SNARKs with Preprocessing

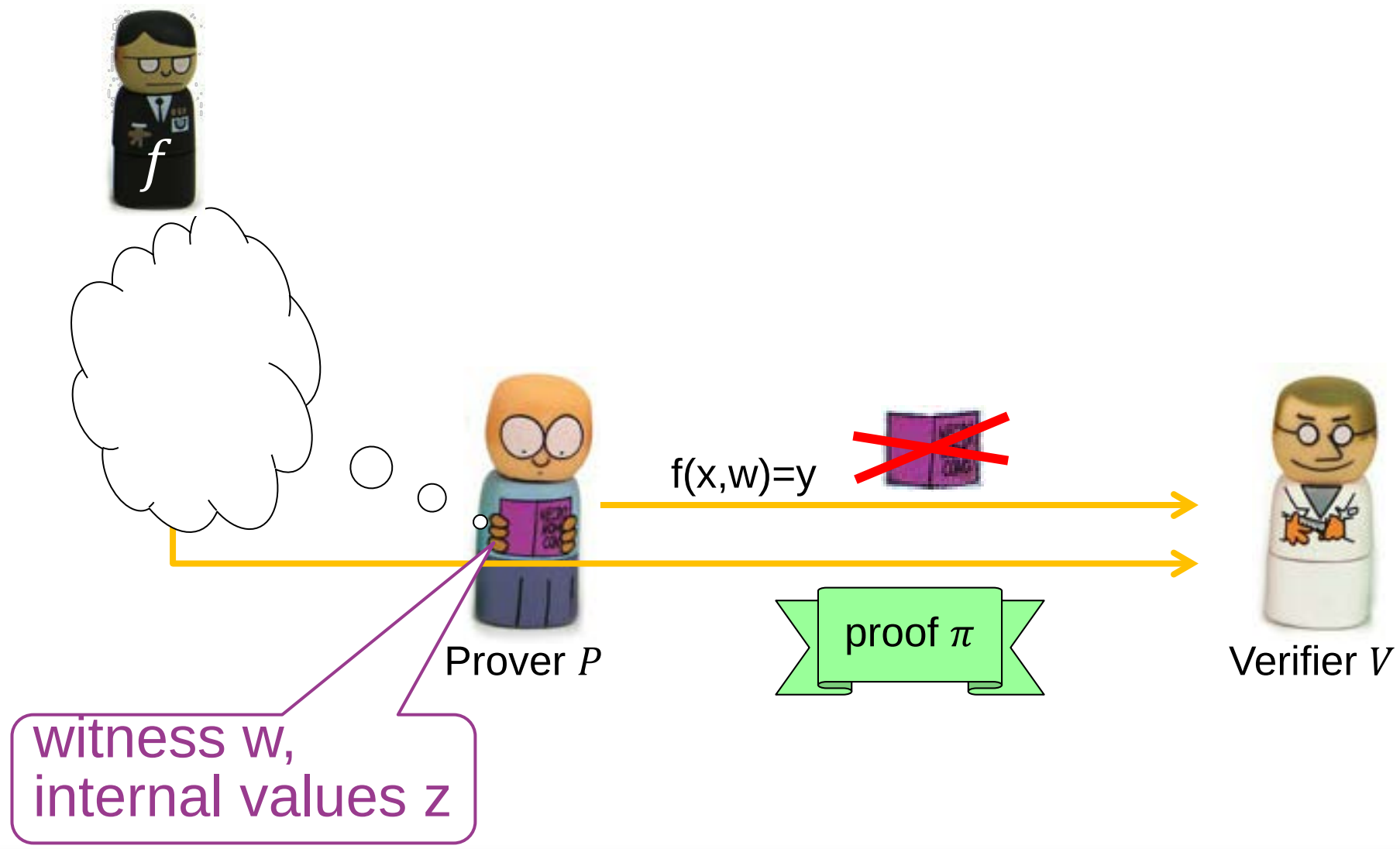
Eran Tromer

G: Someone generates and publishes a common reference string

P: Prover picks NP statement "exists w such that f(x,w)=y" and sends x,y, and a succint proof

V: Verifier efficiently checks proof and is convinced that prover knows a witness w.

zkSNARK for NP: setting



zkSNARK for NP: setting

zero **k**nowledge
succinct (***V*** and π)
noninteractive
argument
of **k**nowledge

zkSNARK

Prover P

f(x,w)=y

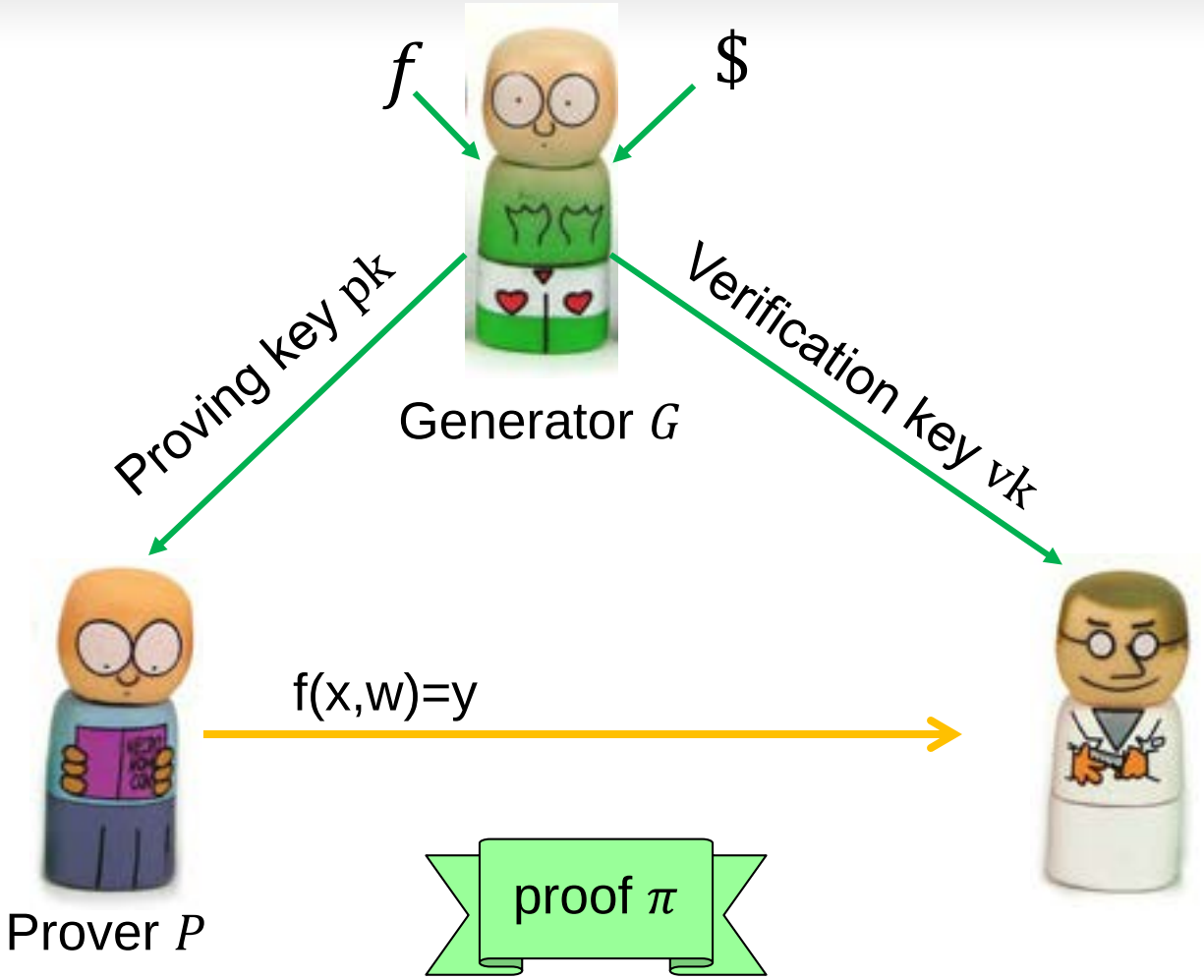
proof π

Verifier V

Preprocessing zkSNARK for NP: setting

Variants:

