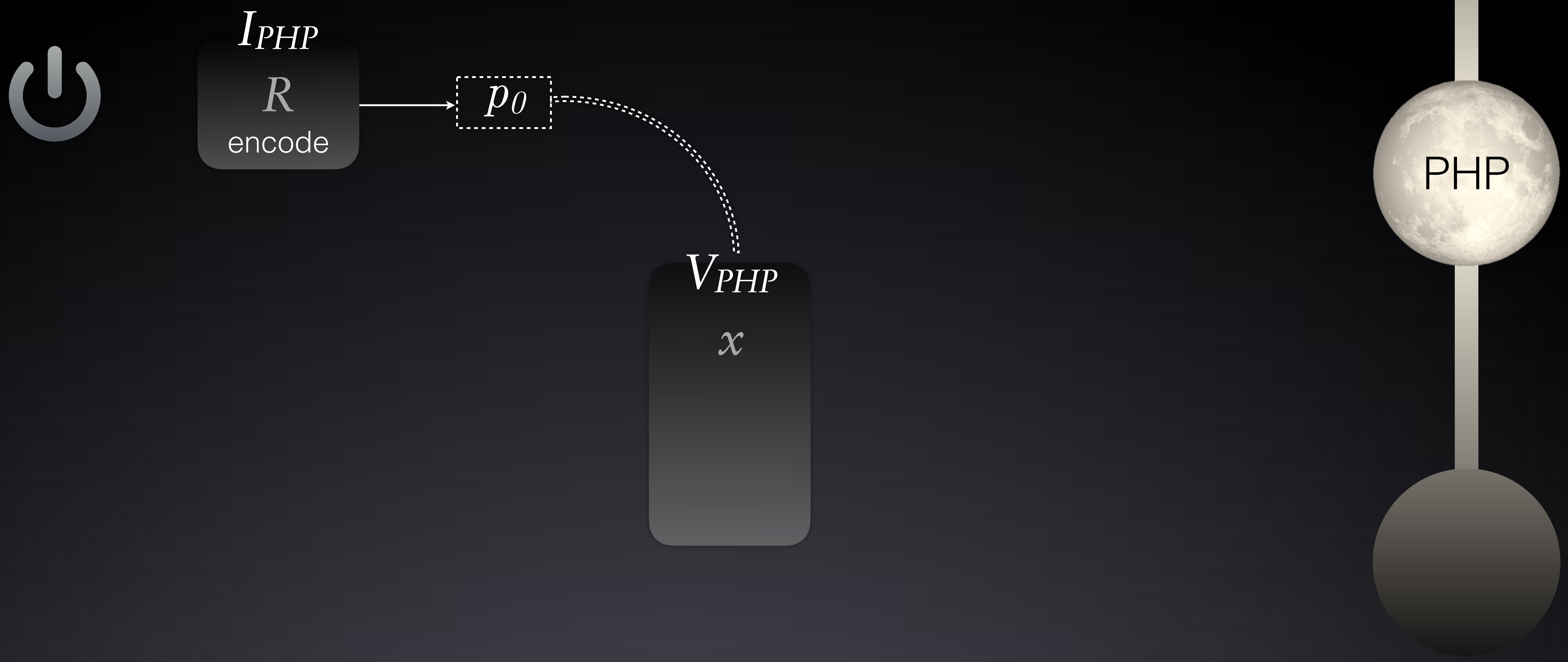
polynomial commitments

IOP-like information theoretic object

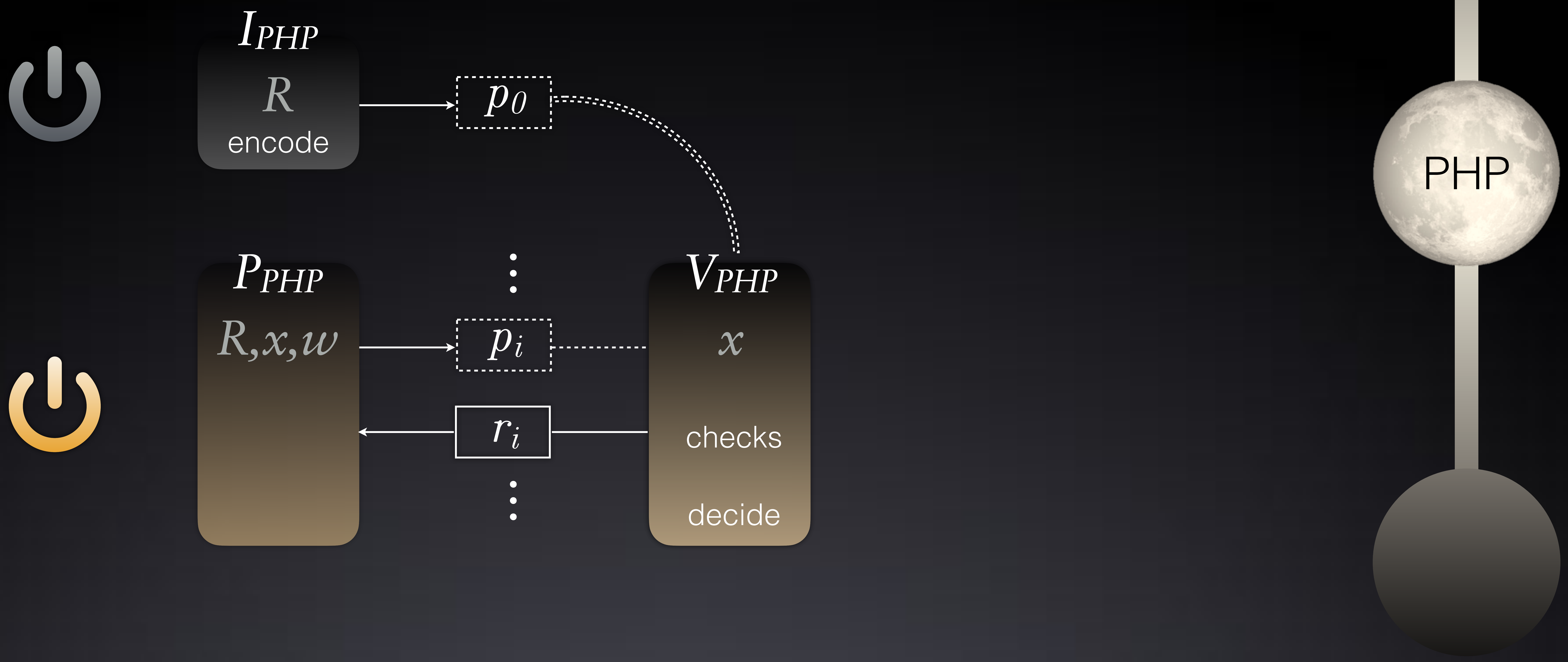


- point evaluation 1 F per polynomial
- lacks zero knowledge formalization
- optimizations deviate from abstraction

