Modern Verifiable Computation

Justin Thaler

Yahoo Labs, New York City

Lecture Outline

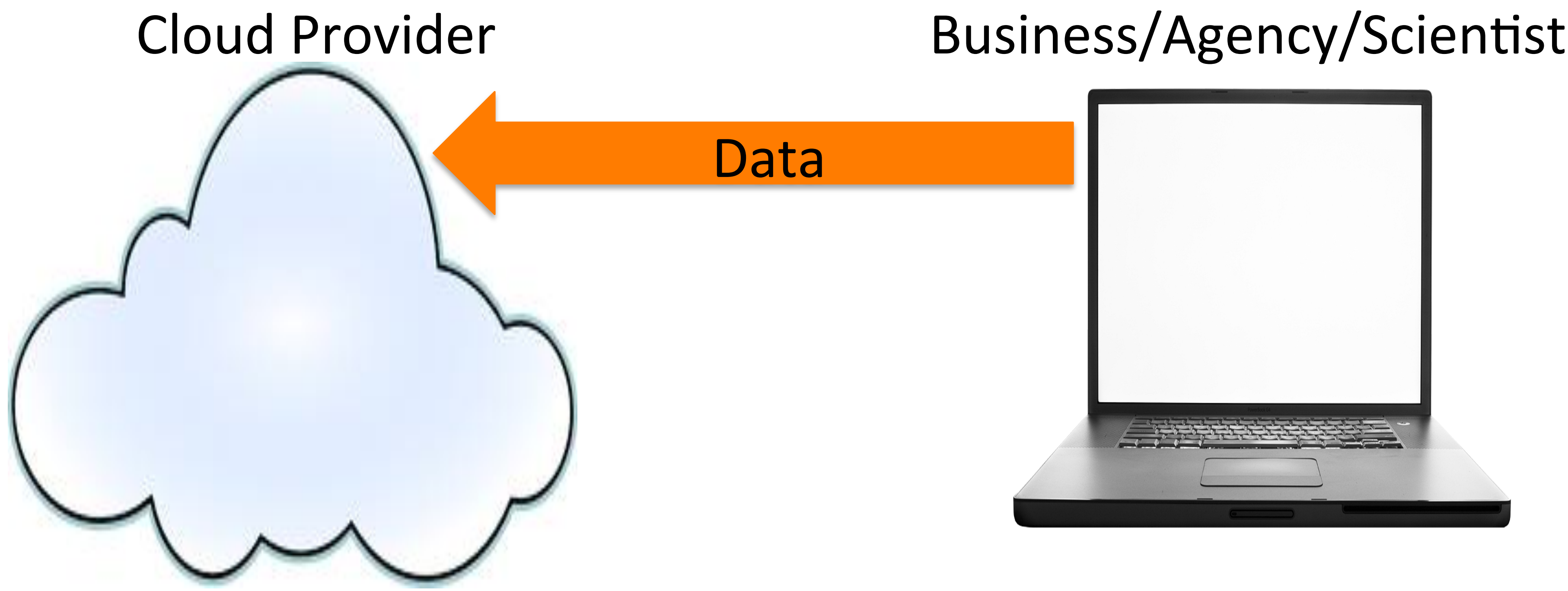
1. Interactive Proofs
   - Motivation, History of Work
   - Techniques:
     - Sum-Check Protocol
     - IP=PSPACE [LFKN, Shamir]
     - MatMult Protocol [T., 2013]
     - GKR Protocol [GKR, 2008]
2. Multi-Prover Interactive Proofs
   - Why can MIPs with polynomial-time verifiers solve harder problems than IPs?
   - Why can MIPs with linear-time verifiers solve "easy" problems more efficiently than IPs?
   - Sketch of a state-of-the-art MIP [BTVW, unpublished]
3. PCPs
   - Reltionship to MIPs
   - A first PCP from an MIP
   - A state-of-the-art PCP [BSS08]
4. Argument Systems
   - From "short" PCPs [Kilian 1992]
   - Without short PCPs [IKO 2007, GGPR 2013]
     - Basis of all implemented argument systems

Interactive Proofs: Motivation and Model

Outsourcing

- Many applications require outsourcing computation to untrusted service providers.
  - Main motivation: commercial cloud computing services.
  - Also, weak peripheral devices; fast but faulty co-processors.
  - Volunteer Computing (SETI@home, World Community Grid, etc.)
- User requires a guarantee that the cloud performed the computation correctly.

Cloud Computing

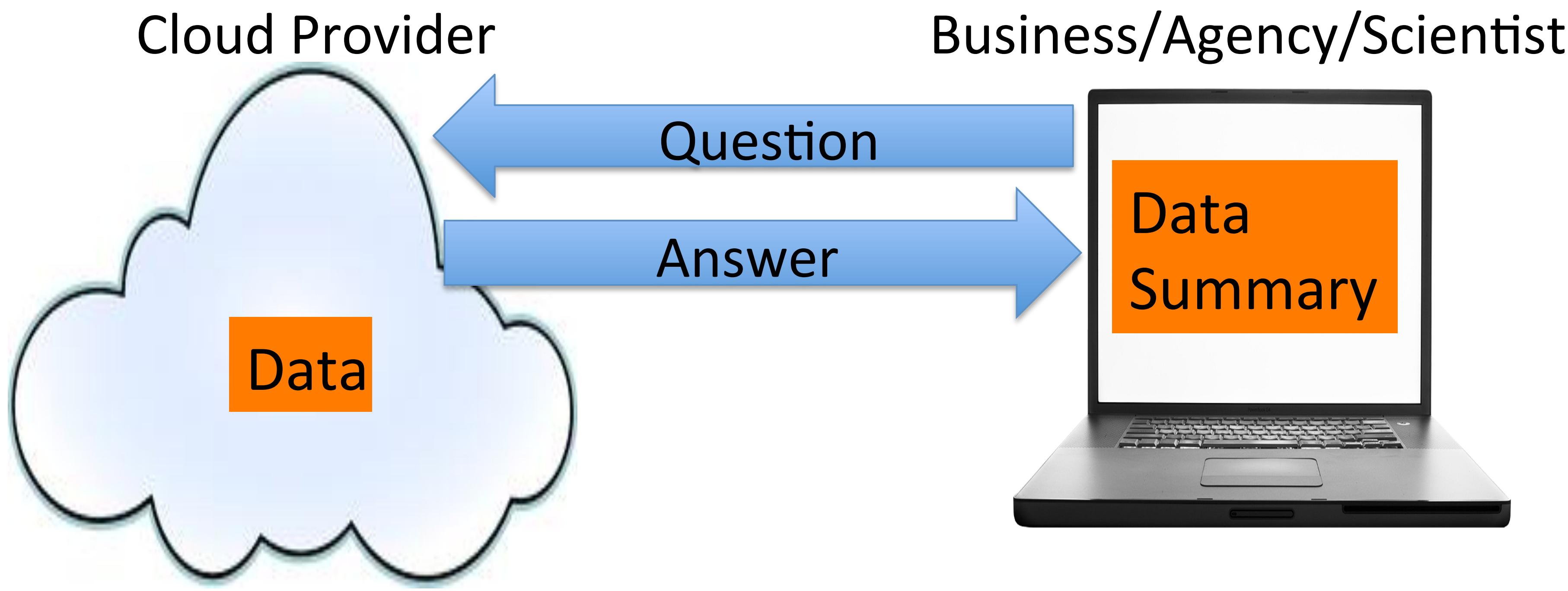


Cloud Computing

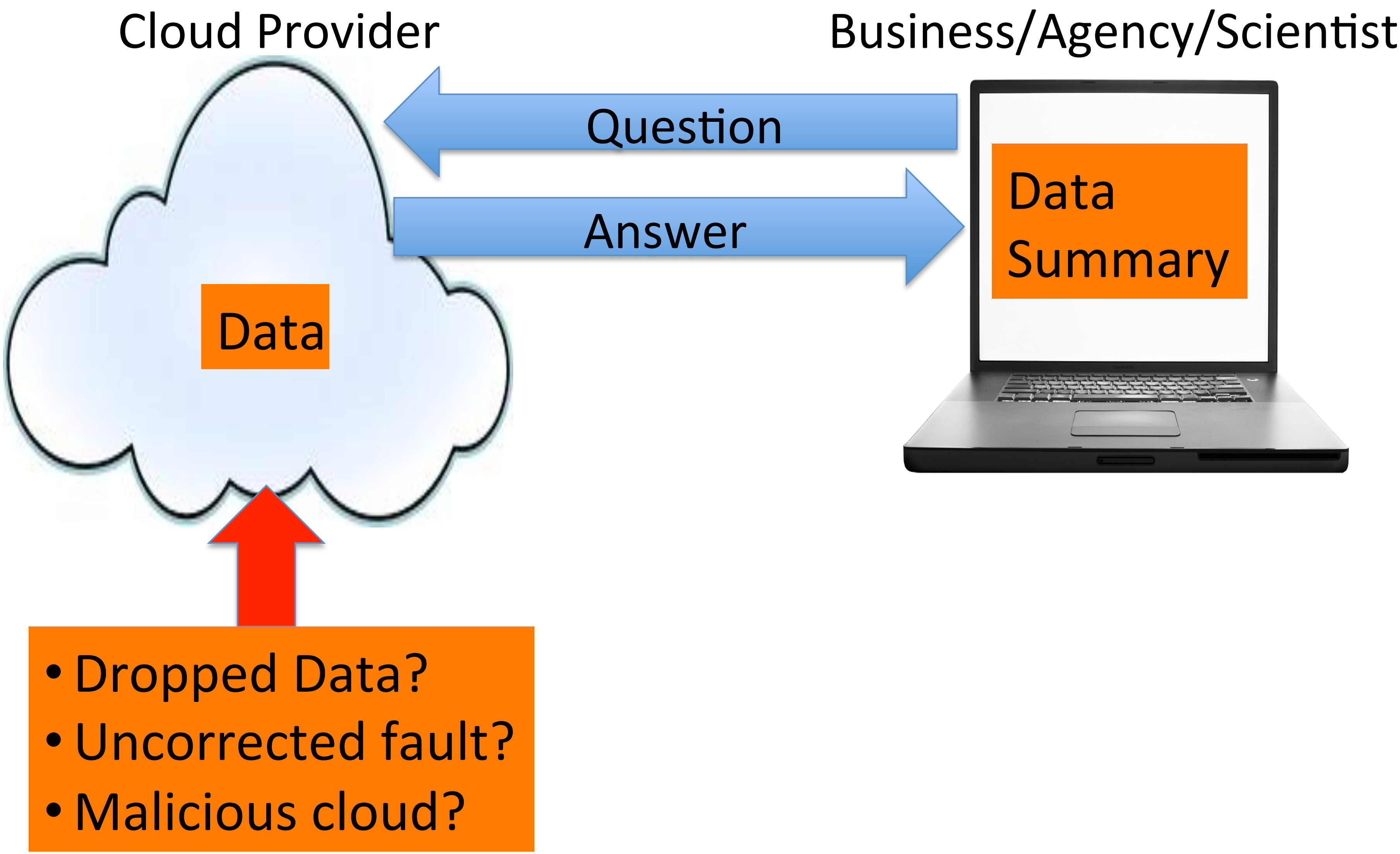
Cloud Provider

Business/Agency/Scientist

Cloud Computing



Cloud Computing



AWS Customer Agreement

WE… MAKE **NO REPRESENTATIONS** OF ANY KIND … THAT THE SERVICE OR THIRD PARTY CONTENT WILL BE UNINTERRUPTED, **ERROR FREE** OR FREE OF HARMFUL COMPONENTS, OR THAT ANY CONTENT … WILL BE SECURE OR **NOT OTHERWISE LOST OR DAMAGED**.

Goals of Verifiable Computation

1. Provide user with guarantee of correctness.
   - Ideally user not do (much) more work than just **read the input**.
   - Ideally cloud will not do much more than just **solve the problem**.

2. Achieve security against malicious clouds, but lightweight for use in benign settings.

Possible Approaches

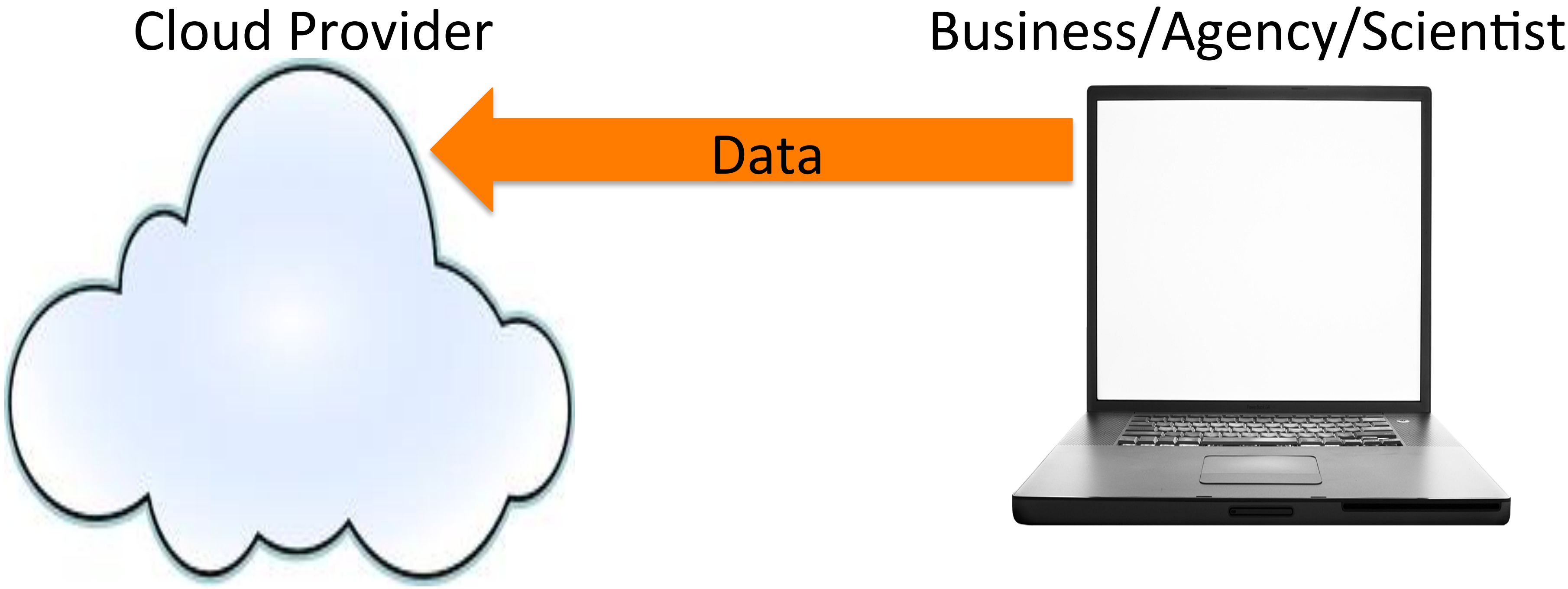
1. Make strong assumptions.
   - Replication [ACKLW02, HKD07,…] assumes majority of responses are correct.
   - Trusted hardware [JSM01, CGJ+09, SSW10…]
2. Make **minimal assumptions**.
   - Interactive proofs (this part of the talk).
   - Argument systems (use cryptography).
3. Use two or more clouds.
   1. Refereed games: assumes 1 cloud is honest.
   2. Multi-Prover Interactive Proofs: assumes clouds cannot communicate with each other.