- Dependence on f
- Cheap / expensive
- Secret / public randomness
- Publicly-verifiable / designated-verifier

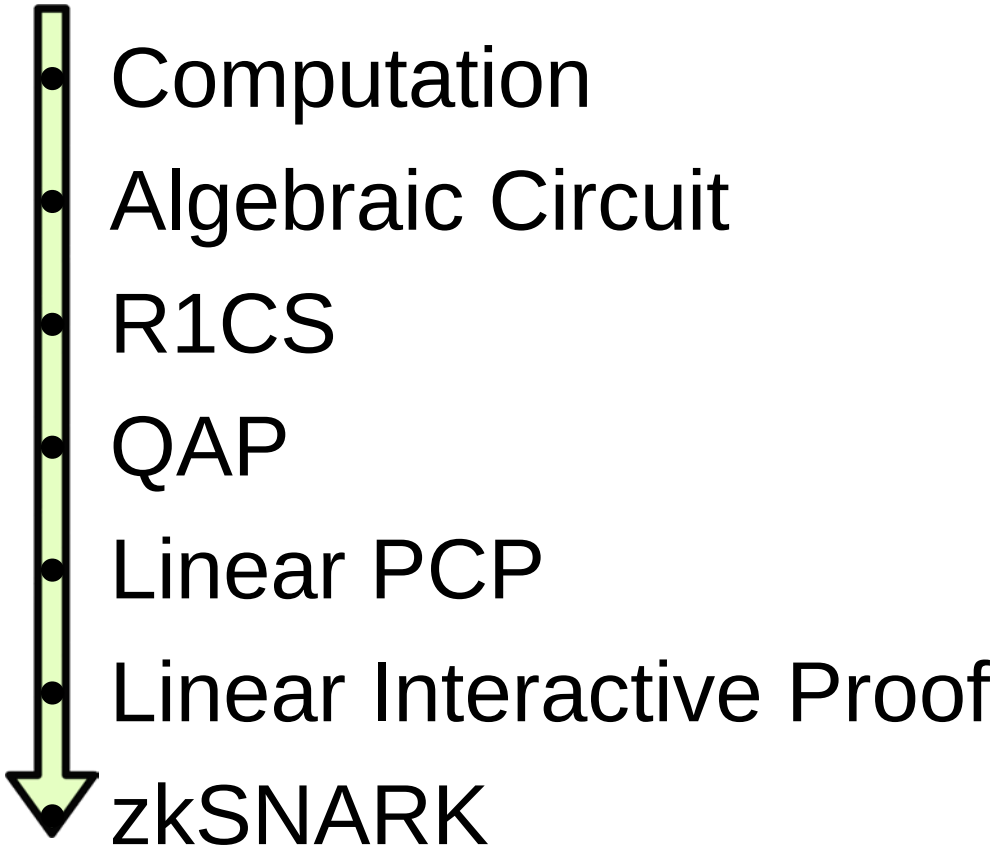


Preprocessing SNARKs for NP

	Proof size (field elements)	CRS size	Prover runtime
[Groth10]	42	$O(s^2)$	$O(s^2)$
[Lipmaa12]	39	$\tilde{O}(s)$	$O(s^2)$
QAP-based [GGPR13] Reinterpreted as linear PCPs: [BCIOP13] [SBVBBW13] Improvements: [PGHR13] [BCTV14usenix] [BBFR15] [CFHKKNPX15] ...	7—8	$O(s)$	$\tilde{O}(s)$
[DFGK14]	4	$O(s)$	$\tilde{O}(s)$

Preprocessing is private-coin and costs $\tilde{O}(s)$

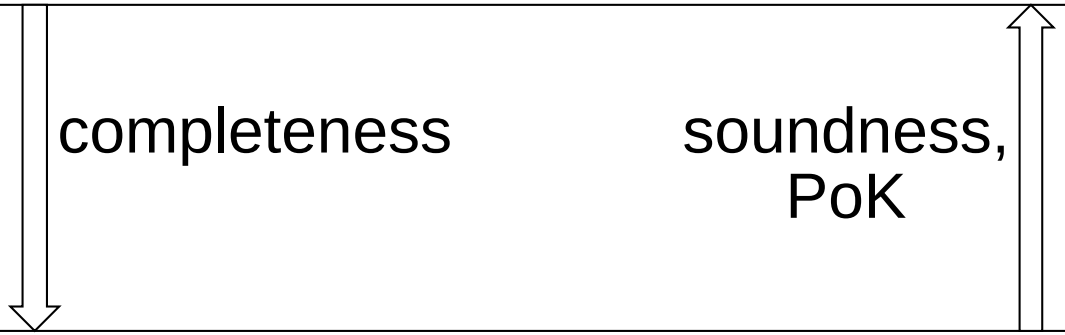
zkSNARK construction via QAP and Linear PCPs



Computation ⇒ Arithmetic Circuit

Efficient computation $f(\cdot)$.

- Deterministic $f(x) \to y$
- Nondeterministic: $\exists w$: $f(x, w) \to y$



Arithmetic circuit $C(\cdot,\cdot)$ over $\mathbb{F}$.

$\exists z$: $C(x, y)$ accepts with internal values $z \in \mathbb{F}^n$

Arithmetic Circuit ⇒ R1CS (Rank-1 Quadratic System) [GGPR13]

Arithmetic circuit $C(\cdot,\cdot)$ over $\mathbb{F}$.

$\exists z$: $C(x,y)$ accepts with internal values $z \in \mathbb{F}^n$

completeness

soundness, PoK

R1CS $(a_j, b_j, c_j)_{j=1}^{m}$ vectors in $\mathbb{F}^k$.

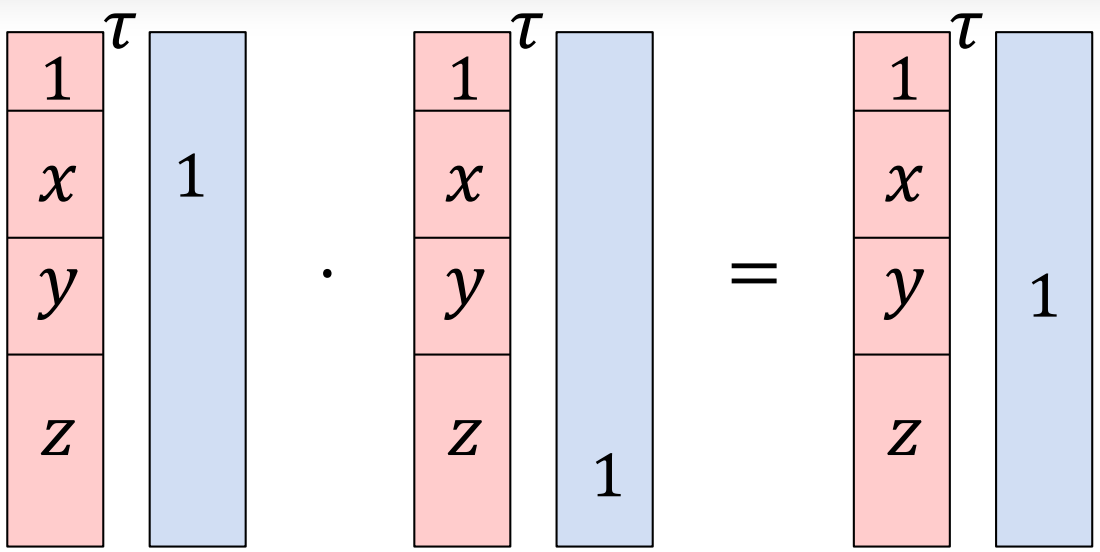
$\exists z \in \mathbb{F}^n$:

$\forall j \in \{1, \ldots, m\}$:

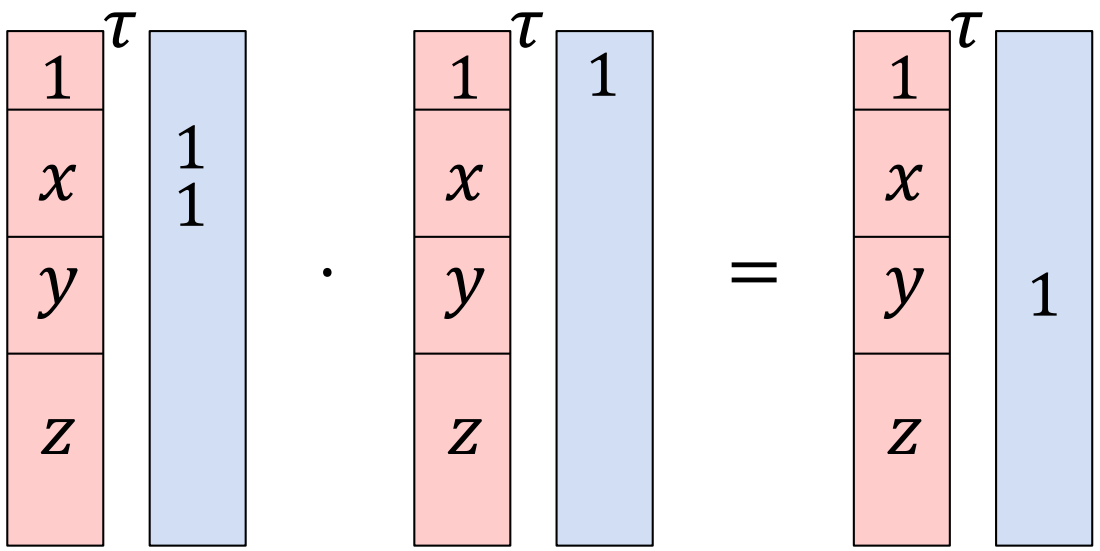
$$\begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} a_j \cdot \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} b_j = \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} c_j$$

Expressing gates as constraints:

Multiplication gate in C converted into a constraint:



Addition gate in C converted into a constraint:



Generally, any bilinear gate.

R1CS (Rank-1 Quadratic Constraint System) ⇒ QAP (Quadratic Arithmetic Program) [GGPR13]

R1CS $(a_j, b_j, c_j)_{j=1}^{m}$ vectors in $\mathbb{F}^k$.

$\exists z \in \mathbb{F}^n$:

$\forall j \in \{1, \ldots, m\}$:

$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} a_j \cdot \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} b_j = \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} c_j$$

Matrices (A, B, C) in $\mathbb{F}^{k \times m}$. $\exists z \in \mathbb{F}^n$:

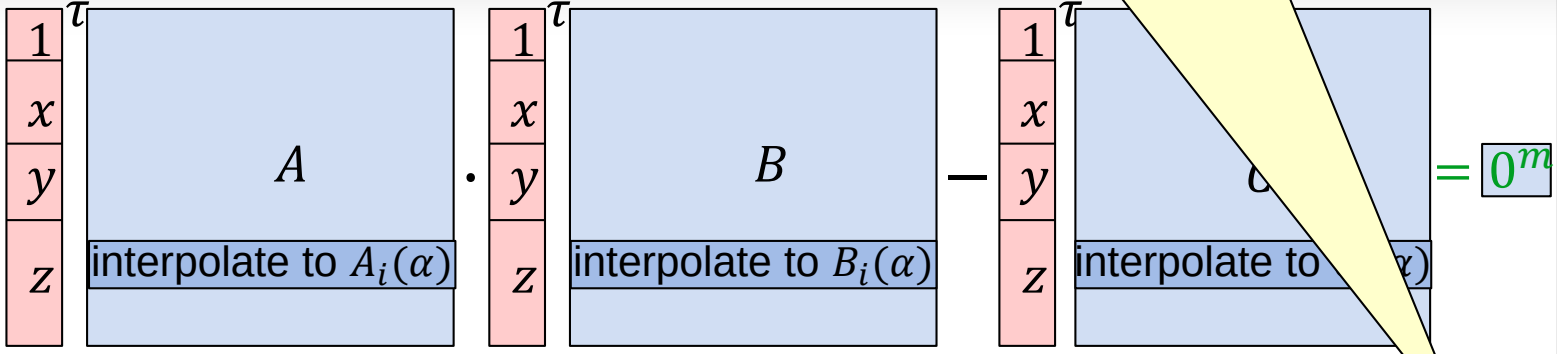
$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} A \cdot \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} B - \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} C = 0^m$$

R1CS ⇒ QAP (cont.)

Intuition: $V(\alpha)$ generates the ideal of all polynomials that vanish on S.

Matrices (A, B, C) in $\mathbb{F}^{k \times m}$.

$\exists z \in \mathbb{F}^m$:



Fix $S = \{\alpha_1, \ldots \alpha_m\} \subset \mathbb{F}$. For $i = 1, \ldots, k$ and $j = 1, \ldots m$:

Let $A_i(\alpha)$ be the degree-$(m-1)$ polynomial such that $A_i(\alpha_j) = A_{i,j}$.

Likewise $B_i(\alpha), C_i(\alpha)$. Let $V(\alpha) = \prod_{j=1}^{m}(\alpha - \alpha_j)$, vanishing on S.

QAP: $\left(A_i(\alpha), B_i(\alpha), C_i(\alpha)\right)_{i=1}^{k}$ and V polynomials in $\mathbb{F}[\alpha]$.

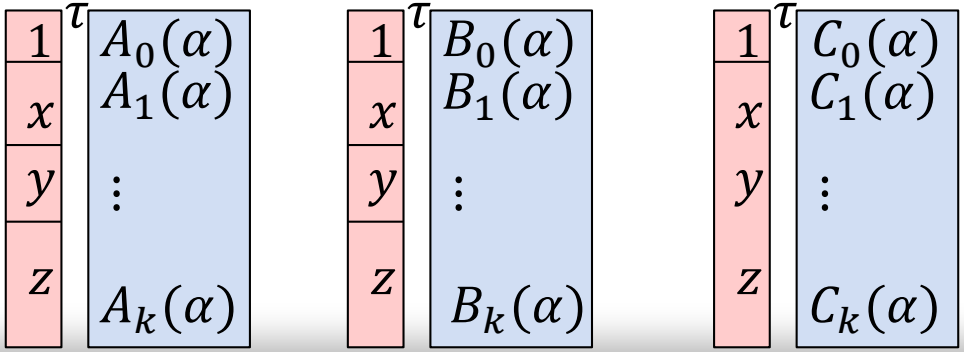
Let $P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)$

$\exists z \in \mathbb{F}^n$:

$V(\alpha)$ divides $P_{x,y,z}(\alpha)$

i.e.,

$\exists H(\alpha): P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)$



QAP ⇒ Linear PCP [GGPR13] [BCIOP13] [BCGTV13]

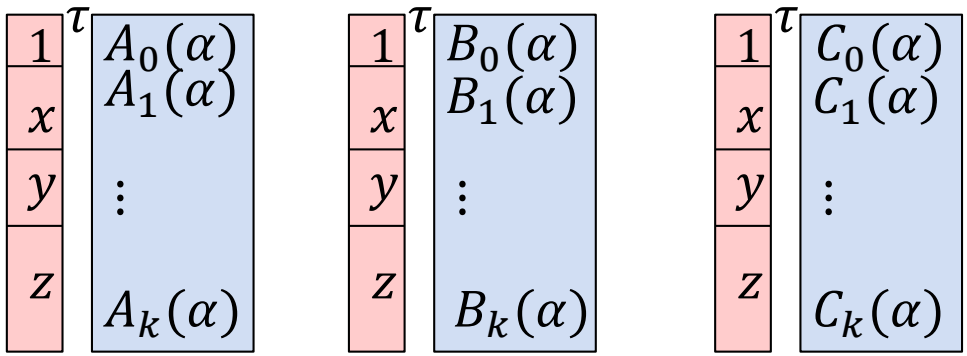
QAP: $\left(A_i(\alpha), B_i(\alpha), C_i(\alpha)\right)_{i=1}^{k}$ and V polynomials in $\mathbb{F}[\alpha]$.