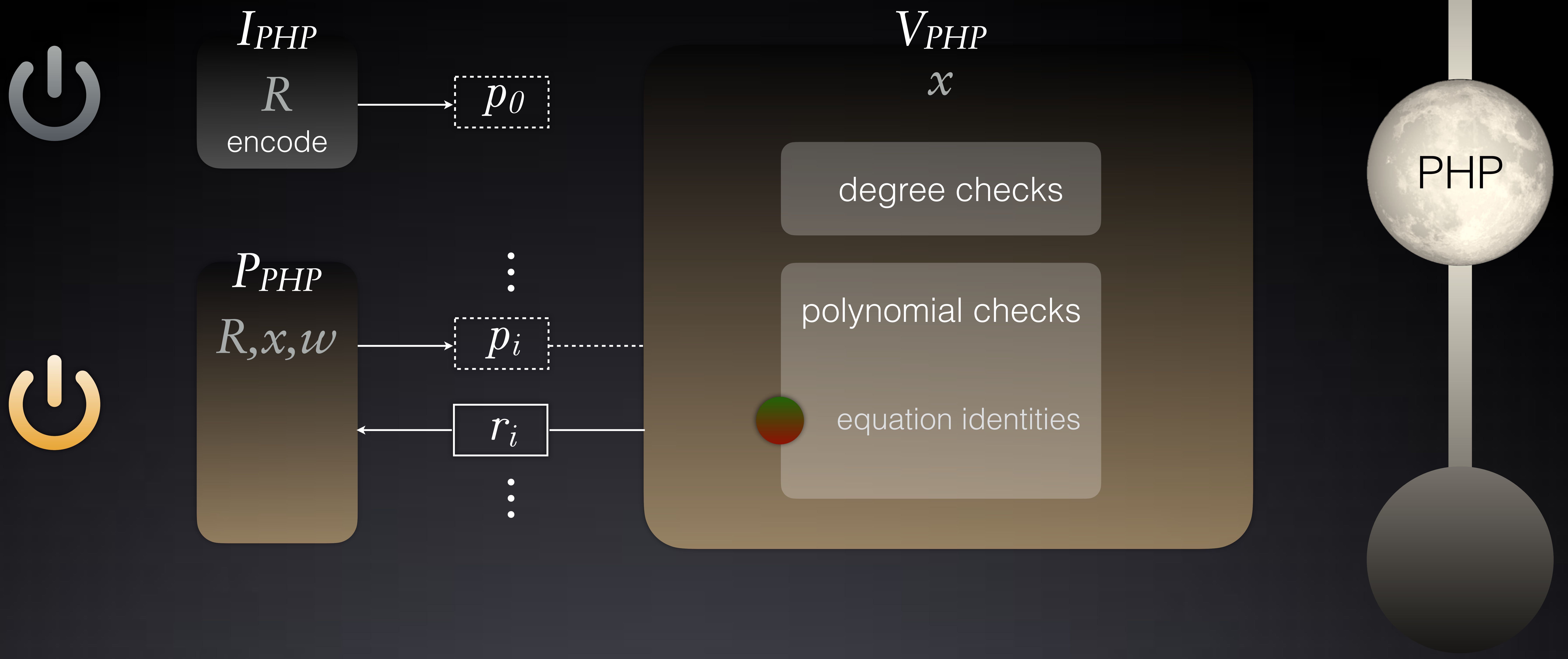
polynomial holographic proof



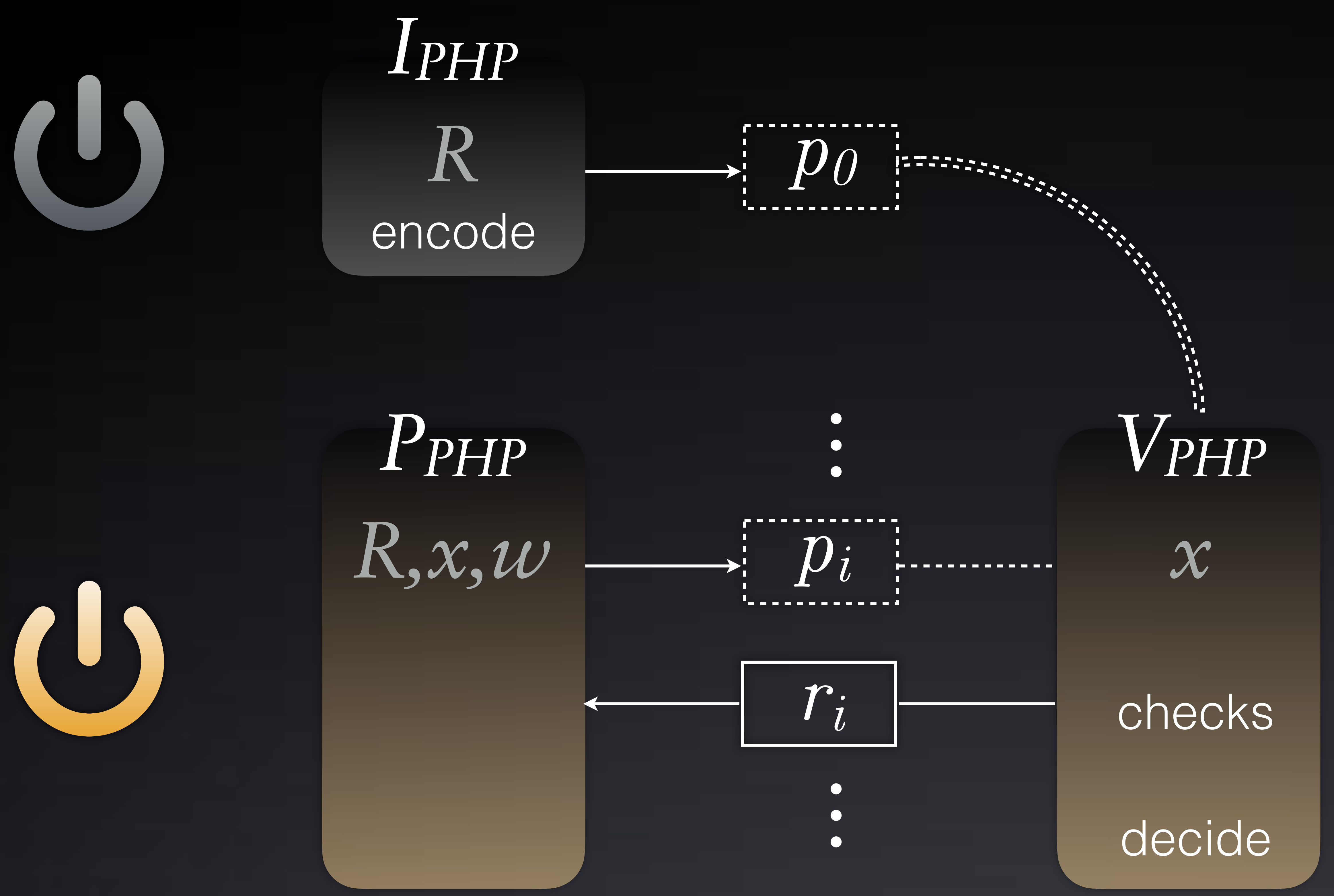
polynomial holographic proof



polynomial holographic proof



polynomial holographic proof



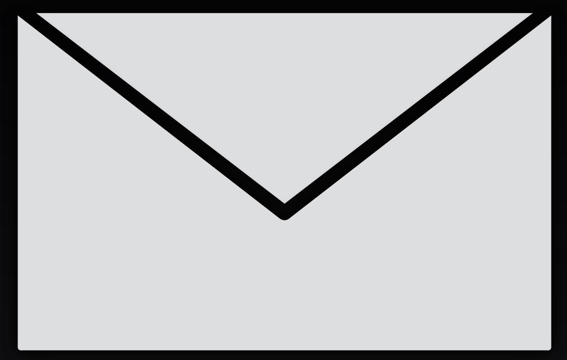
b – bounded ZK

PHP should still be ZK even after b_i evaluations of P_{PHP} oracle polynomial p_i