Interactive Proofs

Cloud Provider

Business/Agency/Scientist

Interactive Proofs

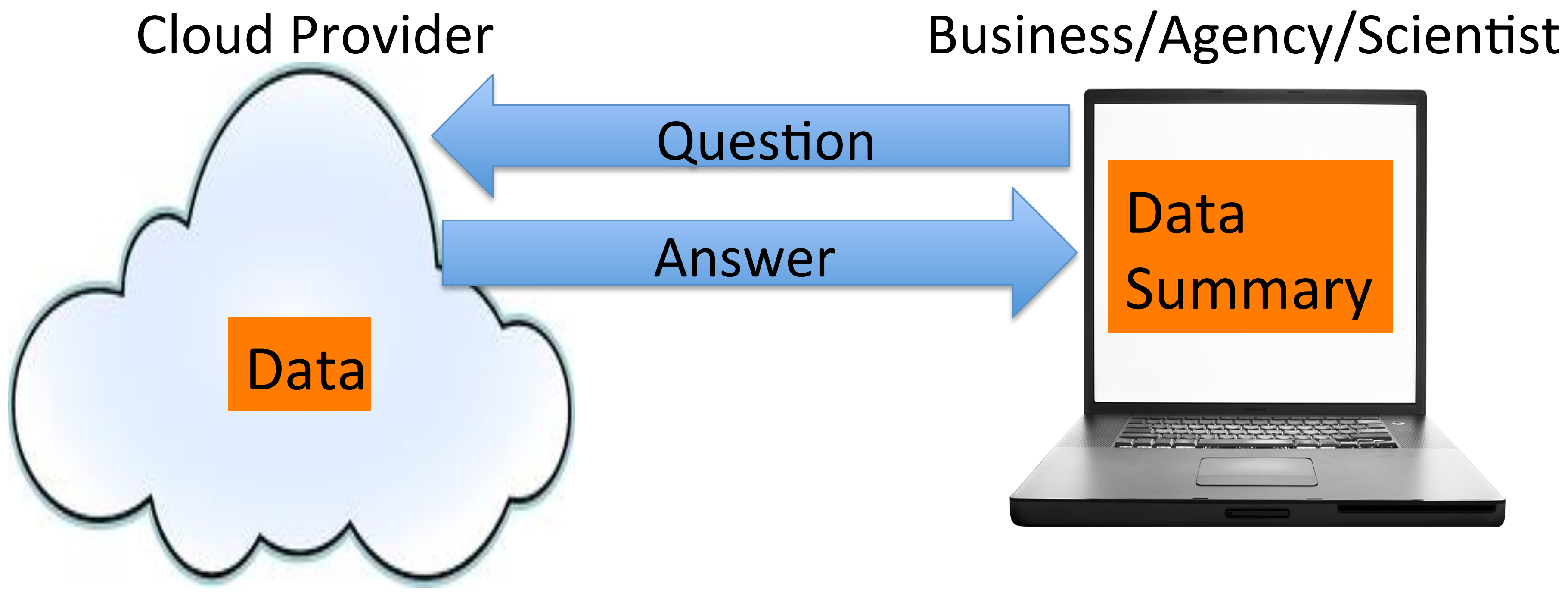


Interactive Proofs

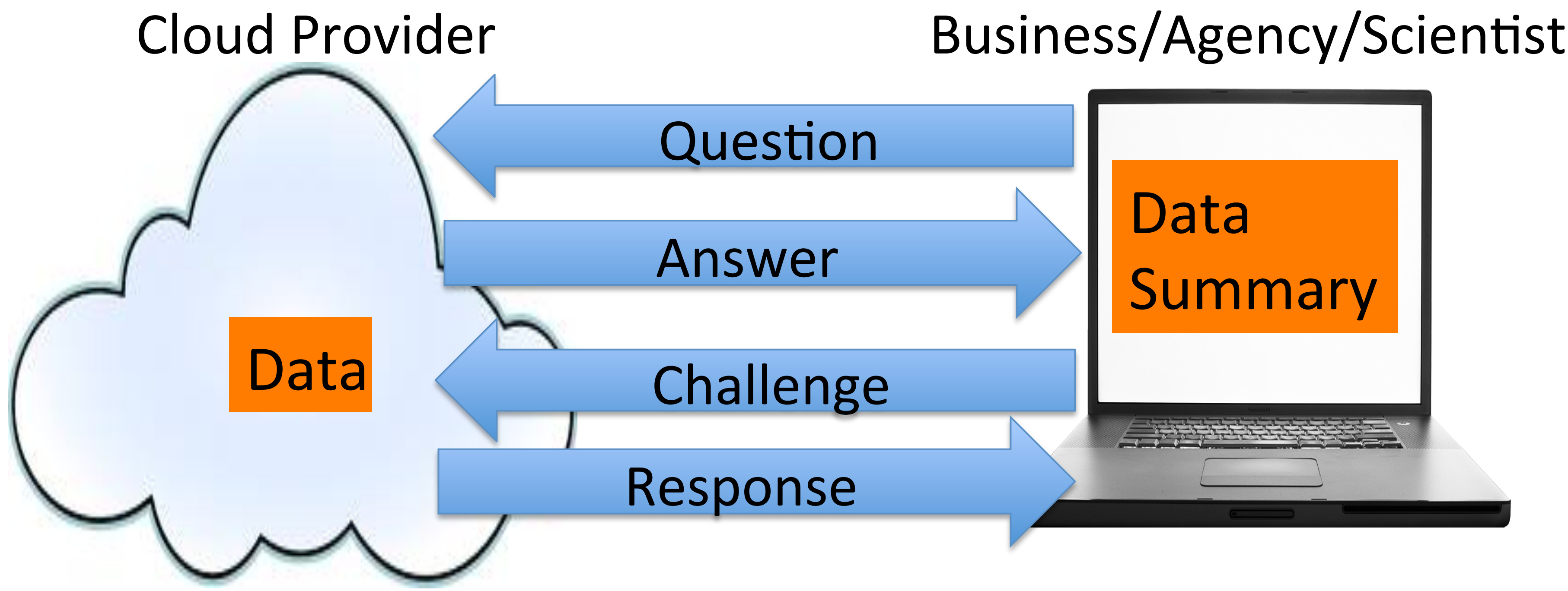
Cloud Provider

Business/Agency/Scientist

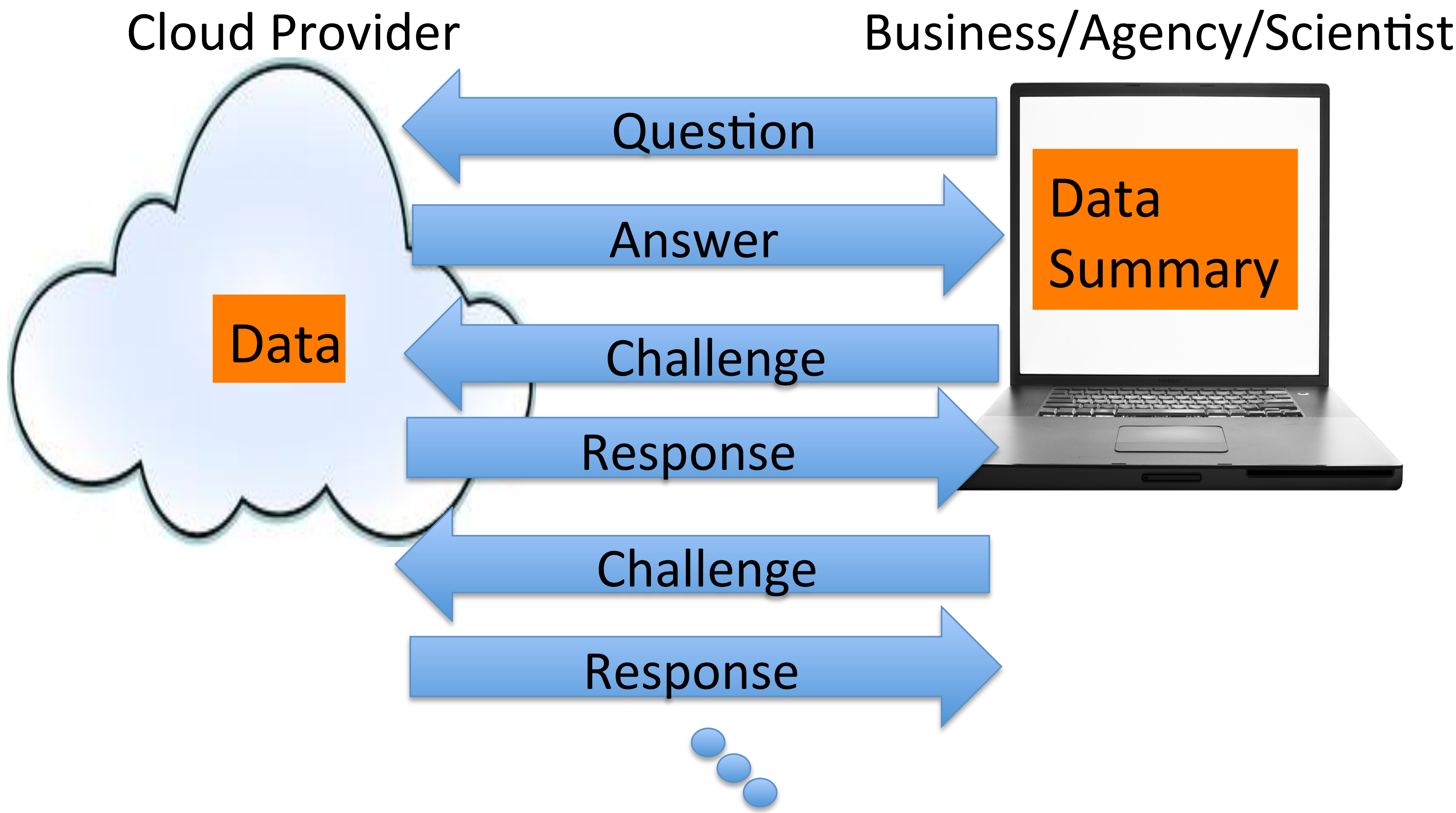
Interactive Proofs



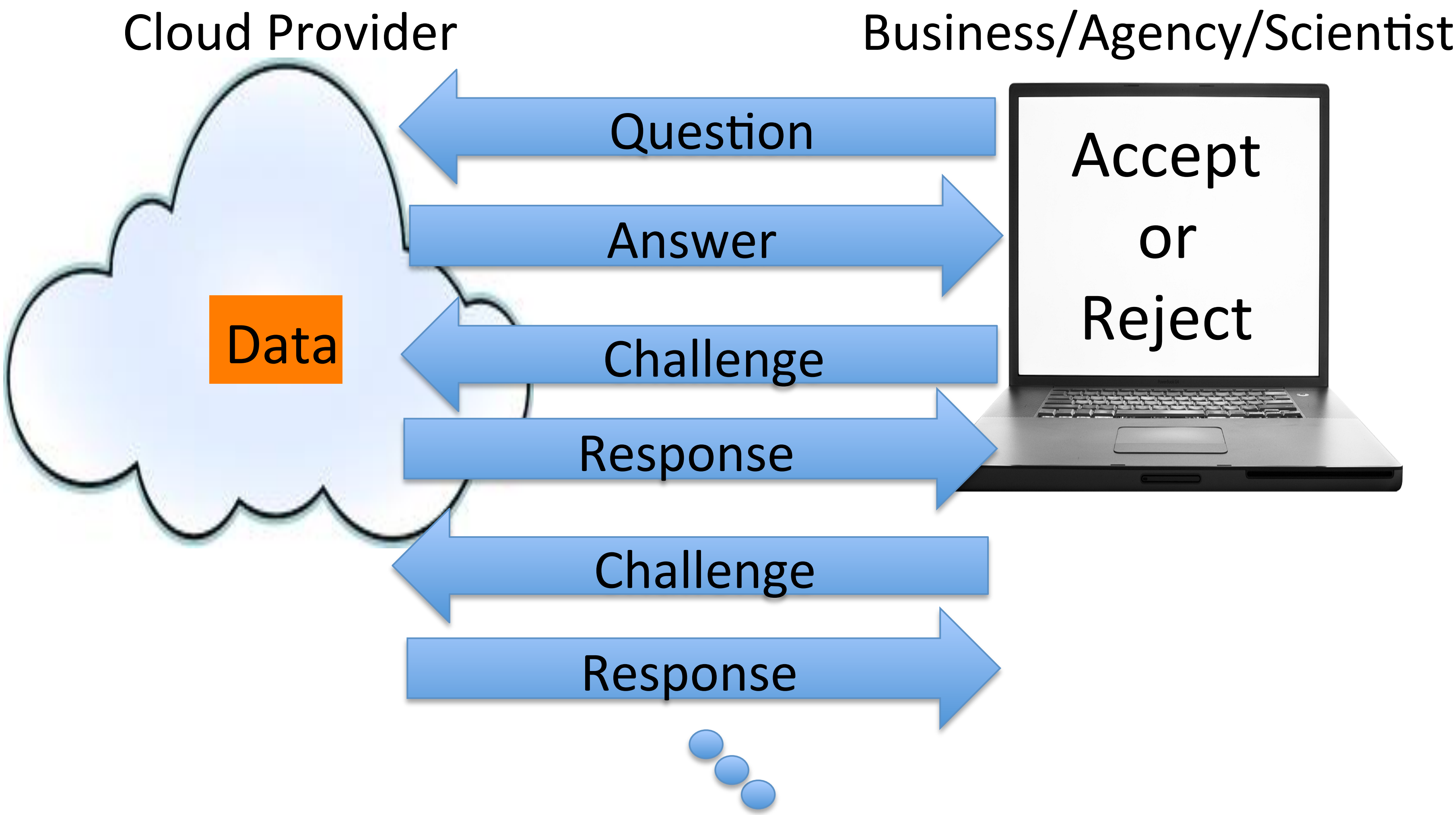
Interactive Proofs



Interactive Proofs



Interactive Proofs



Interactive Proofs

- Prover P and Verifier V.