Let $P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)$

$\exists z \in \mathbb{F}^n$:

$\exists H(\alpha)$:

$P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)$



Probabilistic check:

Draw $\tau \leftarrow_R \mathbb{F}$ and check $P_{x,y,z}(\tau) \stackrel{?}{=} H(\tau) \cdot V(\tau)$.

- Soundness: polynomial identity testing with degree $<2m \ll |\mathbb{F}|$

Let $\pi = (1, x, y, z, h)$ where h is the coefficient vector of H.

- Then this check can be done by 4 linear queries to π, retrieving the monomials.
  (Plus a 5th for checking x, y via random linear combination.)
- Any $\tilde{\pi}$ still commits to some low-degree $\tilde{H}(\tau)$ $\widetilde{P_{x,y,z}}(\tau)$.

QAP ⇒ Linear PCP: the algorithms

Let $P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)$

$\exists H(\alpha)$:
$P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)$

$$\begin{bmatrix}1\\x\\y\\z\end{bmatrix}^{\tau}\begin{bmatrix}A_0(\alpha)\\A_1(\alpha)\\\vdots\\A_k(\alpha)\end{bmatrix} \cdot \begin{bmatrix}1\\x\\y\\z\end{bmatrix}^{\tau}\begin{bmatrix}B_0(\alpha)\\B_1(\alpha)\\\vdots\\B_k(\alpha)\end{bmatrix} - \begin{bmatrix}1\\x\\y\\z\end{bmatrix}^{\tau}\begin{bmatrix}C_0(\alpha)\\C_1(\alpha)\\\vdots\\C_k(\alpha)\end{bmatrix}$$

- Prover: compute H and its coefficient vector h;
  Output $\pi = (1, x, y, z, h)$ where h is the coefficient vector of H.
  Complexity: Dominated by computing the m coefficients of H. With suitable FFT: $\sim m \log m + (\#\text{"nonzero entries in "}\ A, B, C)$ field operations.
- Query: Verify: draw $\tau \leftarrow_R \mathbb{F}$, make linear queries to π according to τ.
  Complexity: $\sim 4m + 2(\#\text{nonzero entries in } A, B, C)$ field operations.
- Decision: check a simple quadratic equation in the answers.

Later: important for public verifiability (will use of pairings).

QAP ⇒ Linear PCP: adding ZK

Let $P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)$

$$\begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} \begin{pmatrix} A_0(\alpha) \\ A_1(\alpha) \\ \vdots \\ A_k(\alpha) \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} \begin{pmatrix} B_0(\alpha) \\ B_1(\alpha) \\ \vdots \\ B_k(\alpha) \end{pmatrix} - \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} \begin{pmatrix} C_0(\alpha) \\ C_1(\alpha) \\ \vdots \\ C_k(\alpha) \end{pmatrix}$$

$\delta_1, \delta_2, \delta_3 \leftarrow_R \mathbb{F}$

$+\delta_1 V(\alpha)$ $+\delta_2 V(\alpha)$ $+\delta_3 V(\alpha)$

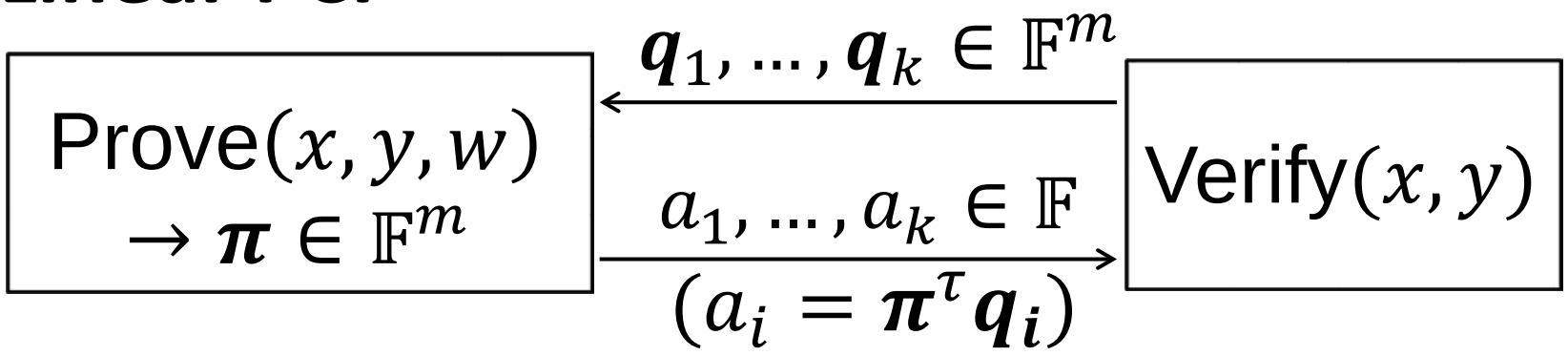
Honest-Verifier Zero Knowledge:

Prover adds random multiple of $V(\alpha)$ to $A, B, C(\alpha)$.

- ZK: The queries to $A_{x,y,z}$, $B_{x,y,z}$, $C_{x,y,z}$ return random independent $\mathbb{F}$ elements. The query to H follows from them. The x, y-consistency query is predictable.

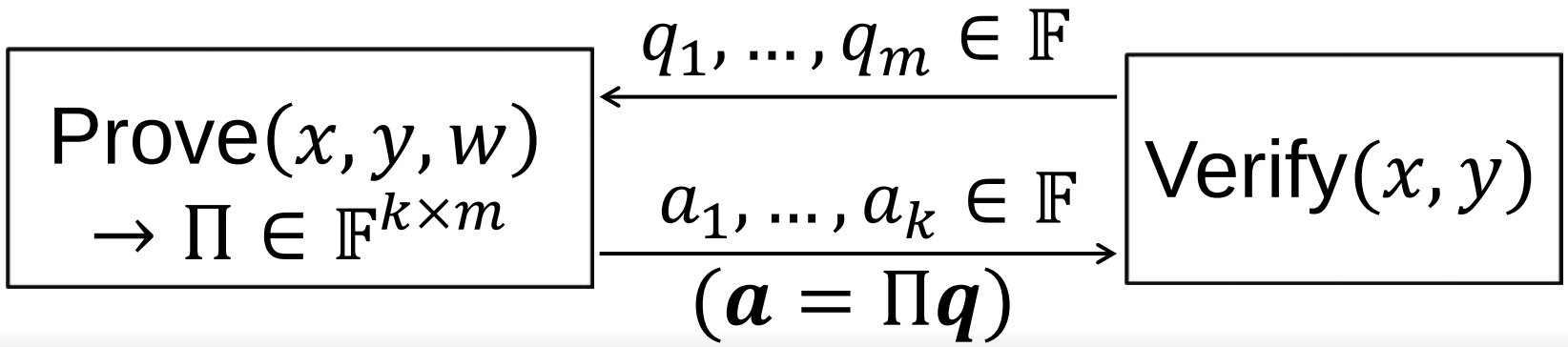
Linear PCP ⇒ Linear Interactive Proof [BCIOP13]