how?
increase degree of oracle polynomials

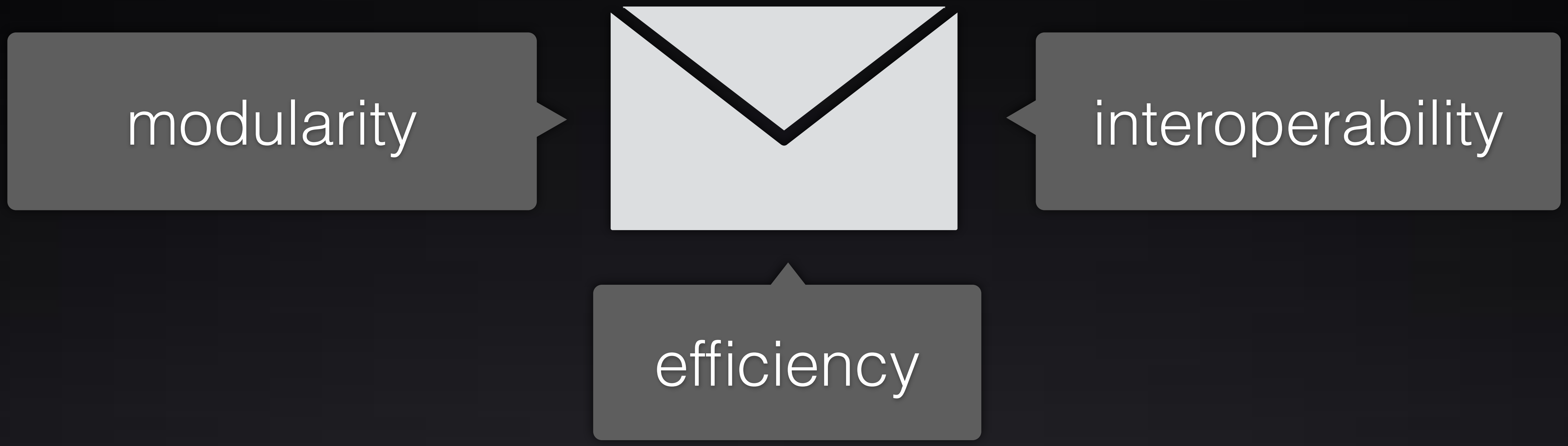
if $b_i = \infty$ then public

commit–and–prove zkSNARKs

R (x ,  , ) = 1 if

R (x , ) = 1 and 

commit–and–prove zkSNARKs



CS: type-based polynomial commitment scheme *in the exponent* `• rel` `• swh`

$\prod$.KeyGen(1^λ, N) → srs

CS.Setup(d) → ck monomials in exp

CP_{php}.KeyGen(ck) → ek_{php}, vk_{php}

CP_{opn}.KeyGen(ck) → ek_{opn}, vk_{opn}

$\prod$.Derive(ck, srs, R) → srs_R

CS.Commit(ck, I_{PHP} R)

$ek_R := ek \cup p_0, \; o_0$ CS₁ or CS₂

$vk_R := vk \cup$ p_0 $= [p_0(\$)]_{1 \vee 2}$

`• rel` non hiding commitments for relation polynomials

`• swh` somewhat hiding commitments for polynomials sent by the prover

1 — committed polynomials leak at most 1 evaluation at a random point, scheme can be deterministic

CS: type-based polynomial commitment scheme *in the exponent* • rel • swh

$\prod$.KeyGen(1^λ, N) → srs

CS.Setup(d) → ck monomials in exp

CP_{php}.KeyGen(ck) → ek_{php}, vk_{php}

CP_{opn}.KeyGen(ck) → ek_{opn}, vk_{opn}

$\prod$.Derive(ck, srs, R) → srs_R

CS.Commit(ck, I_{PHP} R)

ek_R := ek ∪ p_0, o_0 CS₁ or CS₂

vk_R := vk ∪ p_0 = $[p_0(\$)]_{1 \vee 2}$

$\prod$.Prove(ek_R,x,w) → π

i CS.Commit(ck, P_{PHP} i,ρ) FS

CP_{opn}.Prove(ek_{opn}, p_i , o_i)

π = ({ p_i , m_i , π_{opn_i} }, π_{php})