- P solves problem, tells V the answer.
  - Then P and V have a conversation.
  - P's goal: convince V the answer is correct.

- Requirements:
  - 1. Completeness: an honest P can convince V to accept.
  - 2. Soundness: V will catch a lying P with high probability (secure even if P is computationally unbounded).

A Brief History of Interactive Proofs

Interactive Proofs, Pre-2008

- 1985: Introduced by [GMR, Babai].
  - IPs were believed to be just slightly more powerful than classical static (i.e., NP) proofs.
  - i.e. let **IP** denote class of problems solvable by an interactive proof with a poly-time verifier. It was believed that **IP** ≈ **NP**.

Interactive Proofs, Pre-2008

- 1985: Introduced by [GMR, Babai].
  - IPs were believed to be just slightly more powerful than classical static (i.e., NP) proofs.
  - i.e. let **IP** denote class of problems solvable by an interactive proof with a poly-time verifier. It was believed that **IP** ≈ **NP**.
- 1990: [LFKN, Shamir] proved that **IP=PSPACE**.
  - i.e., IPs with a poly-time verifier can actually solve **much** more difficult problems than can classical static proofs.

Interactive Proofs, Pre-2008

- 1985: Introduced by [GMR, Babai].
  - IPs were believed to be just slightly more powerful than classical static (i.e., NP) proofs.
  - i.e. let **IP** denote class of problems solvable by an interactive proof with a poly-time verifier. It was believed that **IP** ≈ **NP**.
- 1990: [LFKN, Shamir] proved that **IP=PSPACE**.
  - i.e., IPs with a poly-time verifier can actually solve **much** more difficult problems than can classical static proofs.
  - But IPs were still viewed as impractical.
  - Main reason: P's runtime.
    - When applying IPs of [LFKN, Shamir] even to very simple problems, the honest prover would require **superpolynomial** time.

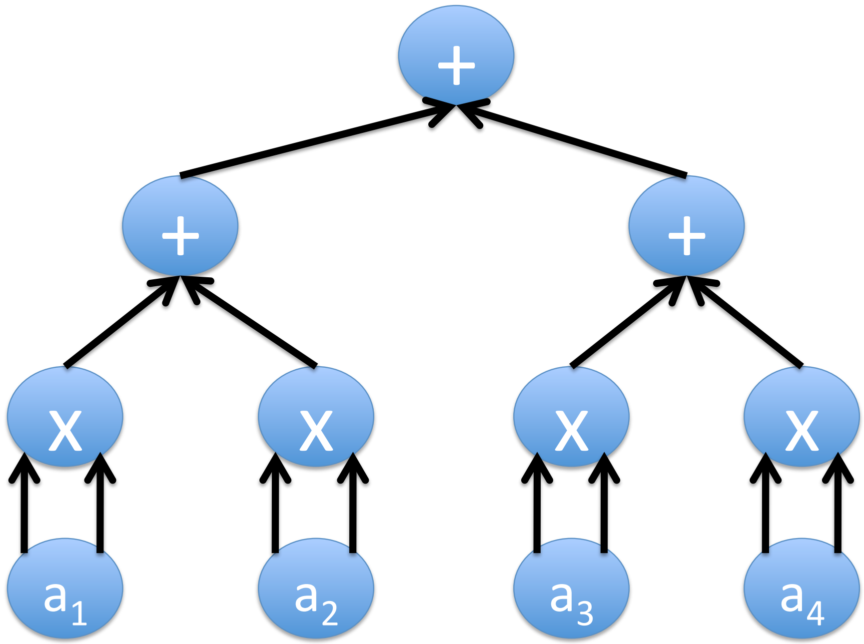
Interactive Proofs, Post-2008

- 2008: [GKR] addressed P's runtime.
  - They gave an IP for any function computed by an efficient **parallel** algorithm.
  - P runs in polynomial time.
  - V runs in (almost) linear time, so outsourcing is useful even though problems are "easy".

Interactive Proofs, Post-2008

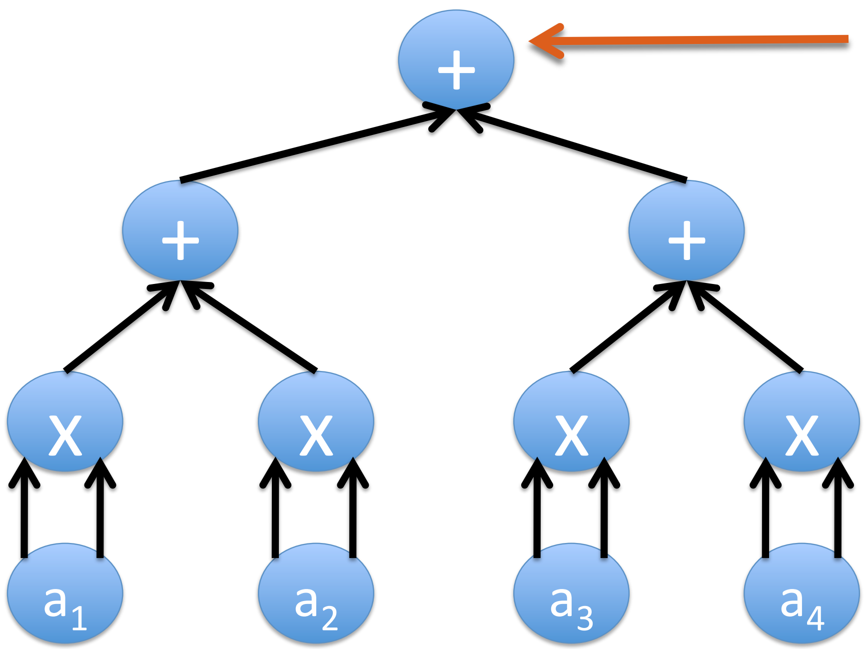
- 2008: [GKR] addressed P's runtime.
  - They gave an IP for any function computed by an efficient **parallel** algorithm.
  - P runs in polynomial time.
  - V runs in (almost) linear time, so outsourcing is useful even though problems are "easy".
- But GKR is not practical out of the box.
  - P still requires a lot of time (**cubic** blowup in runtime).

The GKR Protocol: Overview



F_2 circuit

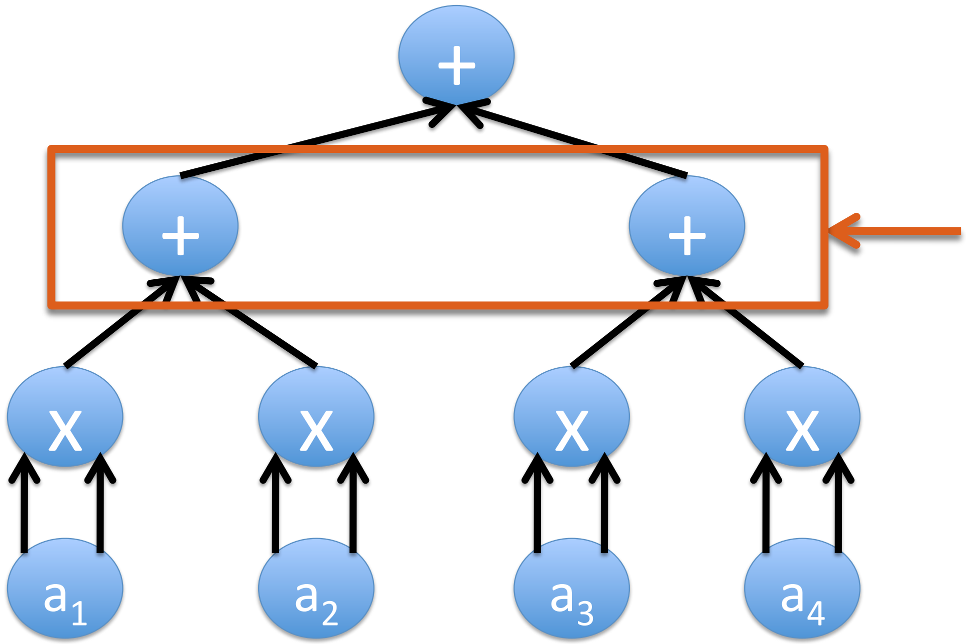
The GKR Protocol: Overview



P starts the conversation with an answer (output).

F_2 circuit

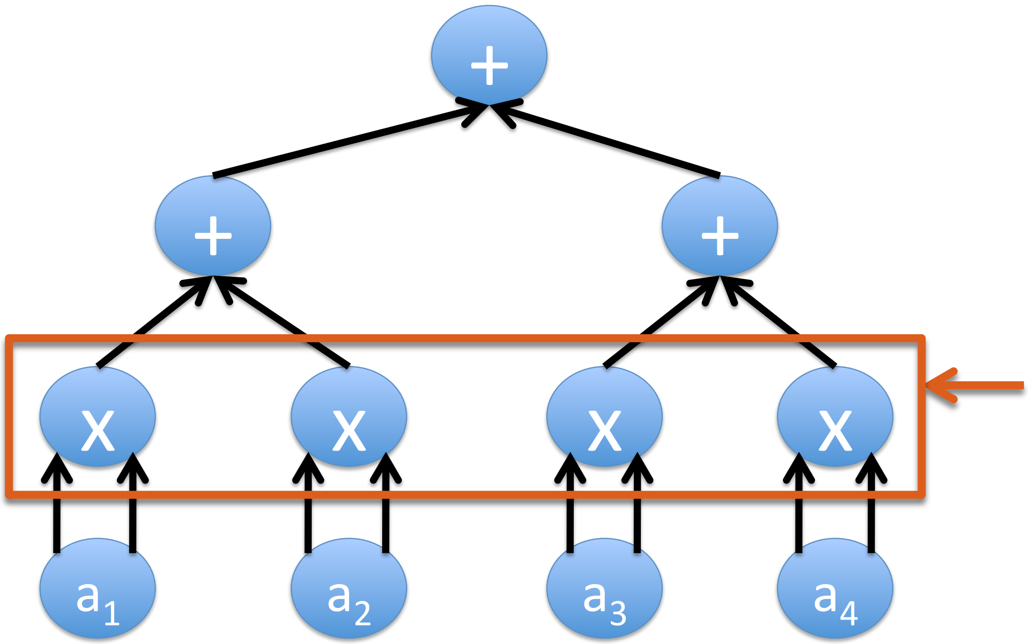
The GKR Protocol: Overview



F_2 circuit

V sends series of challenges. P responds with info about next circuit level.

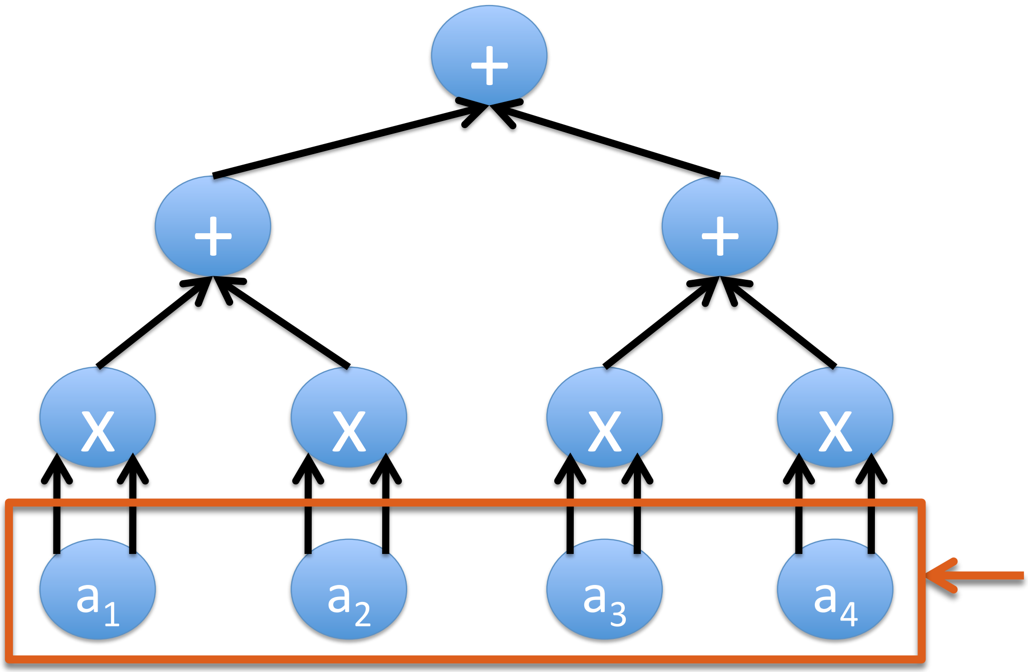
The GKR Protocol: Overview



Challenges continue, layer by layer down to the the input.