Linear PCP

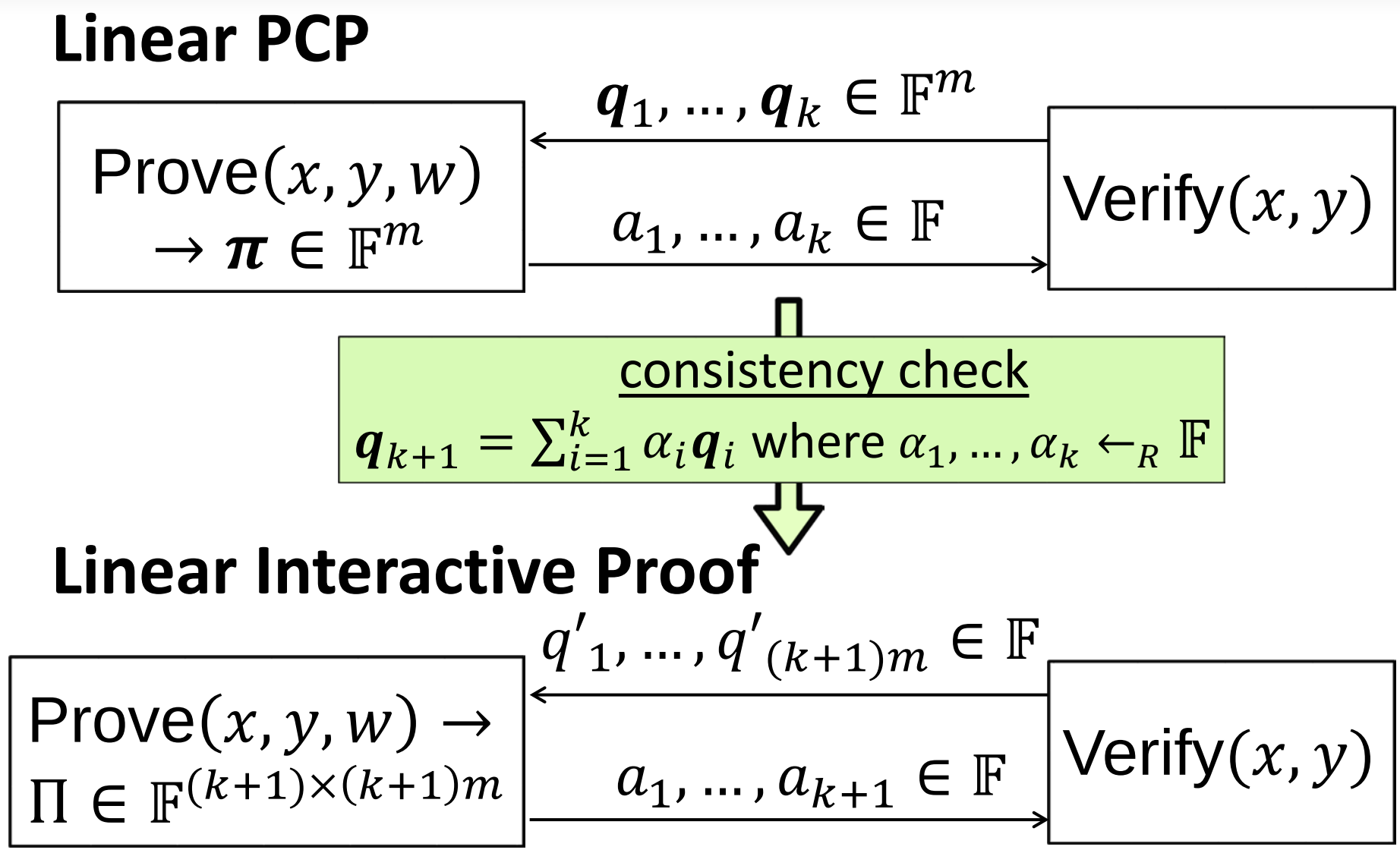


An LPCP is not automatically a LIP, because of consistency (different q_i may "see" different $\boldsymbol{\pi}$)

Linear Interactive Proof



Linear PCP ⇒ Linear Interactive Proof: adding a consistency check [BCIOP13]

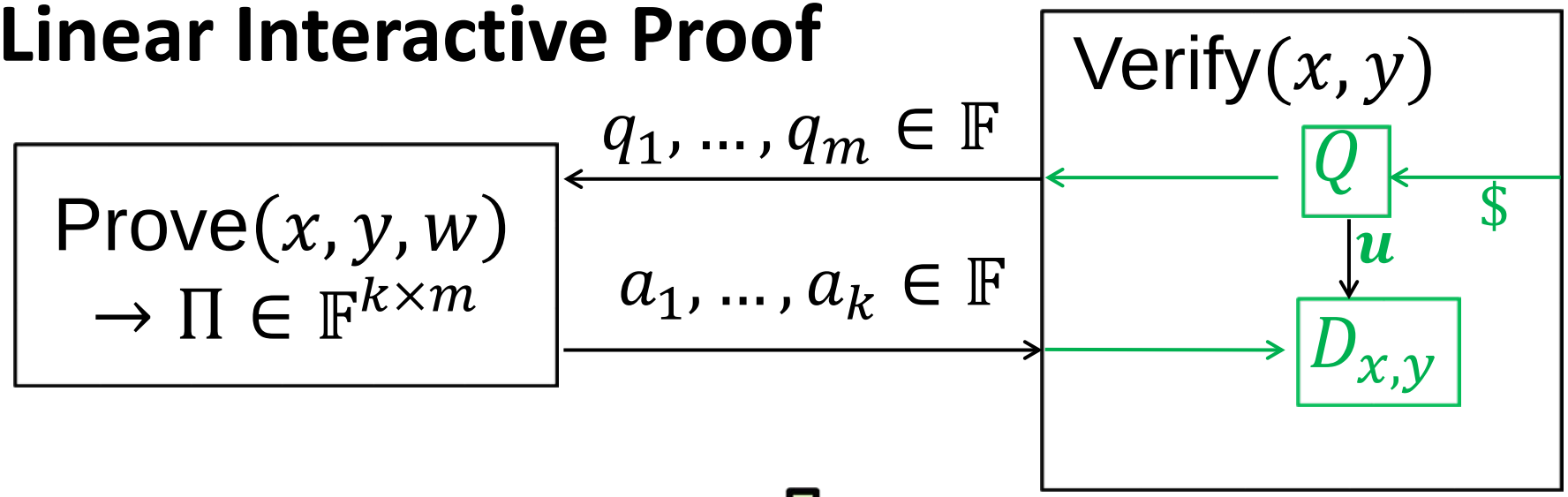


Linear Interactive Proof ⇒ designated-verifier SNARK

[BCIOP13]