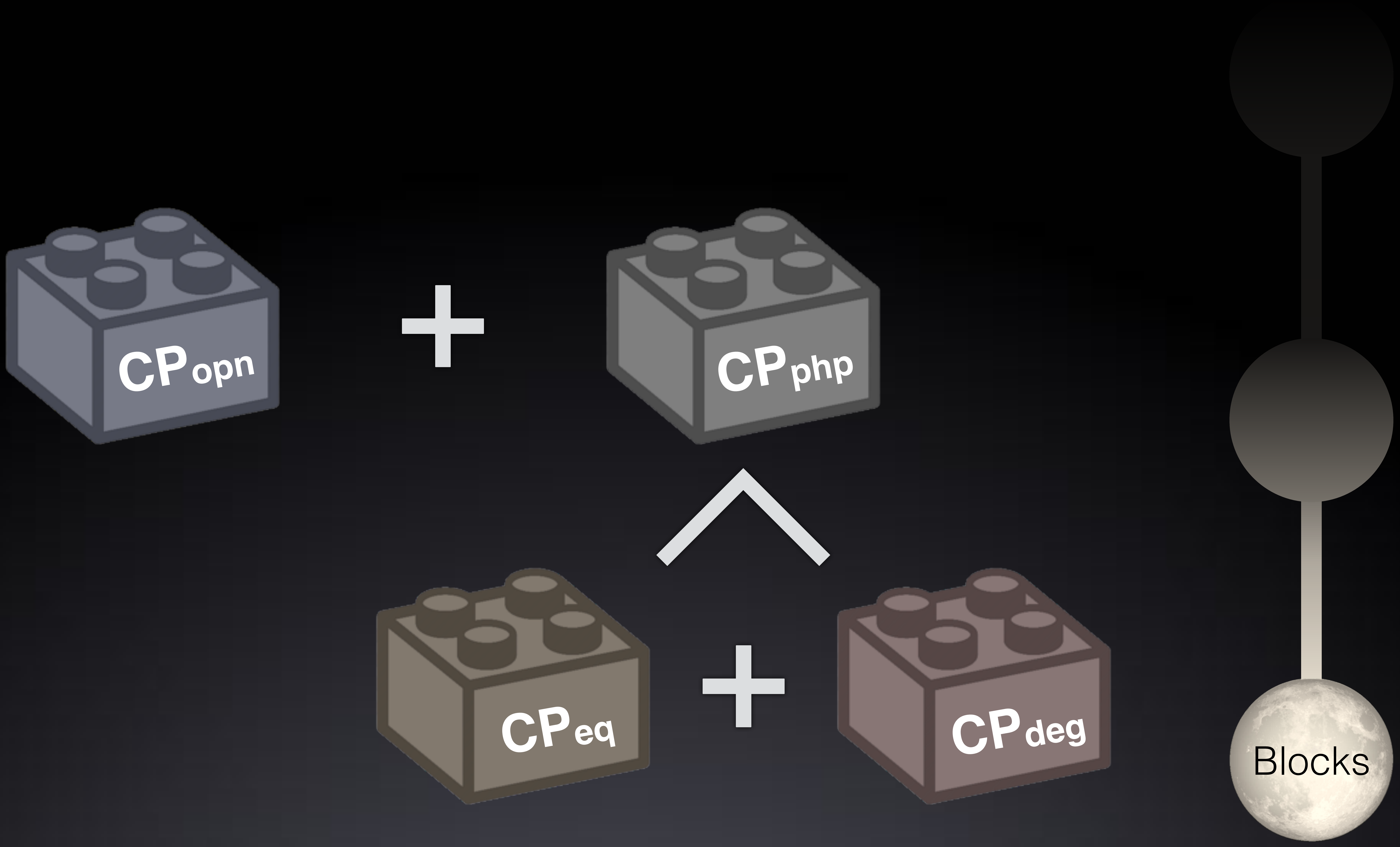
proof that a V_{PHP} would accept

$\prod$.Verify(vk_R,x,π) → ok / ko

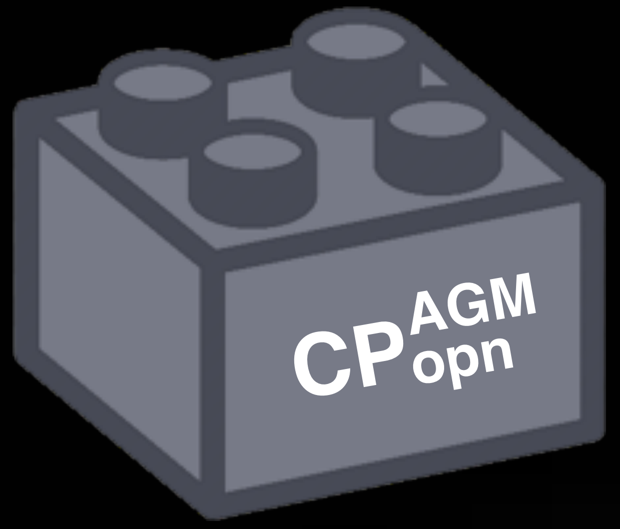
checks

CP_{php}.Verify(vk_{php}, *deg*, *eqs*, p_0 p_1 … p_r , π_{php})

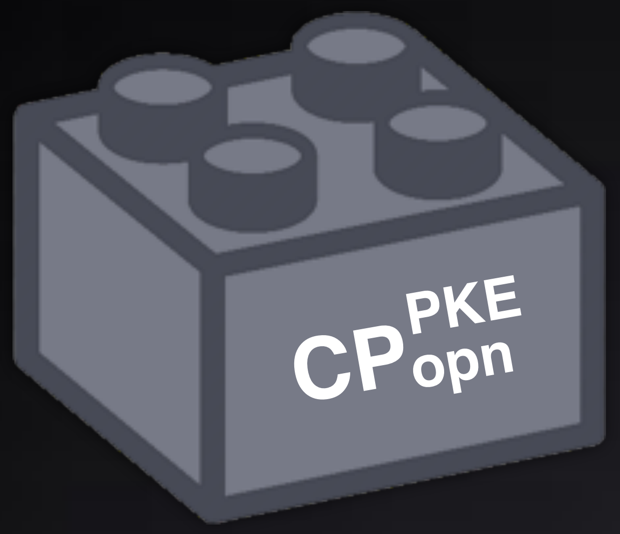
CP_{opn}.Verify(vk_{opn} , p_i , π_{opn_i})



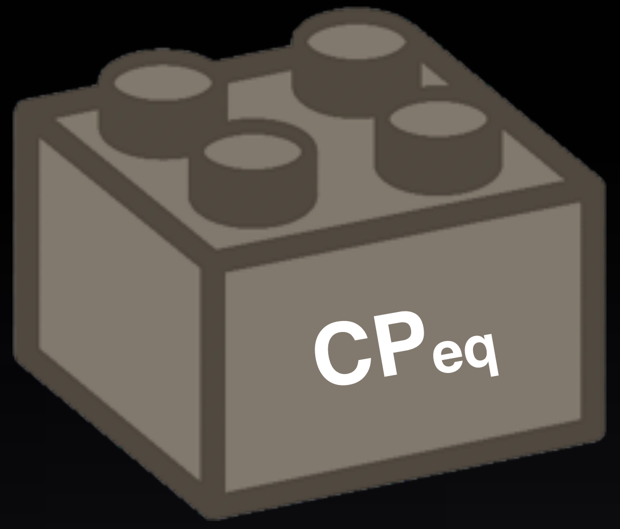
CP_{opn}
+
CP_{php}
$\wedge$
CP_{eq}
+
CP_{deg}
Blocks