F_2 circuit

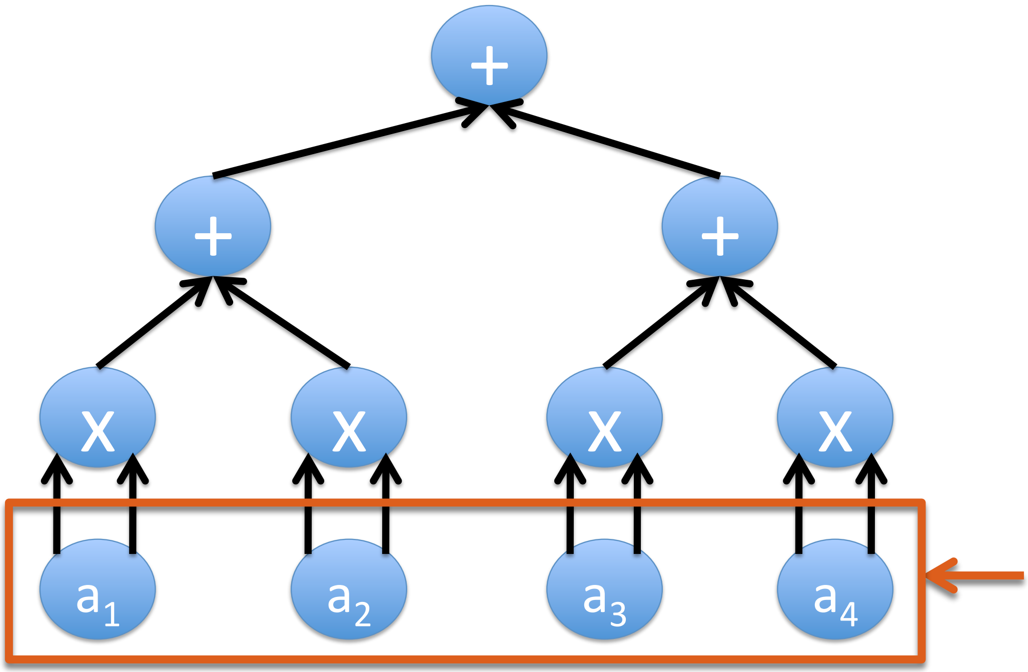
The GKR Protocol: Overview



F_2 circuit

Finally, P says something about the (multilinear extension of the) input.

The GKR Protocol: Overview



F_2 circuit

Finally, P says something about the (multilinear extension of the) input.

V sees input directly, so can check P's final statement directly.

Interactive Proofs, Post-2008

- 2012: [CMT] implemented the GKR protocol (with refinements).
- Demonstrated low concrete costs for V.
- Brought P's runtime down from $\Omega(S^3)$, to $O(S \log S)$, where S is circuit size.
  - Key insight: use **multilinear** extension of circuit within the protocol.
  - Causes enormous cancellation in P's messages, allowing fast computation.

Interactive Proofs, Post-2008

- 2012: [CMT] implemented the GKR protocol (with refinements).
- Demonstrated low concrete costs for V.
- Brought P's runtime down from $\Omega(S^3)$, to O(S log S), where S is circuit size.
  - Key insight: use **multilinear** extension of circuit within the protocol.
  - Causes enormous cancellation in P's messages, allowing fast computation.
- Still not good enough on its own.
  - P is $\sim 10^3$ times slower than just evaluating the circuit.
  - Naïve implementation of GKR would take trillions of times longer.

Interactive Proofs, Post-2008

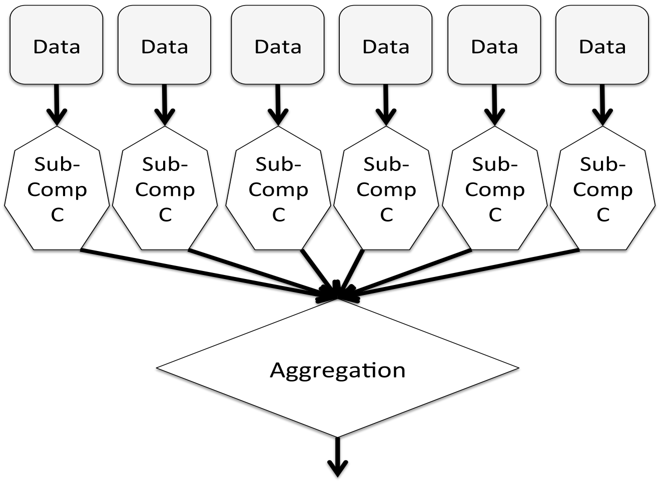
- 2012: [CMT] implemented the GKR protocol (with refinements).
- Demonstrated low concrete costs for V.
- Brought P's runtime down from $\Omega(S^3)$, to O(S log S), where S is circuit size.
  - Key insight: use **multilinear** extension of circuit within the protocol.
  - Causes enormous cancellation in P's messages, allowing fast computation.
- Still not good enough on its own.
  - P is ~10^3 times slower than just evaluating the circuit.
  - Naïve implementation of GKR would take trillions of times longer.
  - Both P and V can be sped up 40x-100x using GPUs [TRMP12].

Interactive Proofs, Post-2008

- 2013: [Thaler] brought P's runtime down to O(S), where S is circuit size, for any circuit that exhibits **repeated structure.**
  - Includes any **data parallel** computation.
- Experimentally yields a prover just 10x slower than a C++ program that evaluates the circuit gate-by-gate.

Interactive Proofs, Post-2008

- 2013: [Thaler] brought P's runtime down to O(S), where S is circuit size, for any circuit that exhibits **repeated structure.**
  - Includes any **data parallel** computation.
- Experimentally yields a prover just 10x slower than a C++ program that evaluates the circuit gate-by-gate.



Interactive Proofs, Post-2008

- 2013: [Thaler] brought P's runtime down to O(S), where S is circuit size, for any circuit that exhibits **repeated structure.**
  - Includes any **data parallel** computation.
- Experimentally yields a prover just 10x slower than a C++ program that evaluates the circuit gate-by-gate.

Problem	P time [CMT12]	P time [T13]	Circuit Eval Time	V time [Both]	Protocol Comm [T13]	Rounds [T13]
DISTINCT ($n=2^{20}$)	56.6 minutes	17.2 s	1.88 s	.03 s	40.7 KB	236

Interactive Proofs in Context: Related Work on Argument Systems

Work on Argument Systems (We'll See Them Later)

- Substantial body of recent work implements argument systems **with pre-processing** for circuit evaluation.
  - [SMBW12, SVP+12, B-SCGT13, GGPR13, SVB+13, PHGR13, BFR+13, B-SCGT+13, B-SCTV14, WSRBW15, CFH+15, …]
- Advantages of our approach:
  - Secure against computationally unbounded provers.
  - No or minimal pre-processing for large classes of computation.
  - Unmatched prover efficiency when applicable.
- Advantages of arguments:
  - Applicable to "deep" circuits.
  - Support for "non-deterministic circuits".
  - Crypto properties: public verifiability, zero-knowledge, etc.

Comparison to Argument Systems, Cont'd

- [WHGSW16] have implemented our interactive proofs in hardware.
  - Motivation: Protecting against Hardware Trojans.
- They chose our interactive proofs instead of argument systems because of advantages not mentioned on previous slide.
  - IPs **do not require crypto** operations (expensive; hard to implement in hardware).
  - Our IPs permit **superior parallelization** for both P and V.
  - Our IPs have **highly local data flows**.
    - Existing argument systems require P to perform FFTs on vectors as large as the circuit being verified, and all "parts" of the prover algorithm must touch these vectors.

Interactive Proof Techniques: Preliminaries

Schwartz-Zippel Lemma

- Informally: any two distinct low-degree polynomials over **F** disagree at a randomly chosen input with high probability.

- Formally: let $g_1 \neq g_2$ be d-variate polynomials over field $\mathbf{F}$. Then

$$\Pr[g_1(r_1,\ldots,r_d) = g_2(r_1,\ldots,r_d)] \leq \max(\deg(g_1),\deg(g_2))/|\mathbf{F}|$$

when each r_i is chosen at random from $\mathbf{F}$.

Low-Degree and Multilinear Extensions

- Definition [**Extensions**]. Given function $f:\{0,1\}^v \to \mathbf{F}$, where $\mathbf{F}$ is a field,a $v-$variate polynomial g over $\mathbf{F}$ is said to **extend** f if $f(x)=g(x)$ for all $x \in \{0,1\}^v$.

- Definition [**Multilinear Extensions].** Any function $f:\{0,1\}^v \to \mathbf{F}$ has a **unique** multilinear extension (MLE), denoted $\widetilde{f}$.

$$f : \{0,1\}^2 \rightarrow \mathbf{F}$$

1	2
8	10

$$\tilde{f} : \mathbf{F}^2 \rightarrow \mathbf{F}$$

1	2	3	4	5	6
8	10	12	14	16	18
15	18	21	24	27	30
22	26	30	34	38	42
29	34	39	44	49	56
36	42	48	54	60	68

A Useful Expression for the MLE

- Lemma (**Lagrange Interpolation**): Let $f:\{0,1\}^v \to \mathbf{F}$. Then as formal polynomials,

$$\text{Equation } (*):\quad \tilde{f}(x) = \sum_{w \in \{0,1\}^{\log n}} f(w) \bullet \tilde{\delta}_w(x),$$

where $\tilde{\delta}_w(x) = \prod_{i=1}^{v} (x_i w_i + (1 - x_i)(1 - w_i))$

is the MLE of the function $\delta_w : \{0,1\}^v \to \mathbf{F}$ defined via:

$\delta_w(y) = 1$ if $y = w$, and $\delta_w(y) = 0$ otherwise.

Evaluating The MLE At Any Point, Efficiently

- Let $f:\{0,1\}^v \to \mathbf{F}$, $r \in \mathbf{F}^v$, and $n = 2^v$. Given as input $f(w)$ for all $w \in \{0,1\}^v$, one can compute $\tilde{f}(r)$ in $O(n \log n)$ time and $O(\log n)$ space with a single streaming pass over the input.

Evaluating The MLE At Any Point, Efficiently

- Let $f:\{0,1\}^v \rightarrow \mathbf{F}$, $r \in \mathbf{F}^v$, and $n = 2^v$. Given as input $f(w)$ for all $w \in \{0,1\}^v$, one can compute $\tilde{f}(r)$ in $O(n \log n)$ time and $O(\log n)$ space with a single streaming pass over the input.

- Proof: By Equation (*): $\tilde{f}(r) = \sum_{w \in \{0,1\}^{\log n}} f(w) \bullet \tilde{\delta}_w(r)$.

Compute RHS by initializing $\tilde{f}(r) \leftarrow 0$ and processing update $(w, f(w))$ via:

$$\tilde{f}(r) \leftarrow \tilde{f}(r) + f(w) \bullet \tilde{\delta}(r).$$

↑

Evaluation takes $O(\log n)$ to compute.

Evaluating The MLE At Any Point, Efficiently

- Let $f:\{0,1\}^v \rightarrow \mathbf{F}$, $r \in \mathbf{F}^v$, and $n = 2^v$. Given as input $f(w)$ for all $w \in \{0,1\}^v$, one can compute $\tilde{f}(r)$ in $O(n \log n)$ time and $O(\log n)$ space with a single streaming pass over the input.

- Proof: By Equation (*): $\tilde{f}(r) = \sum_{w \in \{0,1\}^{\log n}} f(w) \bullet \tilde{\delta}_w(r)$.

  Compute RHS by initializing $\tilde{f}(r) \leftarrow 0$ and processing update $(w, f(w))$ via:

$$\tilde{f}(r) \leftarrow \tilde{f}(r) + f(w) \bullet \tilde{\delta}(r).$$

- Can also reduce runtime to O(n) using dynamic programming, but this requires more space [Vu et al., 2013].

A Final Technical Hammer: Sum-Check Protocol [LFKN90]

Sum-Check Protocol [LFKN90]

- Input: V given oracle access to a d-variate polynomial g over field **F** with $\deg_i(g) = O(1)$ for all $i \in \{1,\ldots,d\}$.
- Goal: compute the quantity:

$$\sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_d \in \{0,1\}} g(b_1,\ldots,b_d)$$

- **Start**: P sends claimed answer C_1.
- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal

$$\sum_{b_2 \in \{0,1\}} \cdots \sum_{b_d \in \{0,1\}} g(X_1, b_2, \ldots, b_d)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks r_1 at random from **F** and lets $C_2 = s_1(r_1)$.
- **Round 2**: V sends r_1 to P. They recursively check that

$$C_2 = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_d \in \{0,1\}} g(r_1, b_2, \ldots, b_d)$$

- **Round d:** P sends **univariate** polynomial $s_d(X_d)$ claimed to equal

$$g(r_1, r_2, \ldots, r_{d-1}, X_d).$$

- V picks r_d at random, checks that $s_d(r_d) = g(r_1, r_2, \ldots, r_d)$.

Costs of Sum-Check Protocol

- P sends d messages, each a univariate polynomial of degree $\deg_i(g) = O(1)$.
- V processes each message in O(1) time, and makes one oracle query to g in final round.
- P computes a sum over up to 2^{d-i} terms in round i. Naively, this requires evaluating g at 2^{d-i} points.

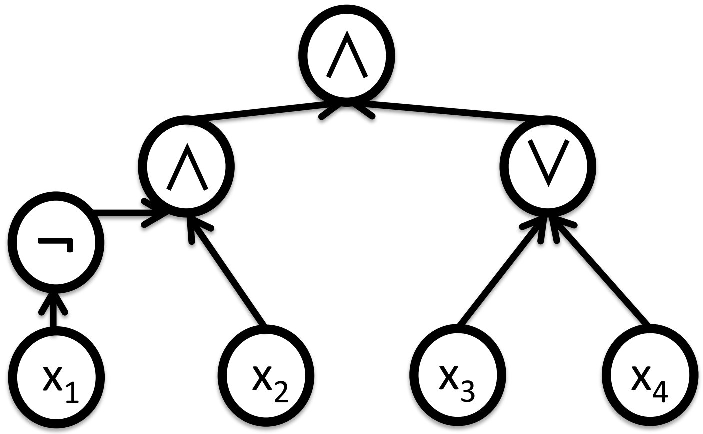
First Application of Sum-Check: An IP For #SAT [LFKN]

#SAT Problem

- Let ϕ be a Boolean formula of size S over n variables.

#SAT Problem

- Let ϕ be a Boolean formula of size S over n variables.