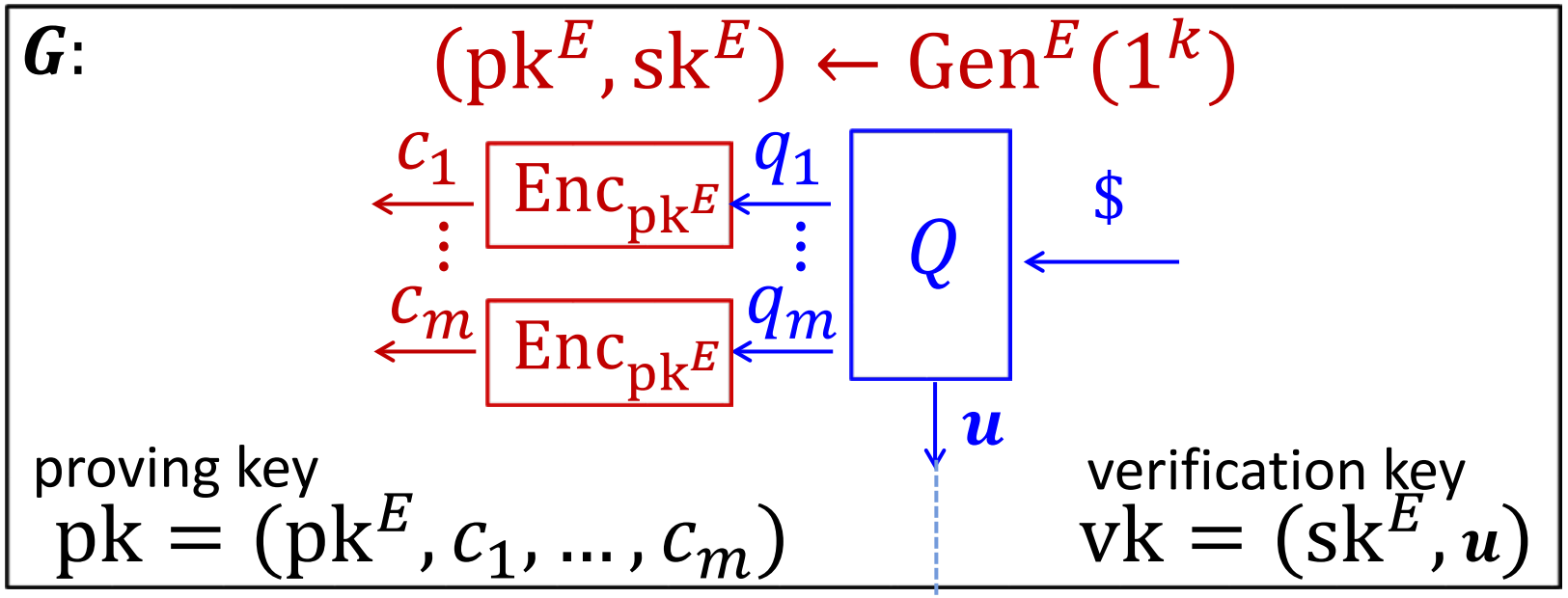
Input-Oblivious

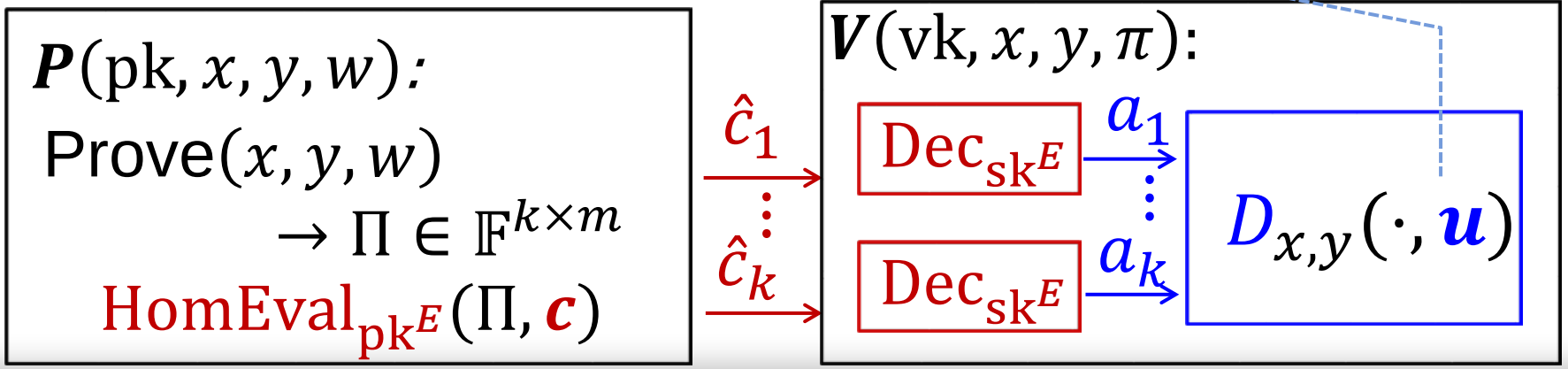
Linear Interactive Proof



Designated-verifier SNARK $(\boldsymbol{G}, \boldsymbol{P}, \boldsymbol{V})$

Linear Interactive Proof ⇒ designated-verifier SNARK: construction





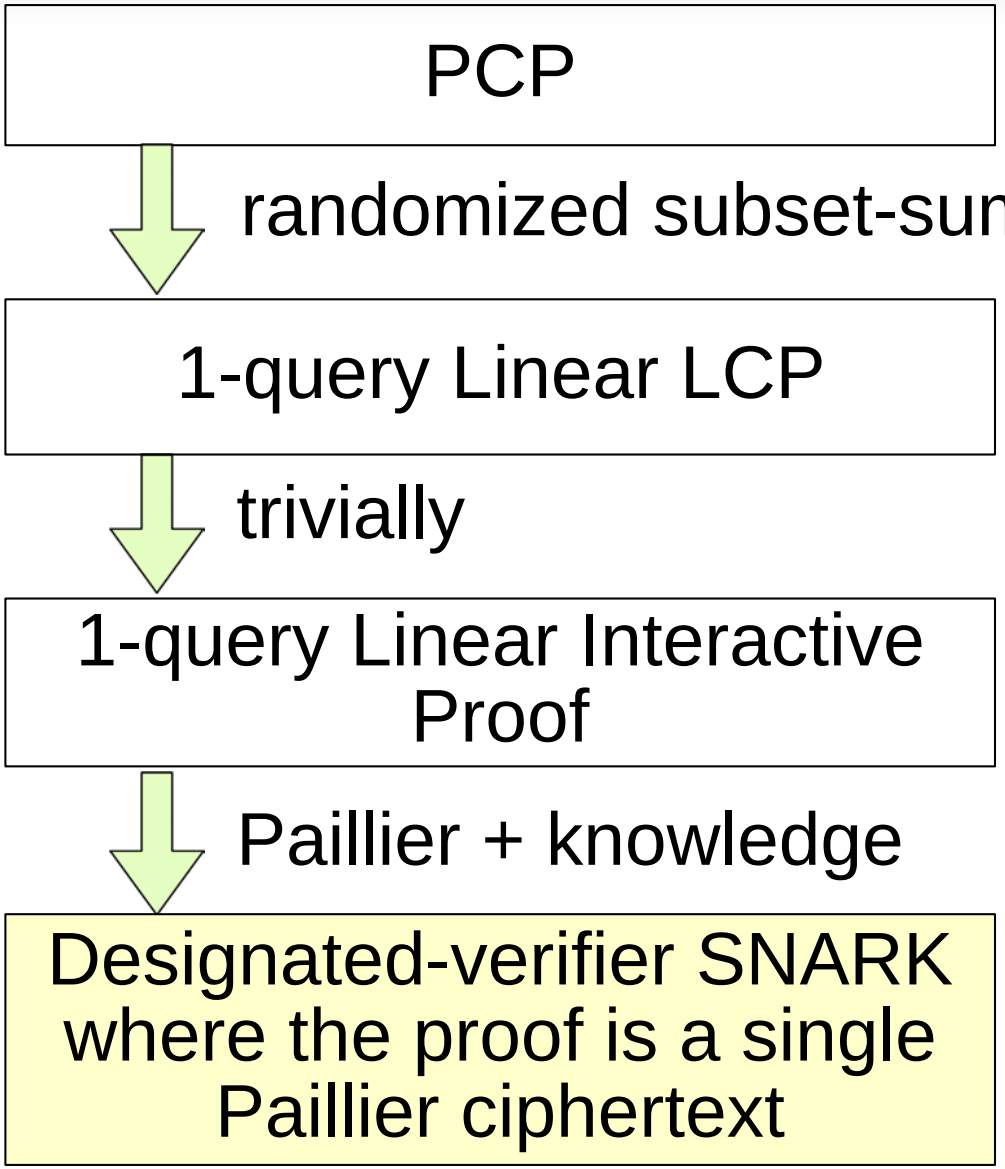
Linear Interactive Proof ⇒ designated-verifier SNARK: assumption

Linear-Only Homomorphism (∼[BSW])
encryption scheme that ONLY allows
$\mathbb{F}$ -linear homomorphic operations
(e.g., knowledge variant of Paillier)

} non-falsifiable assumption

Can add ZK and PoK
with a bit more work.