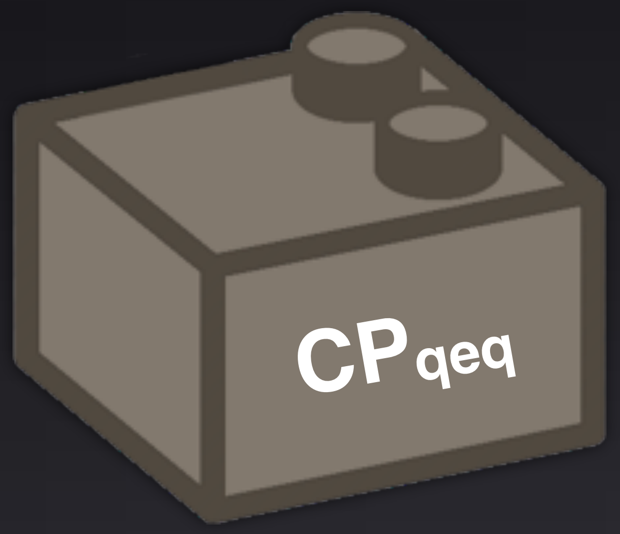
trivial empty proof
Marlin, Plonk

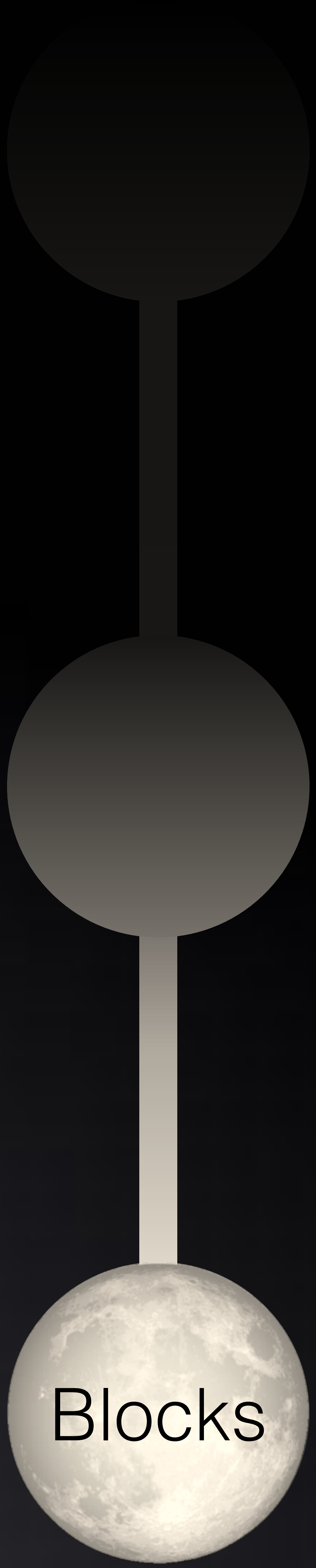


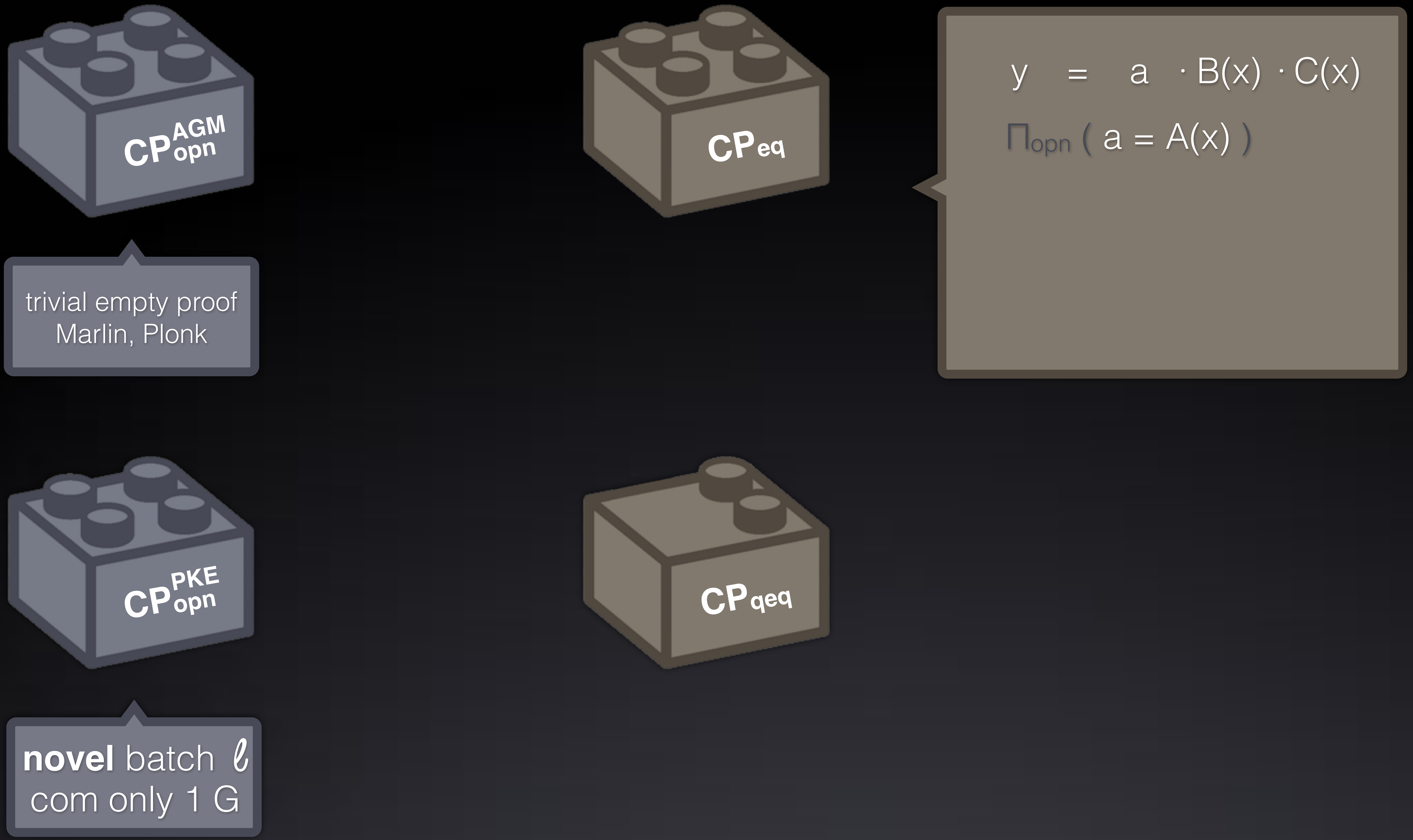
novel batch ℓ
com only 1 G

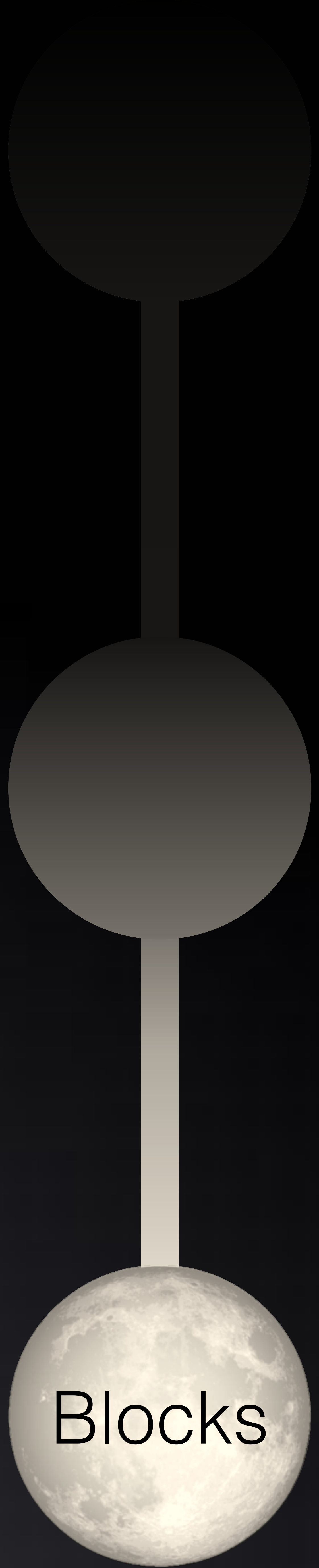


$D(X) = A(X) \cdot B(X) \cdot C(X)$






CP_{opn}^{AGM}
trivial empty proof
Marlin, Plonk
CP_{opn}^{PKE}
novel batch ℓ
com only 1 G
CP_{eq}
$y = a \cdot B(x) \cdot C(x)$
Π_{opn} ($a = A(x)$)
CP_{qeq}


Blocks

CP_{opn}^{AGM}

trivial empty proof
Marlin, Plonk

CP_{opn}^{PKE}

novel batch ℓ
com only 1 G

CP_{eq}

$y = a \cdot b \cdot C(x)$

Π_{opn} ($a = A(x)$)

Π_{opn} ($b = B(x)$)

Π_{eval} ($y = a \cdot b \cdot C(x)$)