#SAT Problem

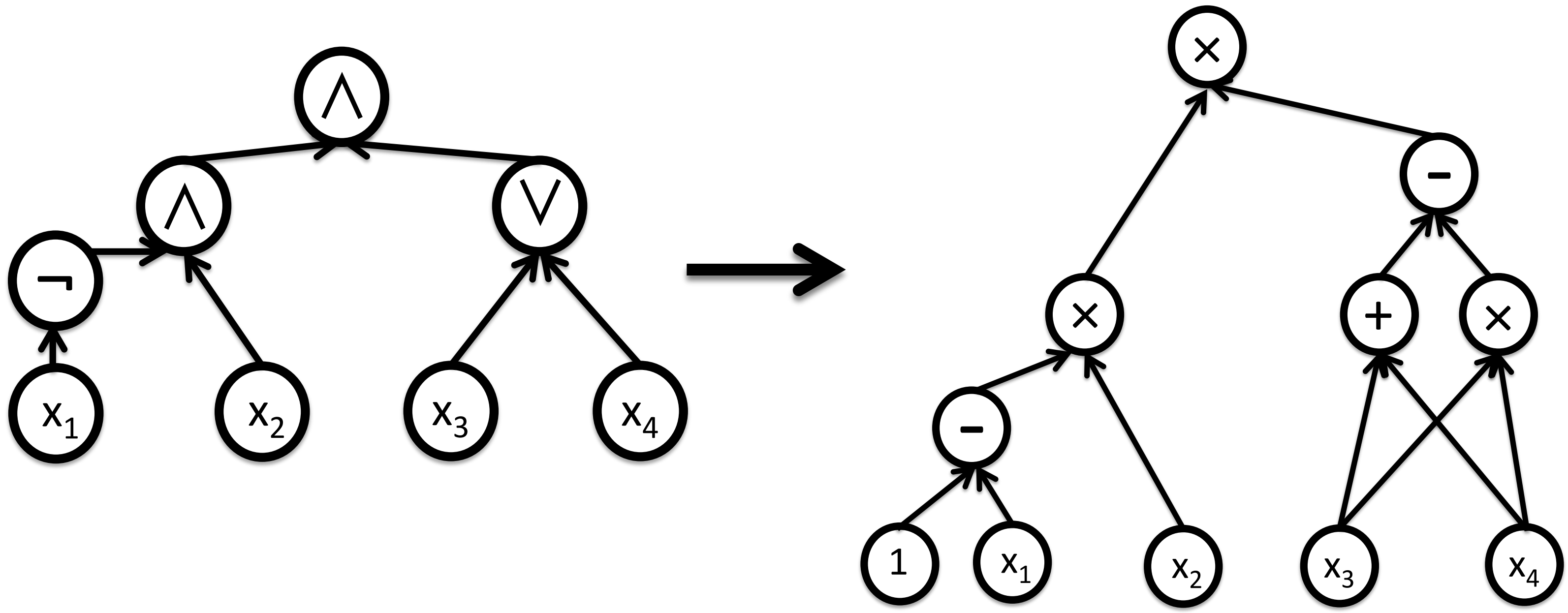
- Let ϕ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x \in \{0,1\}^w} \phi(x)$.

#SAT Problem

- Let ϕ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x \in \{0,1\}^w} \phi(x)$.
- Protocol: Apply sum-check to an extension polynomial g of ϕ.
  - Note: in final round, V needs to compute g(r) for some randomly chosen r in $\mathbf{F}^n$.

#SAT Problem

- Let ϕ be a Boolean formula of size S over n variables.
- Goal: Compute $\sum_{x \in \{0,1\}^w} \phi(x)$.
- Protocol: Apply sum-check to an extension polynomial g of ϕ.
  - Note: in final round, V needs to compute g(r) for some randomly chosen r in $\mathbf{F}^n$.
- Where does g come from? Arithmetize ϕ.
  - i.e., replace ϕ with arithmetic circuit computing extension g of ϕ.
    - $AND(y_1, y_2) \Longrightarrow$ multiplication gate $y_1 * y_2$.
    - $NOT(y_1) \Longrightarrow 1 - y_1$
    - $OR(y_1, y_2) \Longrightarrow y_1 + y_2 - y_1 * y_2$.
  - Total degree of g is at most S, and V can evaluate g(r) gate-by-gate in time O(S).



Transforming a Boolean circuit ϕ into an arithmetic circuit computing an extension of ϕ.

Costs of #SAT Protocol for Φ

- Let ϕ be a Boolean formula of size S over n variables.

Rounds	Communication	V Time	P Time
n	P sends a degree S polynomial in reach round $\Longrightarrow$ O(S*n) field elements sent in total.	•O(S) time to process each of the n messages of P •O(S) time to evaluate g(r) $\Longrightarrow$ O(S*n) time total	P must evaluate g at O(2^n) points to determine each message $\Longrightarrow$ O(S*n*2^n) time in total.

Second Application: An Optimal Interactive Proof For Matrix Multiplication

[Thaler13]: Optimal IP For n x n MatMult

- Goal: Given n x n matrices A, B over field **F**, compute C=A*B.

[Thaler13]: Optimal IP For n x n MatMult

- Goal: Given n x n matrices A, B over field **F**, compute C=A*B.
- P simply determines the "right answer", and then P does $O(n^2)$ extra work to prove its correctness.
- Doesn't matter how P obtains the right answer!
- Optimal runtime up to leading constant assuming no $O(n^2)$ time algorithm for MatMult.
- V runs in linear time (which is also optimal).

[Thaler13]: Optimal IP For n x n MatMult

- Goal: Given n x n matrices A, B over field **F**, compute C=A*B.
- P simply determines the "right answer", and then P does $O(n^2)$ extra work to prove its correctness.
- Doesn't matter how P obtains the right answer!
- Optimal runtime up to leading constant assuming no $O(n^2)$ time algorithm for MatMult.
- V runs in linear time (which is also optimal).

Problem Size	Naïve MatMult Time	Additional P time	V Time	Rounds	Protocol Comm
1024 x 1024	2.17 s	0.03 s	0.09 s	11	264 bytes
2048 x 2048	18.23 s	0.13 s	0.30 s	12	288 bytes

Comparison to Freivalds' Algorithm

- Freivalds (MFCS, 1979) gave the following protocol for MatMult. To check AB=C:
  - V picks random vector x.
  - Accepts if A*(Bx) = Cx.
  - **No** extra work for P, $O(n^2)$ time for V.

Comparison to Freivalds' Algorithm

- Freivalds (MFCS, 1979) gave the following protocol for MatMult. To check AB=C:
  - V picks random vector x.
  - Accepts if A*(Bx) = Cx.
  - **No** extra work for P, $O(n^2)$ time for V.
- Our big win: verifying algorithms that invoke MatMult, but aren't really interested in matrices.
  - E.g. Best-known graph diameter algorithms square the adjacency matrix, but are only interested in a single number.
  - Total communication for us is $O(\log^2 n)$, Freivalds' is $\Omega(n^2)$.

MatMult Protocol: Technical Details

Notation

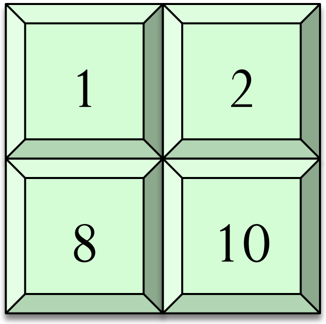
- Given $n \times n$ input matrices A, B over field $\mathbf{F}$, interpret A and B as functions mapping $\{0, 1\}^{\log n} \times \{0, 1\}^{\log n}$ to $\mathbf{F}$ via:

$$A(i_1,\ldots,i_{\log n}, j_1,\ldots, j_{\log n}) = A_{ij}.$$

- Let C=A*B denote the true answer.

- Let $\widetilde{A}, \widetilde{B} : \mathbf{F}^{\log n} \times \mathbf{F}^{\log n} \rightarrow \mathbf{F}$ denote the multilinear extensions of the functions A and B.

$$D:\{0,1\}^2 \rightarrow \mathbf{F}$$



$$\widetilde{D}:\mathbf{F}^2 \rightarrow \mathbf{F}$$

1	2	3	4	5	6
8	10	12	14	16	18
15	18	21	24	27	30
22	26	30	34	38	42
29	34	39	44	49	56
36	42	48	54	60	68

$$\widetilde{D} : \mathbf{F}^2 \rightarrow \mathbf{F}$$

1	2	3	4	5	6
8	10	12	14	16	18
15	18	21	24	27	30
22	26	30	34	38	42
29	34	39	44	49	56
36	42	48	54	60	68

MatMult Protocol

- P sends a matrix D claimed to equal C=A*B.
- V evaluates $\tilde{D}$ at a random point $(\mathbf{r}_1, \mathbf{r}_2) \in \mathbf{F}^{\log n} \times \mathbf{F}^{\log n}$.
- By Schwartz-Zippel: it is safe for V to believe that D equals the correct answer C as long as:

$$\tilde{D}(\mathbf{r}_1, \mathbf{r}_2) = \tilde{C}(\mathbf{r}_1, \mathbf{r}_2).$$

- Goal becomes: compute $\tilde{C}(\mathbf{r}_1, \mathbf{r}_2)$.

MatMult Protocol

- Goal: Compute $\tilde{C}(\mathbf{r}_1, \mathbf{r}_2)$.
- For Boolean vectors $\mathbf{i}, \mathbf{j} \in \{0,1\}^{\log n}$, clearly:

$$C(\mathbf{i}, \mathbf{j}) = \sum_{\mathbf{k} \in \{0,1\}^{\log n}} A(\mathbf{i}, \mathbf{k}) B(\mathbf{k}, \mathbf{j}).$$

- This implies the following **polynomial** identity:

$$\tilde{C}(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{b} \in \{0,1\}^{\log n}} \tilde{A}(\mathbf{x}, \mathbf{b}) \tilde{B}(\mathbf{b}, \mathbf{y}).$$

- So V applies sum-check protocol to compute

$$\tilde{C}(\mathbf{r}_1, \mathbf{r}_2) = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(b_1, \ldots, b_{\log n}),$$

where $g(\mathbf{z}) = \tilde{A}(\mathbf{r}_1, \mathbf{z}) * \tilde{B}(\mathbf{z}, \mathbf{r}_2)$.

Making V Fast

- At end of sum-check, V must evaluate $g(\mathbf{r}_3) = \widetilde{A}(\mathbf{r}_1, \mathbf{r}_3) \bullet \widetilde{B}(\mathbf{r}_3, \mathbf{r}_2)$.
- Suffices to evaluate $\widetilde{A}(\mathbf{r}_1, \mathbf{r}_3)$ and $\widetilde{B}(\mathbf{r}_3, \mathbf{r}_2)$. How?

Making V Fast

- At end of sum-check, V must evaluate $g(\mathbf{r}_3) = \widetilde{A}(\mathbf{r}_1,\mathbf{r}_3) \bullet \widetilde{B}(\mathbf{r}_3,\mathbf{r}_2)$.
- Suffices to evaluate $\widetilde{A}(\mathbf{r}_1,\mathbf{r}_3)$ and $\widetilde{B}(\mathbf{r}_3,\mathbf{r}_2)$. How?
- Can be done in $O(n^2)$ time by "Fast Evaluation of MLE" lemma in preliminaries.

Making P Fast

- Recall: using sum-check to compute $\sum_{k \in \{0,1\}^{\log n}} g(k_1, \ldots, k_{\log n})$.
- Round i: P sends quadratic polynomial $s_i(X_i)$ claimed to equal:

$$\sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \ldots, r_{3,i-1}, X_i, b_{i+1} \ldots, b_{\log n}).$$

- Suffices for P to specify $s_i(0)$, $s_i(1)$, and $s_i(2)$.
- Thus: **Enough to evaluate g at all points of the form**

$$(r_{3,1}, \ldots, r_{3,i-1}, \{0,1,2\}, b_{i+1} \ldots, b_{\log n}) : (b_{i+1} \ldots, b_{\log n}) \in \{0,1\}^{\log n - i}.$$

Making P Fast

- **Enough to evaluate g at all points of the form:**

$$(r_{3,1},\ldots,r_{3,i-1},\{0,1,2\},b_{i+1}\ldots,b_{\log n}):(b_{i+1}\ldots,b_{\log n})\in\{0,1\}^{\log n-i}.$$

Making P Fast

- **Enough to evaluate g at all points of the form:**

  $$(r_{3,1},\ldots,r_{3,i-1},\{0,1,2\},b_{i+1}\ldots,b_{\log n}):(b_{i+1}\ldots,b_{\log n})\in\{0,1\}^{\log n-i}.$$

- Already showed: Can evaluate g at any point in $O(n^2)$ time.
  - By determining the contribution of each matrix entry A_{ij}, B_{ij} independently.
- So $O(n*n^2)=O(n^3)$ **total** time. Can we improve this?

Making P Fast

- **Enough to evaluate g at all points of the form:**

  $(r_{3,1},\ldots,r_{3,i-1},\{0,1,2\},b_{i+1}\ldots,b_{\log n}):(b_{i+1}\ldots,b_{\log n})\in\{0,1\}^{\log n-i}.$

- Already showed: Can evaluate g at any point in $O(n^2)$ time.
  - By determining the contribution of each matrix entry A_{ij}, B_{ij} independently
- So $O(n*n^2)=O(n^3)$ **total** time. Can we improve this?
- Yes: each entry A_{ij} contributes to $g(r_{3,1},\ldots,r_{3,i-1},\{0,1,2\},b_{i+1}\ldots,b_{\log n})$ for only **one** tuple $(b_{i+1}\ldots,b_{\log n})\in\{0,1\}^{\log n-i}$.

Making P Fast

- $\widetilde{A}(\mathbf{r}_1,\mathbf{z}) = \sum_{\mathbf{i},\mathbf{j}\in\{0,1\}^{\log n}} \widetilde{A}_{\mathbf{ij}}\,\delta_{(\mathbf{i},\mathbf{j})}(\mathbf{r}_1,\mathbf{z}).$

- Only interested in $\mathbf{z}$'s of the form

$$\mathbf{z} = (r_{3,1},\ldots,r_{3,i-1},\{0,1,2\},b_{i+1}\ldots,b_{\log n}).$$

- Claim: $\widetilde{\delta}_{(\mathbf{i},\mathbf{j})}(\mathbf{r}_1,\mathbf{z}) = 0$ unless $(j_{i+1}\ldots,j_{\log n}) = (b_{i+1}\ldots,b_{\log n})$

Implementing P Quickly

- Summary: In round i, P must evaluate g at $n/2^i$ points of a special form (trailing entries are **Boolean**).
- Each matrix entry A_{ij}, B_{ij} contributes to only **one** of these evaluations.
- So P can run in $O(n^2)$ time per round, or $O(n^2 \log n)$ time across all log n rounds.