Bonus: ultrasuccinct designated-verifier SNARK [BCIOP13]

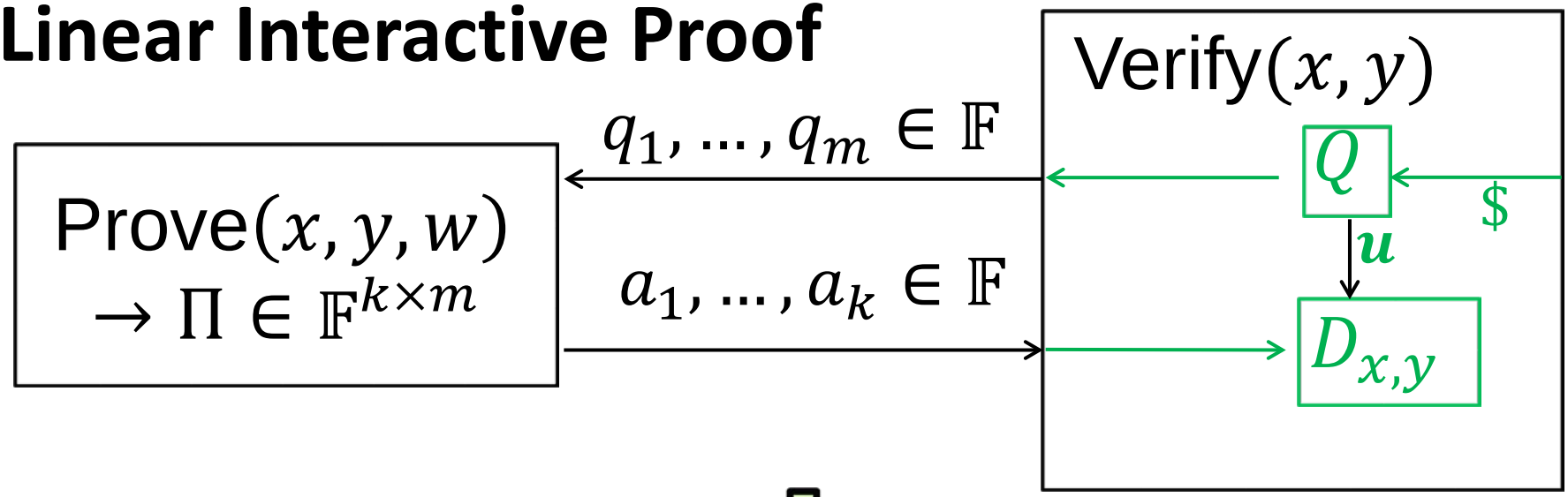


Efficiency caveat: prover computes a full-blown "query-efficient" PCP.

Linear Interactive Proof ⇒ (publicly-verifiable) SNARK [BCIOP13]

Input-Oblivious

Linear Interactive Proof



(Publicly-verifiable) SNARK $(\boldsymbol{G}, \boldsymbol{P}, \boldsymbol{V})$

Linear Interactive Proof ⇒ (publicly-verifiable) SNARK: approach [BCIOP13]

$\mathrm{Enc}_{\mathrm{pk}E}(\cdot)$ satisfies - ~~semantic security~~
- linear-only homomorphism

Being able to test properties of the prover's answers implies that we must **give up** semantic security. Need:

$\mathrm{Enc}_{\mathrm{pk}}(\cdot)$ with
- quadratic predicate test
- linear-only homomorphism
- Δ-power one-wayness

← **Bilinear maps**
Dominates V runtime

← **Knowledge of Exponent**
Unfalsfiable.

Also need the LIP, hence, underlying Linear PCP, to have low-degree Q and D algorithms. QAP works!

zkSNARK construction via QAP and Linear PCPs

- Computation
- Algebraic Circuit
- R1CS
- QAP
- Linear PCP
- Linear Interactive Proof
- zkSNARK

the ONION
America's Finest News Source.™

Frank Gehry No Longer Allowed To Make Sandwiches For Grandkids

Full [PGHR13] protocol ([BCTV14USENIX] variant)

Public parameters. A prime r, two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order r with generators $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and a pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (where $\mathbb{G}_T$ is also cyclic of order r).

(a) Key generator G

- INPUTS: circuit $C\colon \mathbb{F}_r^n \times \mathbb{F}_r^h \to \mathbb{F}_r^l$
- OUTPUTS: proving key pk and verification key vk

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{QAPinst}(C)$; extend $\vec{A}, \vec{B}, \vec{C}$ via
$$A_{m+1} = B_{m+2} = C_{m+3} = Z\,,$$
$$A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0\,.$$
2. Randomly sample $\tau, \rho_\mathsf{A}, \rho_\mathsf{B}, \alpha_\mathsf{A}, \alpha_\mathsf{B}, \alpha_\mathsf{C}, \beta, \gamma \in \mathbb{F}_r$.
3. Set $\mathsf{pk} := (C, \mathsf{pk}_\mathsf{A}, \mathsf{pk}'_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{pk}'_\mathsf{B}, \mathsf{pk}_\mathsf{C}, \mathsf{pk}'_\mathsf{C}, \mathsf{pk}_\mathsf{K}, \mathsf{pk}_\mathsf{H})$ where for $i = 0,1,\ldots,m+3$:
$$\mathsf{pk}_{\mathsf{A},i} := A_i(\tau)\rho_\mathsf{A}\mathcal{P}_1\,, \quad \mathsf{pk}'_{\mathsf{A},i} := A_i(\tau)\alpha_\mathsf{A}\rho_\mathsf{A}\mathcal{P}_1\,,$$
$$\mathsf{pk}_{\mathsf{B},i} := B_i(\tau)\rho_\mathsf{B}\mathcal{P}_2\,, \quad \mathsf{pk}'_{\mathsf{B},i} := B_i(\tau)\alpha_\mathsf{B}\rho_\mathsf{B}\mathcal{P}_1\,,$$
$$\mathsf{pk}_{\mathsf{C},i} := C_i(\tau)\rho_\mathsf{A}\rho_\mathsf{B}\mathcal{P}_1\,, \mathsf{pk}'_{\mathsf{C},i} := C_i(\tau)\alpha_\mathsf{C}\rho_\mathsf{A}\rho_\mathsf{B}\mathcal{P}_1\,,$$
$$\mathsf{pk}_{\mathsf{K},i} := \beta\big(A_i(\tau)\rho_\mathsf{A} + B_i(\tau)\rho_\mathsf{B} + C_i(\tau)\rho_\mathsf{A}\rho_\mathsf{B}\big)\mathcal{P}_1\,,$$
and for $i = 0,1,\ldots,d$, $\mathsf{pk}_{\mathsf{H},i} := \tau^i\mathcal{P}_1$.
4. Set $\mathsf{vk} := (\mathsf{vk}_\mathsf{A}, \mathsf{vk}_\mathsf{B}, \mathsf{vk}_\mathsf{C}, \mathsf{vk}_\gamma, \mathsf{vk}^1_{\beta\gamma}, \mathsf{vk}^2_{\beta\gamma}, \mathsf{vk}_\mathsf{Z}, \mathsf{vk}_\mathsf{IC})$ where
$$\mathsf{vk}_\mathsf{A} := \alpha_\mathsf{A}\mathcal{P}_2\,, \mathsf{vk}_\mathsf{B} := \alpha_\mathsf{B}\mathcal{P}_1\,, \mathsf{vk}_\mathsf{C} := \alpha_\mathsf{C}\mathcal{P}_2$$
$$\mathsf{vk}_\gamma := \gamma\mathcal{P}_2\,, \quad \mathsf{vk}^1_{\beta\gamma} := \gamma\beta\mathcal{P}_1\,, \quad \mathsf{vk}^2_{\beta\gamma} := \gamma\beta\mathcal{P}_2\,,$$
$$\mathsf{vk}_\mathsf{Z} := Z(\tau)\rho_\mathsf{A}\rho_\mathsf{B}\mathcal{P}_2\,, \big(\mathsf{vk}_{\mathsf{IC},i}\big)_{i=0}^n := \big(A_i(\tau)\rho_\mathsf{A}\mathcal{P}_1\big)_{i=0}^n\,.$$
5. Output $(\mathsf{pk}, \mathsf{vk})$.