$(b_1 \ldots b_p)$-leaky ZK

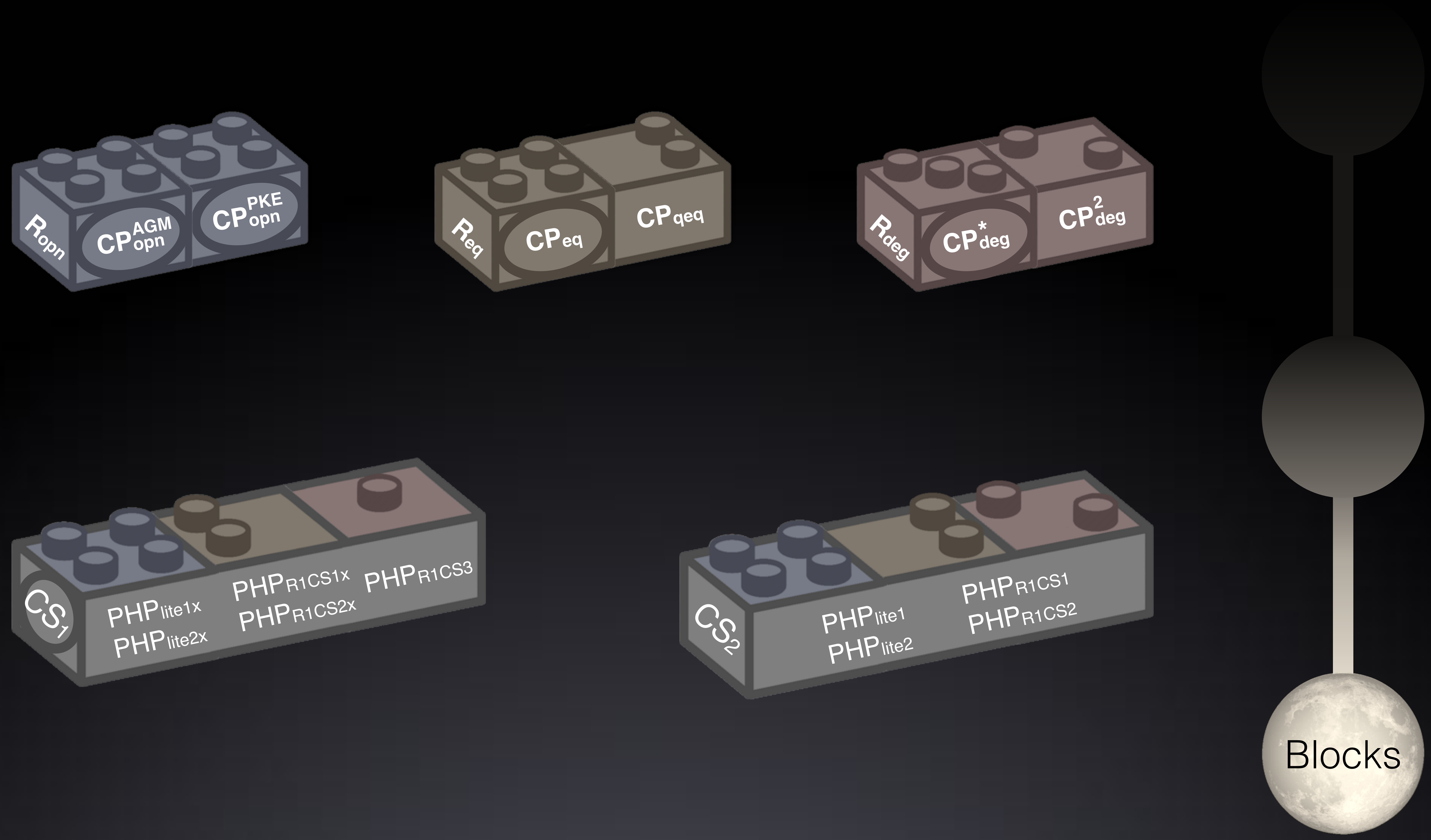
CP_{qeq}

quadratic equation
$C(X) = A(X) \cdot B(X)$
has empty proof if one
polynomial (relation)
is committed in G_2

$$e(A_1, B_2) = e(C_1, [1]_2)$$

Blocks

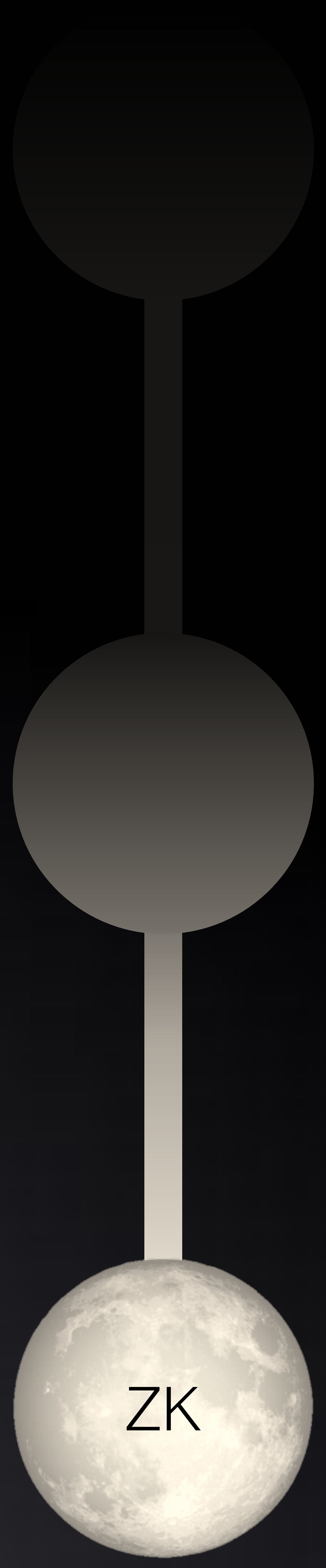

CP^{AGM}_{opn}
trivial empty proof Marlin, Plonk
CP_{eq}
eval random point + Plonk lin tricks
CP^{*}_{deg}
commit to shifted polynomial, batch
CP^{PKE}_{opn}
novel batch ℓ com only 1 G
CP_{qeq}
novel empty proof
CP^{2}_{deg}
commit to shifted polynomial, batch
Blocks



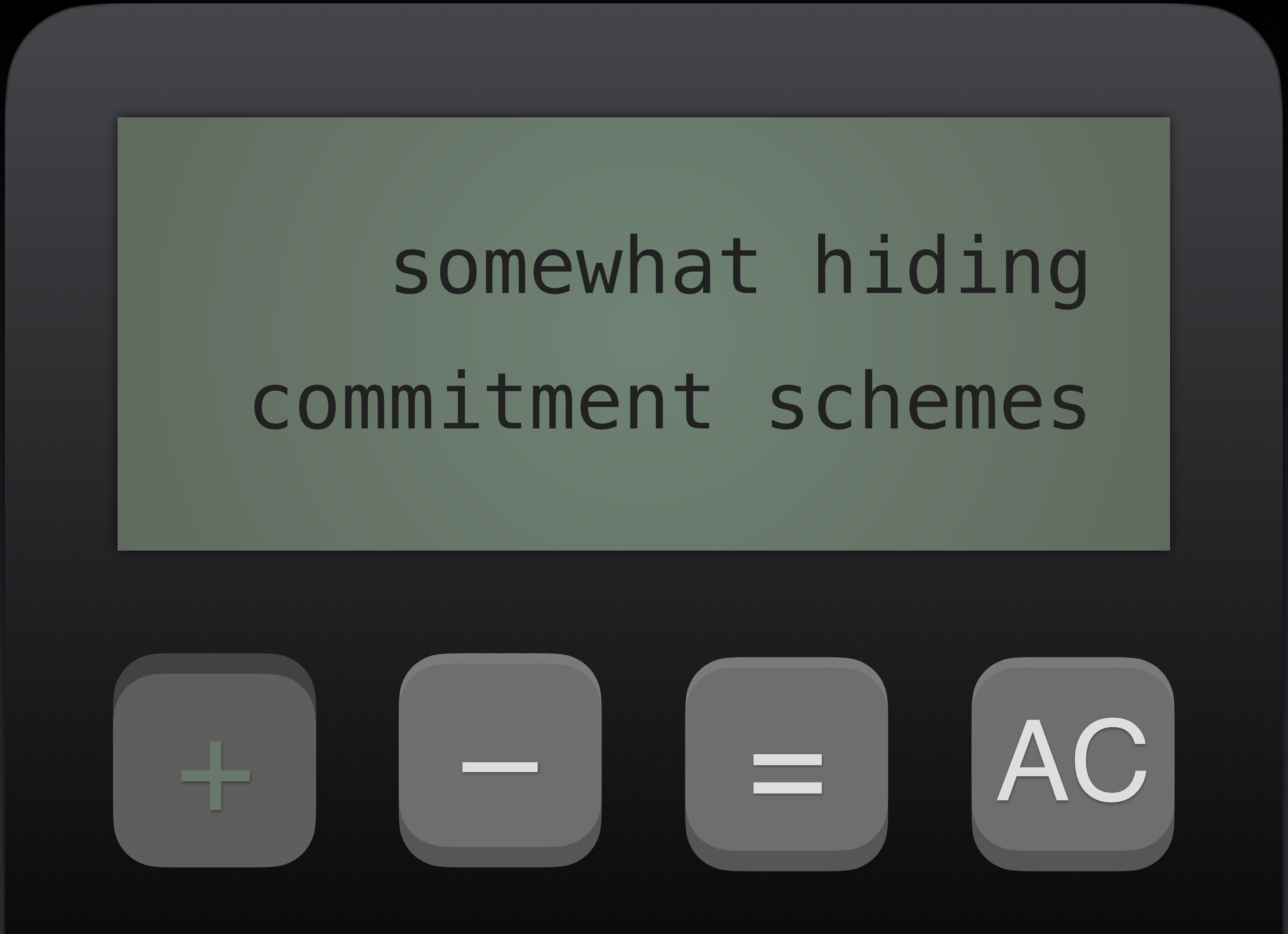
R_{opn}
CP^{AGM}_{opn}
CP^{PKE}_{opn}
R_{eq}
CP_{eq}
CP_{qeq}
R_{deg}
CP^{*}_{deg}
CP^{2}_{deg}
CS_1
PHP_{lite1x}
PHP_{lite2x}
PHP_{R1CS1x}
PHP_{R1CS2x}
PHP_{R1CS3}
CS_2
PHP_{lite1}
PHP_{lite2}
PHP_{R1CS1}
PHP_{R1CS2}
Blocks