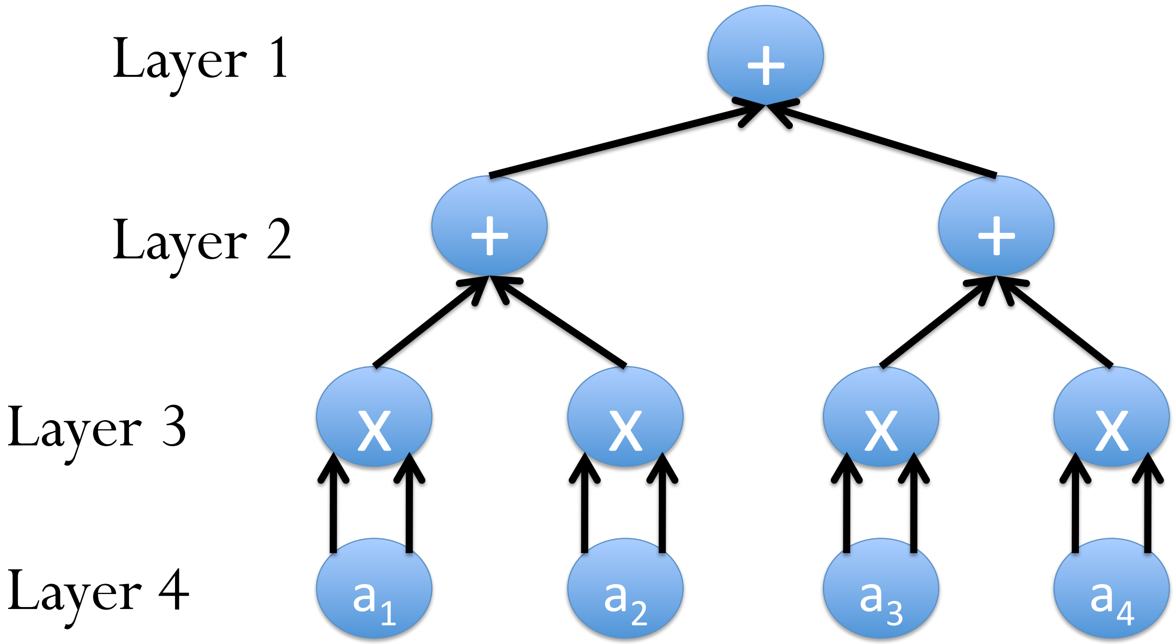
Implementing P Quickly

- With care: can bring P's time down to $O(n^2)$.
- Key idea: **Reuse work** across rounds.
  - If two entries $\mathbf{(i,j),(i',j')} \in \{0,1\}^{\log n} \times \{0,1\}^{\log n}$ agree in their last k bits, then $A_{\mathbf{ij}}$ and $A_{\mathbf{i'j'}}$ contribute to the **same** point in rounds k and up.
  - Can treat (**i,j**) and (**i',j'**) as a single entity thereafter.
  - Only $n/2^k$, entities of interest in round k.
  - Total work across all rounds is proportional to

$$\sum_{1 \le k \le \log n} n/2^k = 2n.$$

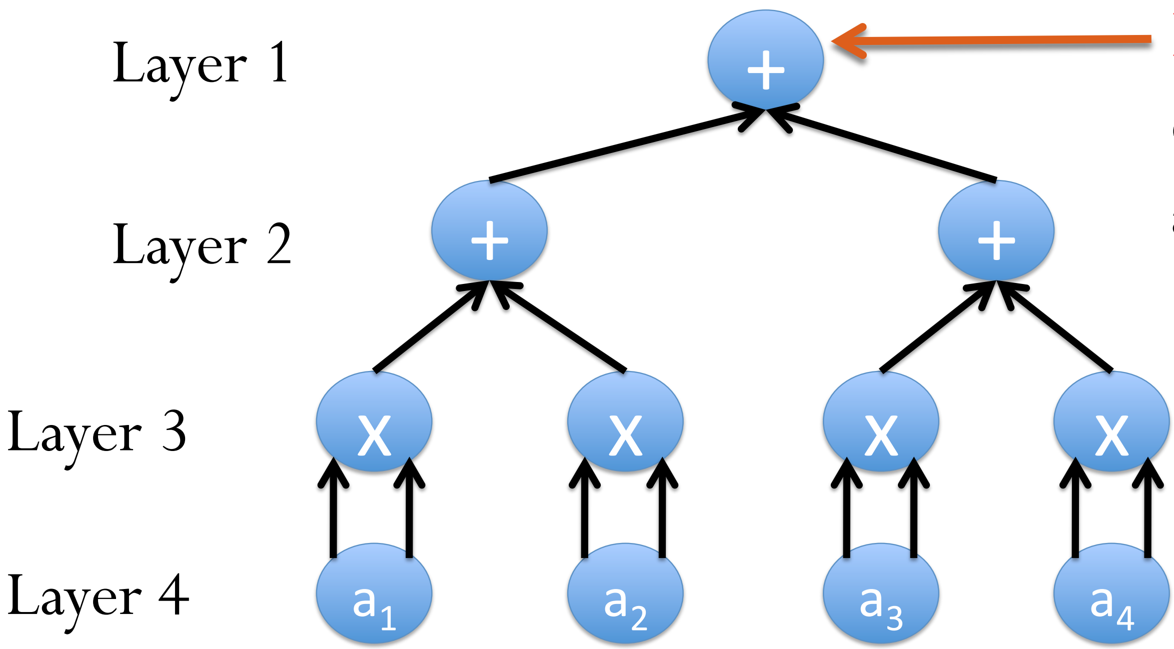
Third Application: The GKR Protocol

The GKR Protocol: Overview



F_2 circuit

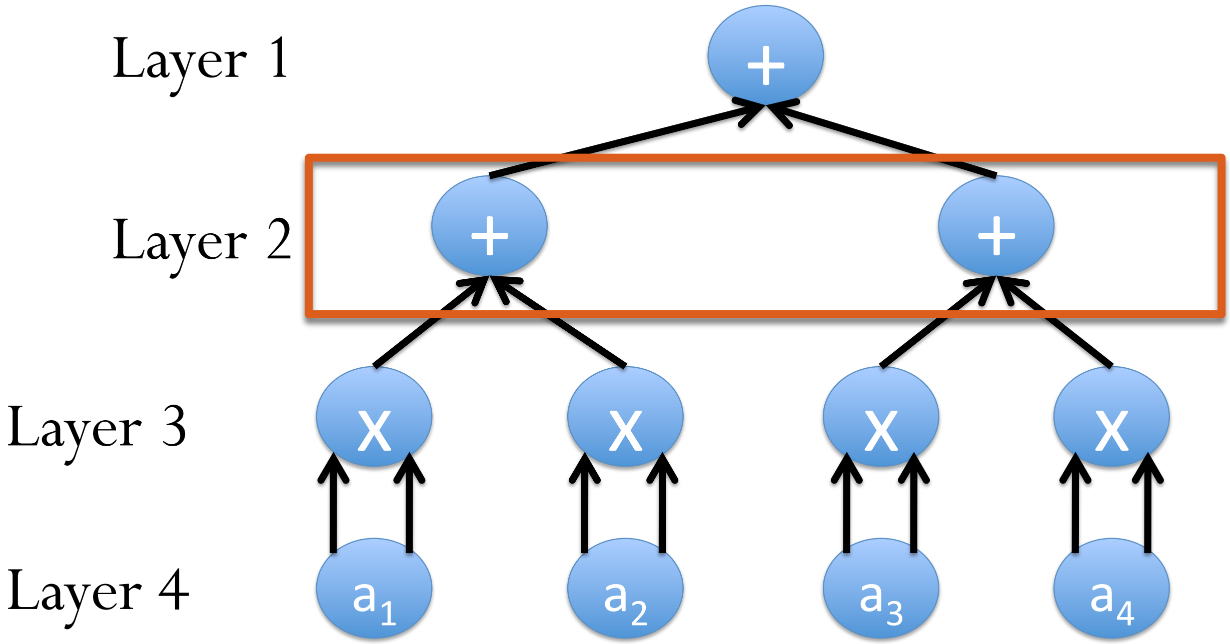
The GKR Protocol: Overview



P starts the conversation with an answer (output).

F_2 circuit

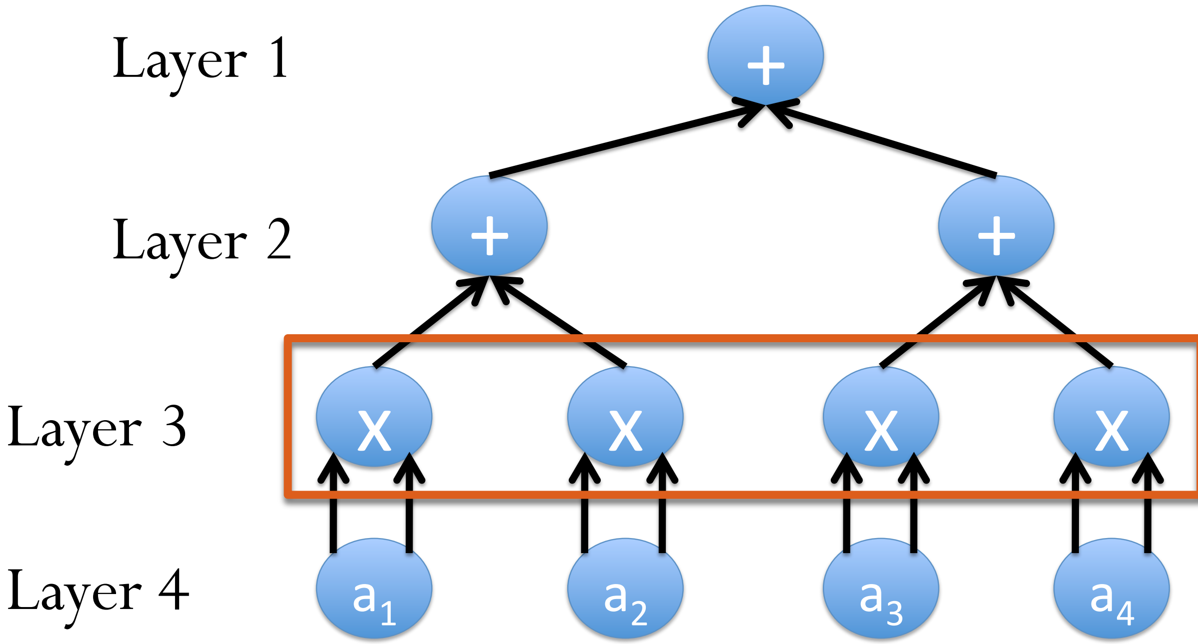
The GKR Protocol: Overview



F_2 circuit

V sends series of challenges. P responds with info about next circuit level.

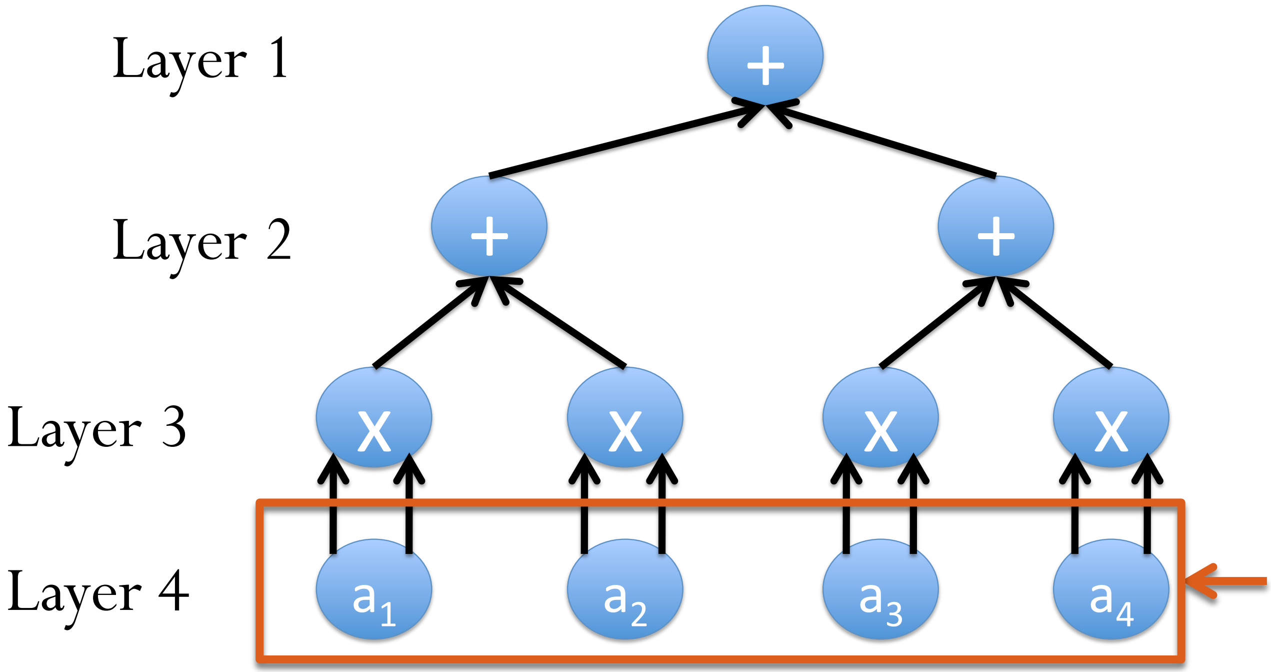
The GKR Protocol: Overview



Challenges continue, layer by layer down to the the input.

F_2 circuit

The GKR Protocol: Overview



F_2 circuit

Finally, P says something about the (multilinear extension of the) input.

Notation

- Assume layers i and i+1 of C have S gates each.
  - Assign each gate a binary label (log S bits).
- Let $W_i(\mathbf{a}):\{0,1\}^{\log S} \rightarrow \mathbf{F}$ output the value of gate **a** at layer i.
- Let $\mathrm{add}_i(\mathbf{a},\mathbf{b},\mathbf{c}):\{0,1\}^{3\log S} \rightarrow \mathbf{F}$ output 1 iff $(\mathbf{b},\mathbf{c})=(\mathrm{in}_1(\mathbf{a}), \mathrm{in}_2(\mathbf{a}))$ and gate **a** is an addition gate.
- Let $\mathrm{mult}_i(\mathbf{a},\mathbf{b},\mathbf{c}):\{0,1\}^{3\log S} \rightarrow \mathbf{F}$ output 1 iff $(\mathbf{b},\mathbf{c})=(\mathrm{in}_1(\mathbf{a}), \mathrm{in}_2(\mathbf{a}))$ and gate **a** is a multiplication gate.