Key sizes. When invoked on a circuit $C\colon \mathbb{F}_r^n \times \mathbb{F}_r^h \to \mathbb{F}_r^l$ with a wires and b (bilinear) gates, the key generator outputs:

- pk with $(6a+b+n+l+26)$ $\mathbb{G}_1$-elements and $(a+4)$ $\mathbb{G}_2$-elements;
- vk with $(n+3)$ $\mathbb{G}_1$-elements and 5 $\mathbb{G}_2$-elements.

Proof size. The proof always has 7 $\mathbb{G}_1$-elements and 1 $\mathbb{G}_2$-element.

(b) Prover P

- INPUTS: proving key pk, input $\vec{x} \in \mathbb{F}_r^n$, and witness $\vec{a} \in \mathbb{F}_r^h$
- OUTPUTS: proof π

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{QAPinst}(C)$.
2. Compute $\vec{s} := \mathsf{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$.
3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$.
4. Compute $\vec{h} = (h_0, h_1, \ldots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)-C(z)}{Z(z)}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:
$$A(z) := A_0(z) + \textstyle\sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z)\,,$$
$$B(z) := B_0(z) + \textstyle\sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z)\,,$$
$$C(z) := C_0(z) + \textstyle\sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z)\,.$$
5. Set $\tilde{\mathsf{pk}}_\mathsf{A} :=$ "same as pk_A, but with $\mathsf{pk}_{\mathsf{A},i} = 0$ for $i = 0,1,\ldots,n$".
Set $\tilde{\mathsf{pk}}'_\mathsf{A} :=$ "same as pk'_A, but with $\mathsf{pk}'_{\mathsf{A},i} = 0$ for $i = 0,1,\ldots,n$".
6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, compute
$$\pi_\mathsf{A} := \langle \vec{c}, \tilde{\mathsf{pk}}_\mathsf{A}\rangle,\ \pi'_\mathsf{A} := \langle \vec{c}, \tilde{\mathsf{pk}}'_\mathsf{A}\rangle,\ \pi_\mathsf{B} := \langle \vec{c}, \mathsf{pk}_\mathsf{B}\rangle,\ \pi'_\mathsf{B} := \langle \vec{c}, \mathsf{pk}'_\mathsf{B}\rangle,$$
$$\pi_\mathsf{C} := \langle \vec{c}, \mathsf{pk}_\mathsf{C}\rangle,\ \pi'_\mathsf{C} := \langle \vec{c}, \mathsf{pk}'_\mathsf{C}\rangle,\ \pi_\mathsf{K} := \langle \vec{c}, \mathsf{pk}_\mathsf{K}\rangle,\ \pi_\mathsf{H} := \langle \vec{h}, \mathsf{pk}_\mathsf{H}\rangle.$$
7. Output $\pi := (\pi_\mathsf{A}, \pi'_\mathsf{A}, \pi_\mathsf{B}, \pi'_\mathsf{B}, \pi_\mathsf{C}, \pi'_\mathsf{C}, \pi_\mathsf{K}, \pi_\mathsf{H})$.

(c) Verifier V

- INPUTS: verification key vk, input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\mathsf{vk}_{\vec{x}} := \mathsf{vk}_{\mathsf{IC},0} + \sum_{i=1}^n x_i \mathsf{vk}_{\mathsf{IC},i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments for A, B, C:
$$e(\pi_\mathsf{A}, \mathsf{vk}_\mathsf{A}) = e(\pi'_\mathsf{A}, \mathcal{P}_2)\,, e(\mathsf{vk}_\mathsf{B}, \pi_\mathsf{B}) = e(\pi'_\mathsf{B}, \mathcal{P}_2)\,,$$
$$e(\pi_\mathsf{C}, \mathsf{vk}_\mathsf{C}) = e(\pi'_\mathsf{C}, \mathcal{P}_2)\,.$$
3. Check same coefficients were used:
$$e(\pi_\mathsf{K}, \mathsf{vk}_\gamma) = e(\mathsf{vk}_{\vec{x}} + \pi_\mathsf{A} + \pi_\mathsf{C}, \mathsf{vk}^2_{\beta\gamma}) \cdot e(\mathsf{vk}^1_{\beta\gamma}, \pi_\mathsf{B})\,.$$
4. Check QAP divisibility:
$$e(\mathsf{vk}_{\vec{x}} + \pi_\mathsf{A}, \pi_\mathsf{B}) = e(\pi_\mathsf{H}, \mathsf{vk}_\mathsf{Z}) \cdot e(\pi_\mathsf{C}, \mathcal{P}_2)\,.$$
5. Accept if and only if all the above checks succeeded.

[PGHR13] assumptions

$q = poly(circuit\ size)$

- q-power Diffie-Hellman
- q-strong Diffie-Hellman
- q-power Knowledge of Exponent

Assumption 2 (q-**PKE** [21]) *The q-power knowledge of exponent assumption holds for $\mathcal{G}$ if for all $\mathcal{A}$ there exists a non-uniform probabilistic polynomial time extractor $\chi_{\mathcal{A}}$ such that*

$$Pr[\ (p,\mathbb{G},\mathbb{G}_T,e) \leftarrow \mathcal{G}(1^\kappa)\ ;\ g \leftarrow \mathbb{G}\backslash\{1\}\ ;\ \alpha, s \leftarrow \mathbb{Z}_p^*\ ;$$
$$\sigma \leftarrow (p,\mathbb{G},\mathbb{G}_T,e,g,g^s,\ldots,g^{s^q},g^\alpha,g^{\alpha s},\ldots,g^{\alpha s^q})\ ;$$
$$(c,\hat{c}\ ;\ a_0,\ldots,a_q) \leftarrow (\mathcal{A} \parallel \chi_{\mathcal{A}})(\sigma,z)\ :$$
$$\hat{c} = c^\alpha \wedge c \neq \textstyle\prod_{i=0}^q g^{a_i s^i}] = \mathtt{negl}(\kappa)$$

for any auxiliary information $z \in \{0,1\}^{\mathrm{poly}(\kappa)}$ that is generated independently of α. Note that $(y;z) \leftarrow (\mathcal{A} \parallel \chi_{\mathcal{A}})(x)$ signifies that on input x, $\mathcal{A}$ outputs y, and that $\chi_{\mathcal{A}}$, given the same input x and $\mathcal{A}$'s random tape, produces z.

zkSNARK backend implementations

- Pinocchio/Geppetto
  `https://vc.codeplex.com`
  [PGHR13] [CFHKKNPZ15]
- libsnark
  `github.com/scipr-lab/libsnark`
  [BCGTV13a] [BCTV14crypto] [BCTV14usenix] …
- snarklib
  `github.com/jancarlsson/snarklib`
  (clone of libsnark with different C++ style by “Jan Carlsson”)

Numerous frontends (some included in the above), to be discussed tomorrow.

Example: libsnark backends

- [PGHR13] backend with [BCTV14USENIX] improvements
  - speedof verifier by merging parts of the pairing computation
  - reduced verification key size to ~1/3 (when #inputs≪#gates)
- Square Span Programs [DFGK14] backend
- ADSNARK backend, [BBFR15] backend
- Tailored libraries for finite fields, ECC, pairings

1M arithmetic gates, 1000-bit input, desktop PC	80-bit security	128-bit security
Generator	97 s	117 s
Prover	115 s	147 s
Verifier	4.9 ms ($4.7 + 0.0004\|x\|$ ms)	5.1 ms ($4.8 + 0.0005\|x\|$ ms)
Proof size	230 B	288 B

- Full code, MIT license `github.com/scipr-lab/libsnark`

Extensions/variants

- ADSNARK [BBFR15]
  - a preprocessing SNARKs for proving statements on authenticated data
- [DFGK14]
  - A preprocessing SNARK for the language Square Span Programs, more efficient for Boolean circuits