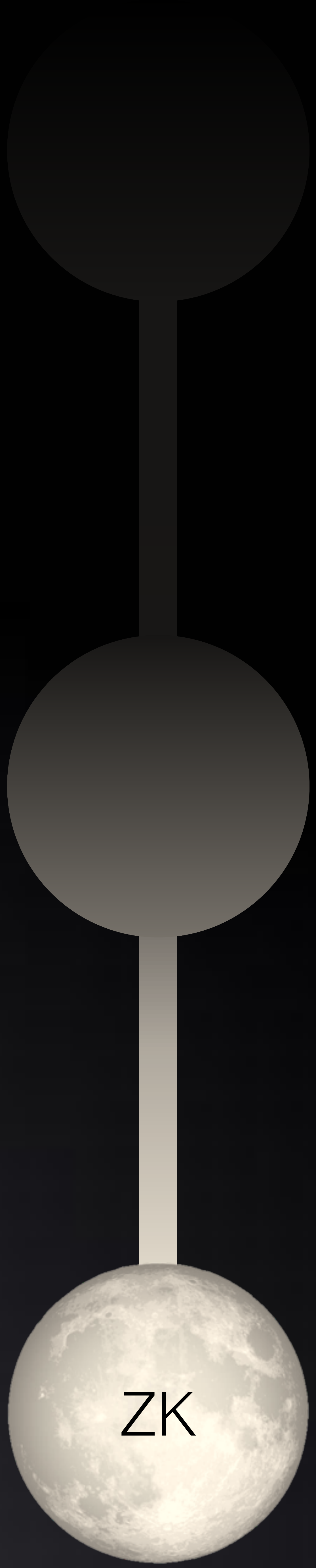
our zero knowledge

$(b_1..b_p)$-leaky zero knowledge CP-SNARKs

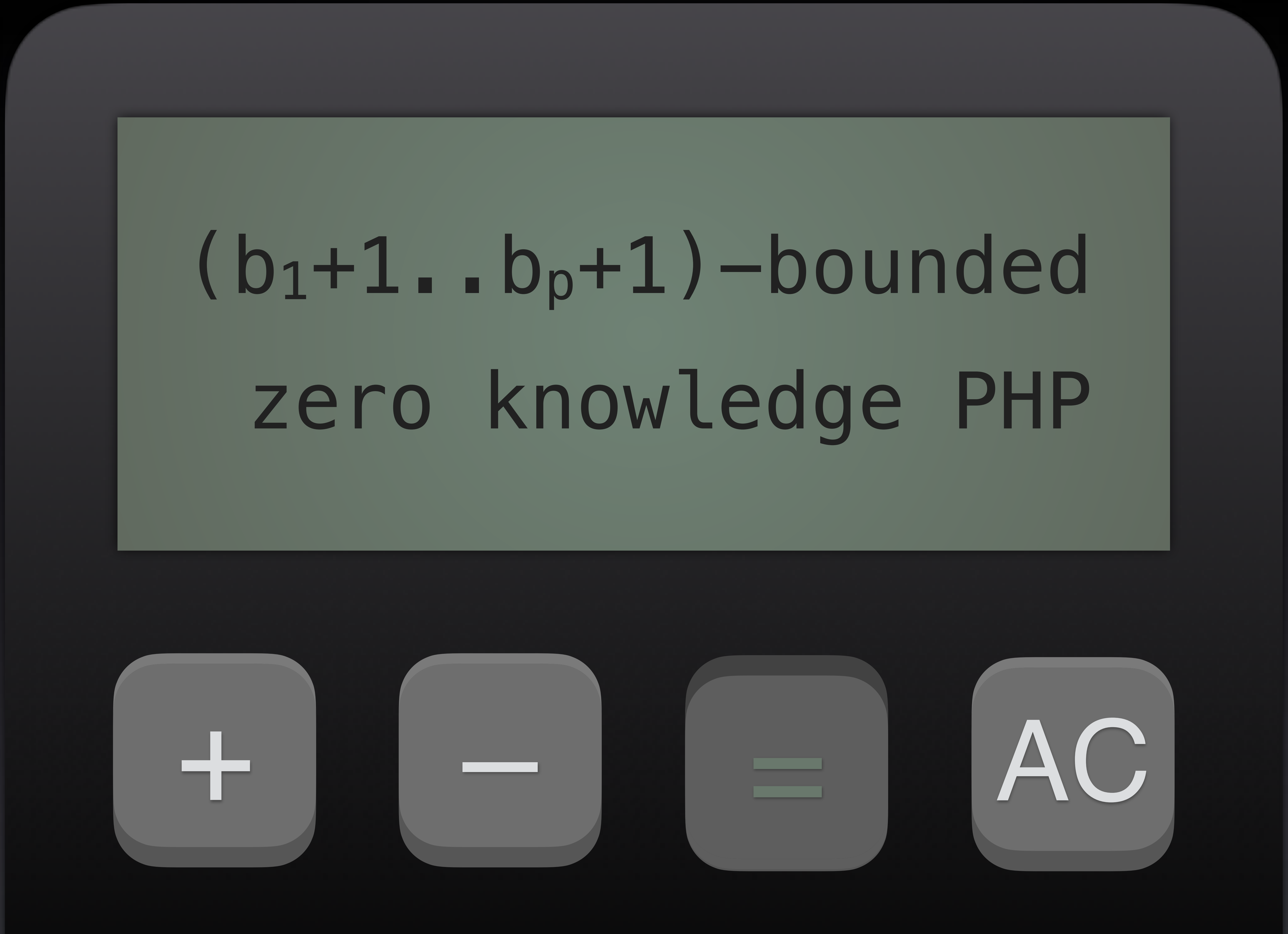


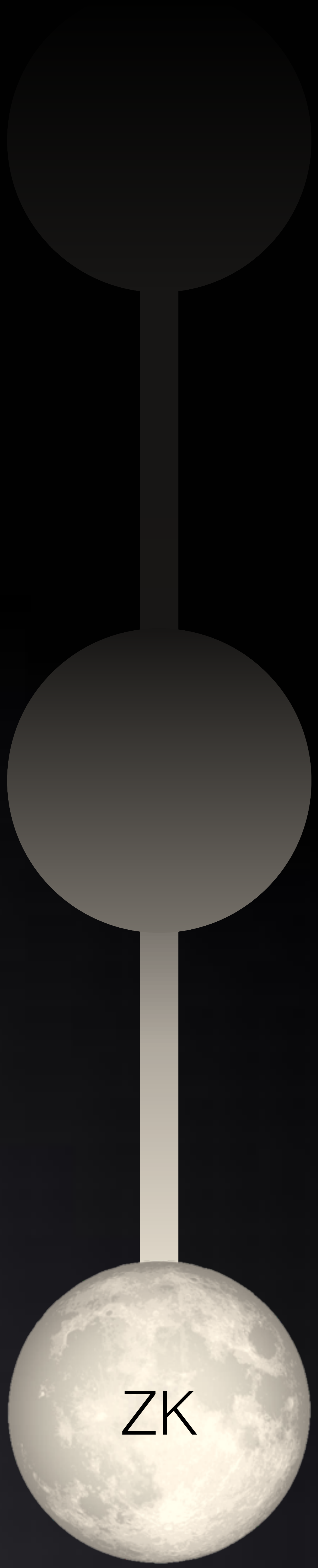
our zero knowledge



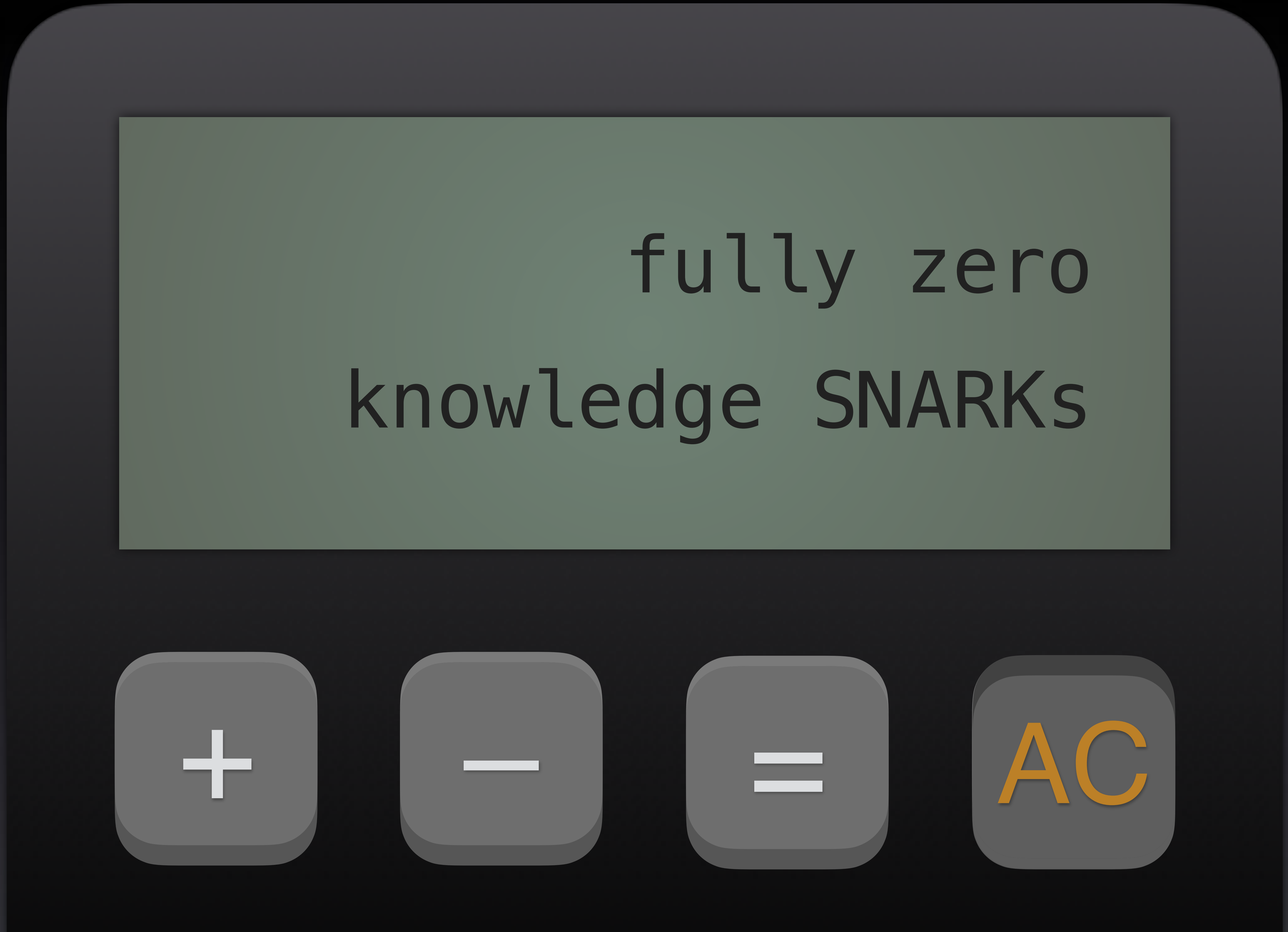


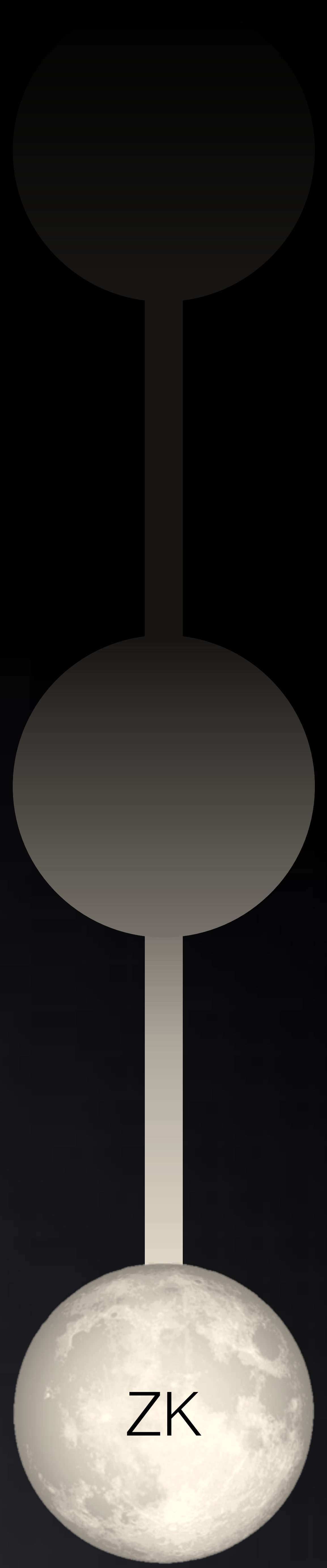
our zero knowledge





our zero knowledge





		srs	π	P_{exp}	V_{pair}
GKM+18	'18	O(n^2)	3 **G**	O(n)	5
Sonic	'19	4N $\mathbf{G_1}$ + 4N $\mathbf{G_2}$	20 $\mathbf{G_1}$ + 16 **F**	273 n	7
Plonk	'19	3N* $\mathbf{G_1}$ + 1 $\mathbf{G_2}$	9 $\mathbf{G_1}$ + 7 **F**	9n + 9a	2
Marlin	'19	3M $\mathbf{G_1}$ + 2 $\mathbf{G_2}$	13 $\mathbf{G_1}$ + 8 **F**	14 n + 8 m	2
LunarLite	'20	M $\mathbf{G_1}$ + M $\mathbf{G_2}$	10 $\mathbf{G_1}$ + 2 **F**	8n + 3m	7
Basilisk	'21	M $\mathbf{G_1}$ + 1 $\mathbf{G_2}$	11 $\mathbf{G_1}$ + 4 **F**	8n + 3m	2

Table

- somewhat hiding commitments
- sparse masking method for polynomials
- compilation with efficient CP-SNARK
- new constraint system with 2 matrices R1CS-lite

Thanks