GKR Protocol: Goal of Iteration i

- Iteration i starts with a claim from P about $\tilde{W}_i(\mathbf{r}_1)$ for a random point $\mathbf{r}_1 \in \mathbf{F}^{\log S}$.

- Goal: Reduce this to a claim about $\tilde{W}_{i+1}(\mathbf{r}_2)$ for a random point $\mathbf{r}_2 \in \mathbf{F}^{\log S}$.

- Key Polynomial Identity. The following equality holds as formal polynomials:

$$\tilde{W}_i(\mathbf{a}) = \sum_{\mathbf{b},\mathbf{c} \in \{0,1\}^{\log S}} \widetilde{\mathrm{add}}_i(\mathbf{a},\mathbf{b},\mathbf{c})\left(\tilde{W}_{i+1}(\mathbf{b}) + \tilde{W}_{i+1}(\mathbf{c})\right) + \widetilde{\mathrm{mult}}_i(\mathbf{a},\mathbf{b},\mathbf{c})\left(\tilde{W}_{i+1}(\mathbf{b}) \bullet \tilde{W}_{i+1}(\mathbf{c})\right).$$

GKR Protocol: Goal of Iteration i

- So V applies sum-check protocol to compute

$$\tilde{W}_i(\mathbf{r}_1) = \sum_{\mathbf{b},\mathbf{c}\in\{0,1\}^{\log S}} \widetilde{\mathrm{add}}_i(\mathbf{r}_1,\mathbf{b},\mathbf{c})\left(\tilde{W}_{i+1}(\mathbf{b})+\tilde{W}_{i+1}(\mathbf{c})\right)+\widetilde{\mathrm{mult}}_i(\mathbf{r}_1,\mathbf{b},\mathbf{c})\left(\tilde{W}_{i+1}(\mathbf{b})\bullet\tilde{W}_{i+1}(\mathbf{c})\right).$$

GKR Protocol: Goal of Iteration i

- So V applies sum-check protocol to compute

$$\tilde{W}_i(\mathbf{r}_1) = \sum_{\mathbf{b},\mathbf{c}\in\{0,1\}^{\log S}} \widetilde{\text{add}}_i(\mathbf{r}_1,\mathbf{b},\mathbf{c})\left(\tilde{W}_{i+1}(\mathbf{b})+\tilde{W}_{i+1}(\mathbf{c})\right)+\widetilde{\text{mult}}_i(\mathbf{r}_1,\mathbf{b},\mathbf{c})\left(\tilde{W}_{i+1}(\mathbf{b})\bullet\tilde{W}_{i+1}(\mathbf{c})\right).$$

- At end of sum-check protocol, V must evaluate

$$\widetilde{\text{add}}_i(\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3)\left(\tilde{W}_{i+1}(\mathbf{r}_2)+\tilde{W}_{i+1}(\mathbf{r}_3)\right)+\widetilde{\text{mult}}_i(\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3)\left(\tilde{W}_{i+1}(\mathbf{r}_2)\bullet\tilde{W}_{i+1}(\mathbf{r}_3)\right)$$

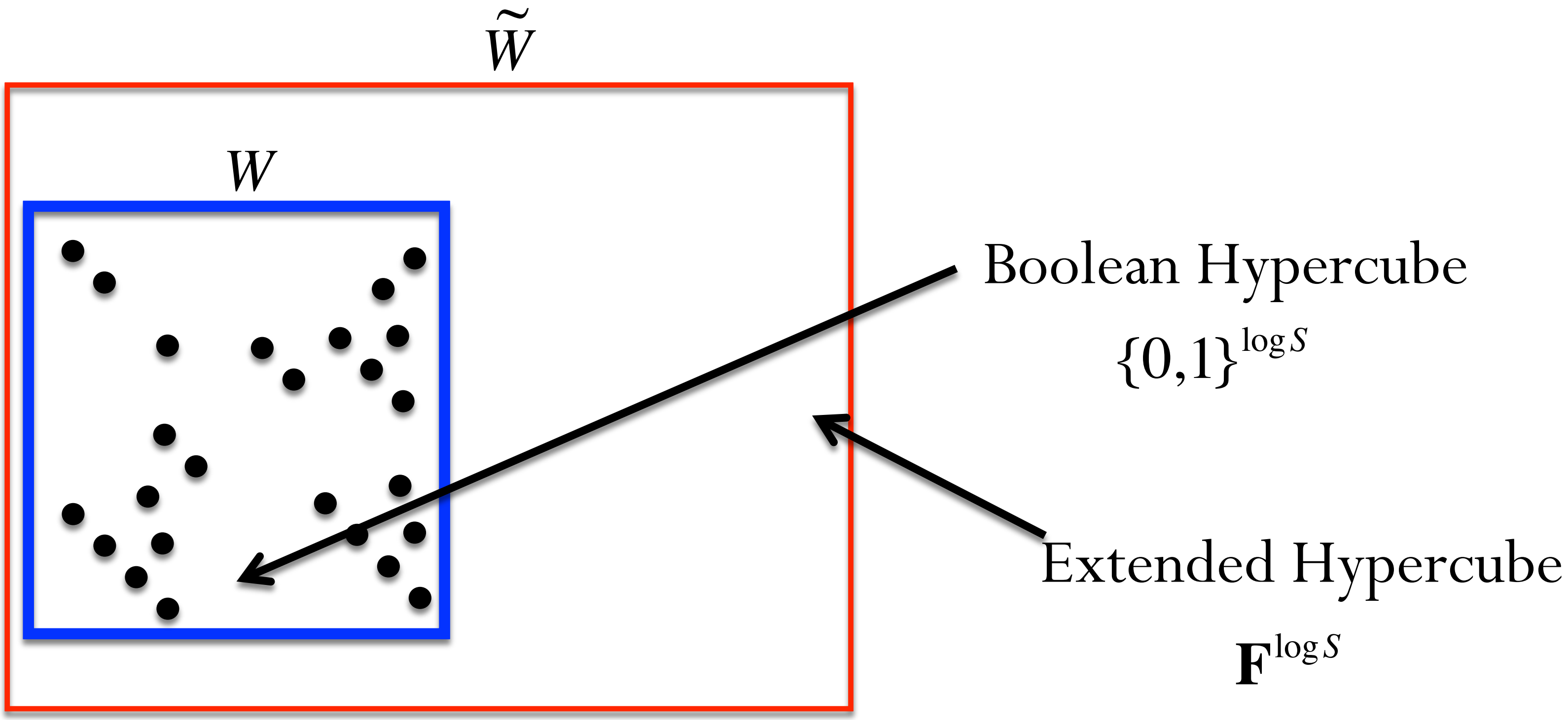
  for randomly chosen $\mathbf{r}_2,\mathbf{r}_3 \in \{0,1\}^{\log S}$.
- Let us assume V can compute $\widetilde{\text{add}}_i(\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3)$ and $\widetilde{\text{mult}}_i(\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3)$ unaided in polylog(n) time.
- Then V only needs to know $\tilde{W}_{i+1}(\mathbf{r}_2)$ and $\tilde{W}_{i+1}(\mathbf{r}_3)$ to complete this check.
- Then iteration i+1 is devoted to computing these values.

Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.

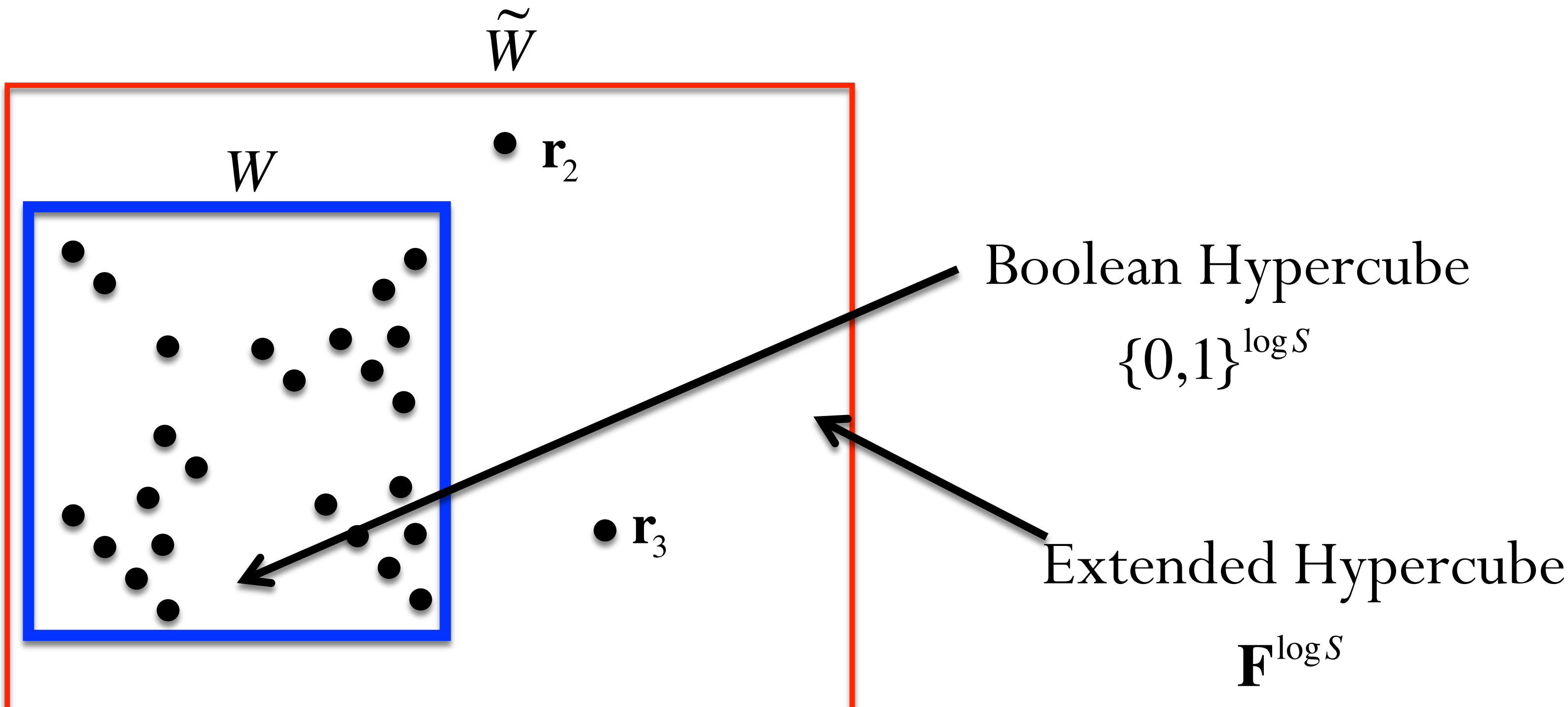
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



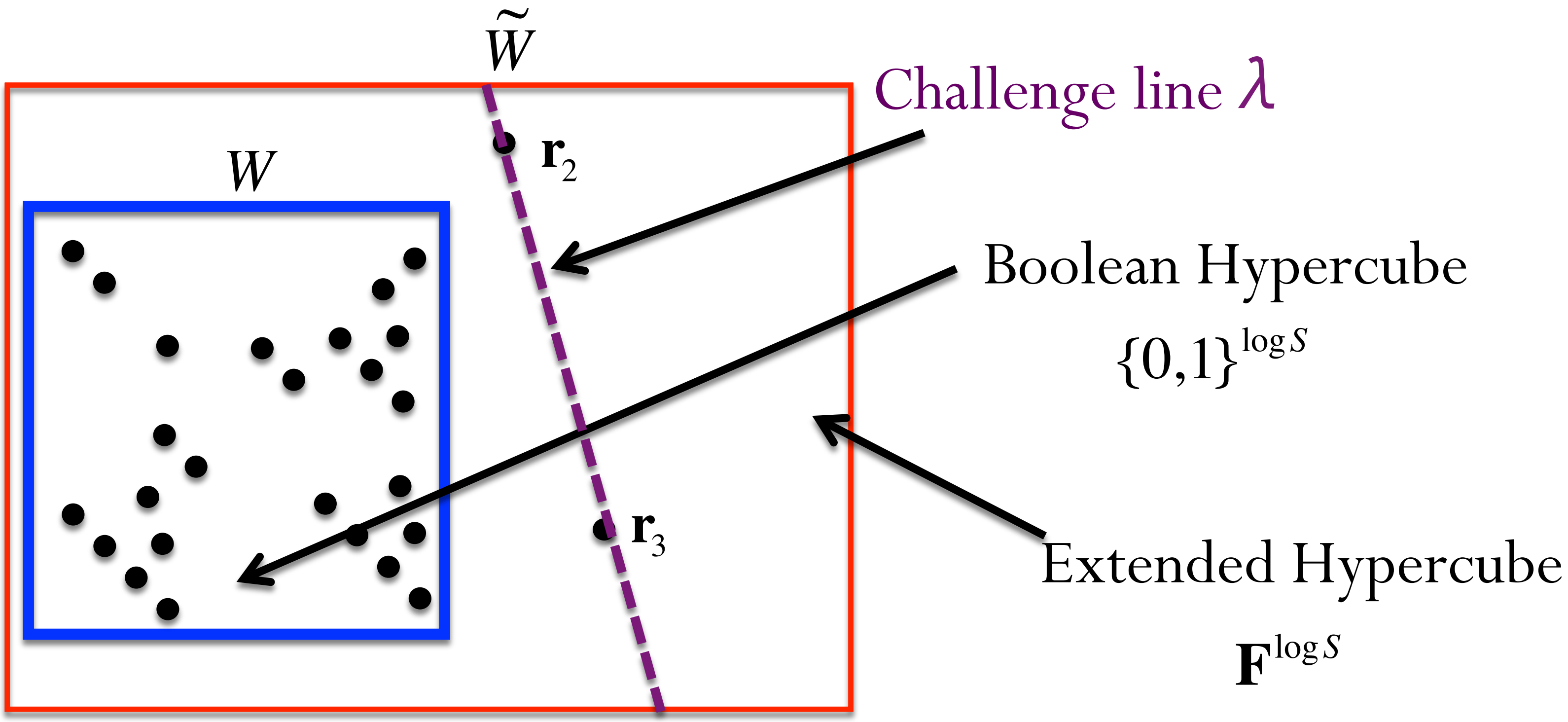
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



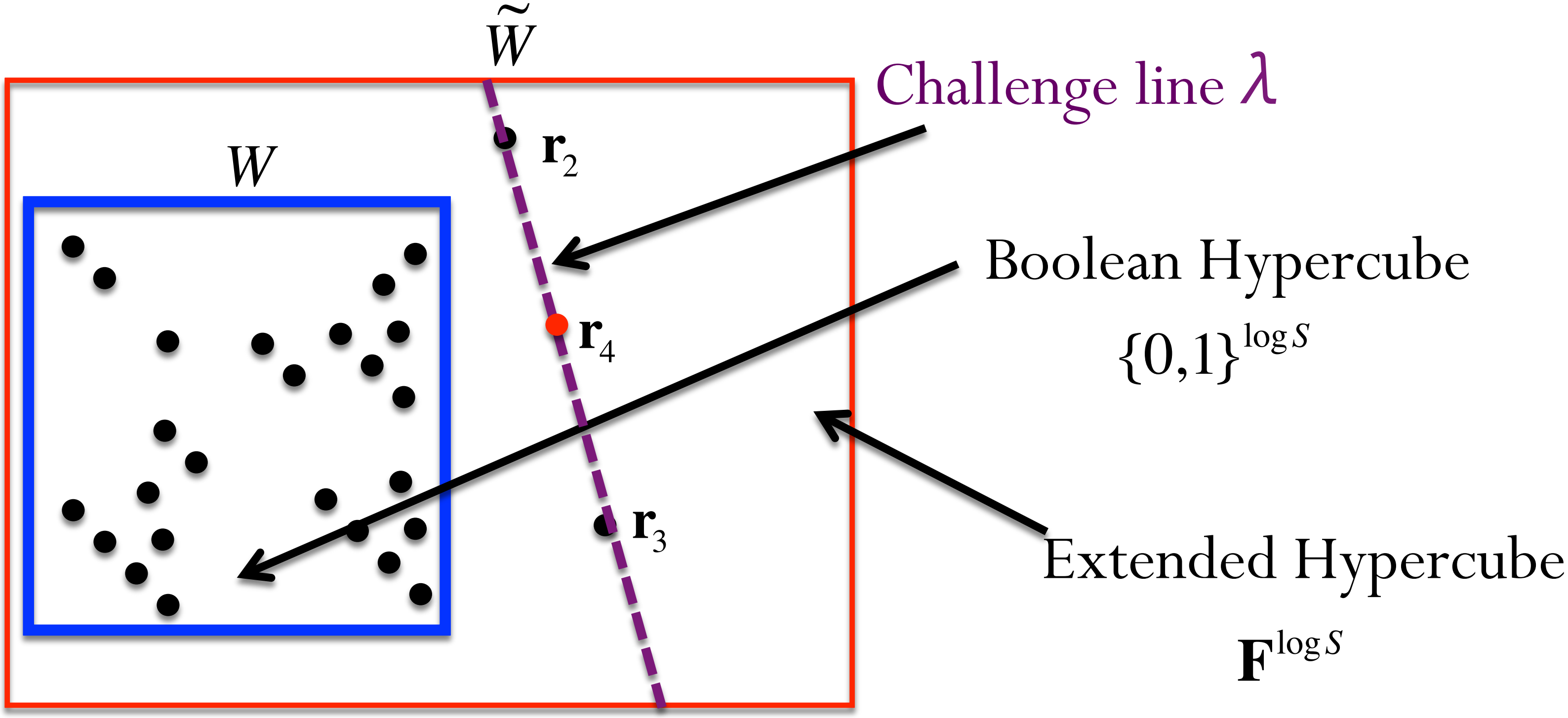
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.



Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don't want to have to separately verify both $\widetilde{W}_{i+1}(\mathbf{r}_2)$ and $\widetilde{W}_{i+1}(\mathbf{r}_3)$ in iteration $i+1$.
- Solution: Reduce verifying both of the above values to verifying $\widetilde{W}_{i+1}(\mathbf{r}_4)$ for a single point $\mathbf{r}_4 \in \mathbf{F}^{\log S}$.

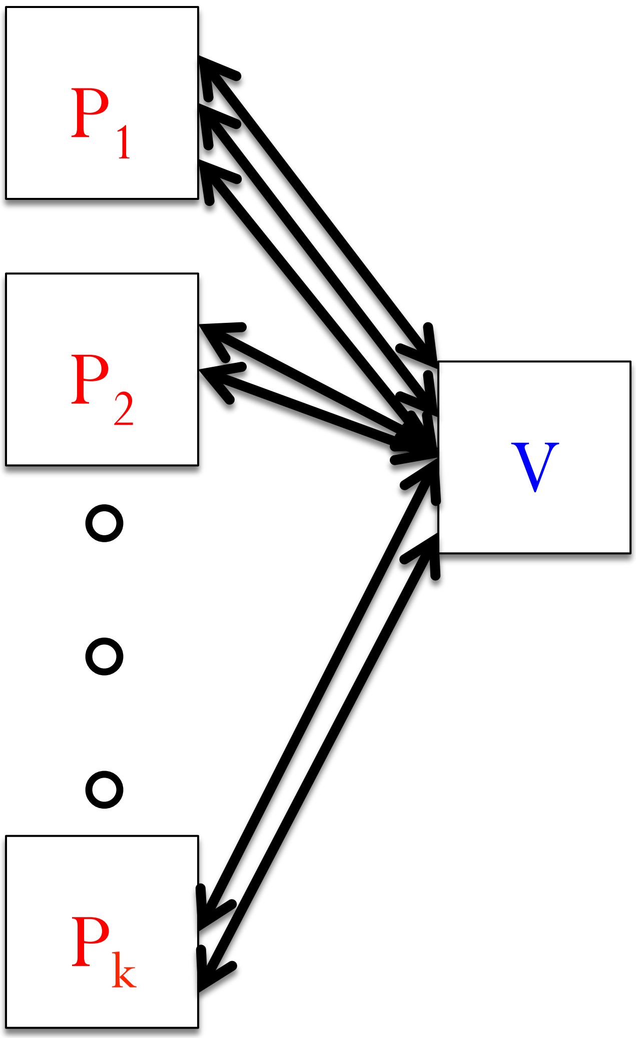


Multi-Prover Interactive Proofs

Lecture Outline

1. Interactive Proofs
   - Motivation, History of Work
   - Techniques:
     - Sum-Check Protocol
     - IP=PSPACE [LFKN, Shamir]
     - MatMult Protocol [T., 2013]
     - GKR Protocol [GKR, 2008]
2. Multi-Prover Interactive Proofs
   - Why can MIPs with polynomial-time verifiers solve harder problems than IPs?
   - Why can MIPs with linear-time verifiers solve "easy" problems more efficiently than IPs?
   - Sketch of a state-of-the-art MIP [BTVW, unpublished]
3. PCPs
   - Reltionship to MIPs
   - A first PCP from an MIP
   - A state-of-the-art PCP [BSS08]
4. Argument Systems
   - From "short" PCPs [Kilian 1992]
   - Without short PCPs [IKO 2007, GGPR 2013]
     - Basis of all implemented argument systems

A k-Prover MIP
[Ben-Or, Goldwasser, Kilian, Wigderson, 1988]



Provers cannot communicate with each other.

What Does a Second Prover Buy?

- First Answer: **Non-Adaptivity**.
- Theorem [FRS 1994]: Let L be a language and M a probabilistic polynomial time Turing Machine such that:
  a) x in L $\Leftrightarrow$ there exists an oracle O such that M^O accepts x with probability 1.
  b) x not in L $\Leftrightarrow$ for all oracles O, M^O rejects x with probability at least 2/3.

  Then there is a 2-prover MIP for L where V runs in polynomial time.

What Does a Second Prover Buy?

- First Answer: **Non-Adaptivity**.
- Theorem [FRS 1990]: Let L be a language and M a probabilistic polynomial time Turing Machine such that:
  a) x in L $\Leftrightarrow$ there exists an oracle O such that M^O accepts x with probability 1.
  b) x not in L $\Leftrightarrow$ for all oracles O, M^O rejects x with probability at least 2/3.

  Then there is a 2-prover MIP for L where V runs in polynomial time.
- Proof: The MIP is
  - V simulates M on input x and every time M poses a query q_i to the oracle, V asks q_i to P_1.
  - Afterward, V picks a random q_i and asks it to P_2.
  - V outputs 0 if P_2's answer to q_i does not match P_1's, or if M would output 0 when treating P_1's answers as the oracle's responses.
  - V repeats the above 3k times, where k is an upper bound on the number of oracle queries M makes. At the end, if V hasn't output 0, it outputs 1.

But What Does Non-Adaptivity Buy?

- Answer: Efficient support for non-determinism.
  - For any language L is in **NP**, the provers will be able to convince V that they hold a witness w that the input is in L, **without** sending w to V.
    - This is the core of the famous result that **MIP=NEXP** [BFL 1991], and ultimately of the PCP theorem.

But What Does Non-Adaptivity Buy?

- Answer: Efficient support for non-determinism.
  - For any language L is in **NP**, the provers will be able to convince V that they hold a witness w that the input is in L, **without** sending w to V.
    - This is the core of the famous result that **MIP=NEXP** [BFL 1991], and ultimately of the PCP theorem.
- But in the real world, no one is solving NEXP-complete problems (or even NP-complete problems) in the worst case.
  - Can MIPs solve "easy" problems more efficiently than IPs?
  - Answer: Yes.
  - Reason: Support for non-determinism enables more efficient transformations from computer programs to problems amenable to probabilistic checking (i.e., circuit evaluation).

Efficient Reductions from RAMs to Non-Deterministic Circuit Evaluation

- Suppose we have a RAM M running in time T.
- We will turn M into a non-deterministic circuit C of size O(T*polylog T) that computes the same function as M. That is:
  - C will take an explicit input x and non-deterministic input w.
  - M accepts x $\Leftrightarrow$ there is a w such that C(x, w)=1.
  - Such efficient transformations from RAMs to **deterministic** circuits are not known.
- And then we can apply to C an efficient MIP for non-deterministic circuit evaluation.

Sketch of the Transformation

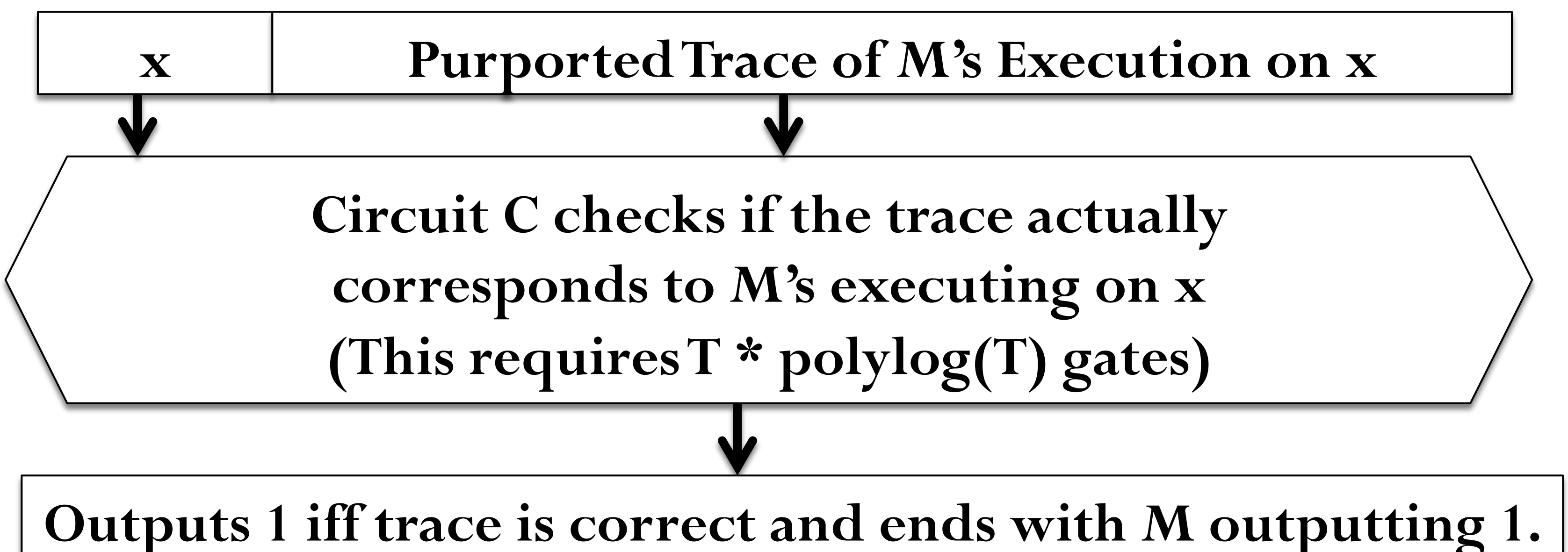
[Gurevich and Shelah 89, Robson91, Ben-Sasson et al. 2013]

- A **trace** of M on input x is the list of the (time, configuration) pairs that arise when running M on x.
  - A configuration specifies the bits in M's program counter and registers.
- C takes x as explicit input, and takes an entire **trace** of M as non-deterministic input.
- C then checks the trace for correctness, and if so outputs whatever M outputs in the trace.

Sketch of the Transformation

[Gurevich and Shelah 89, Robson91, Ben-Sasson et al. 2013]

- A **trace** of M on input x is the list of the (time, configuration) pairs that arise when running M on x.
  - A configuration specifies the bits in M's program counter and registers.
- C takes x as explicit input, and takes an entire **trace** of M as non-deterministic input.
- C then checks the trace for correctness, and if so outputs whatever M outputs in the trace.



Sketch of the Transformation

[Gurevich and Shelah 89, Robson91, Ben-Sasson et al. 2013]

- A **trace** of M on input x is the list of the (time, configuration) pairs that arise when running M on x.
  - A configuration specifies the bits in M's program counter and registers.
- C takes x as explicit input, and takes an entire **trace** of M as non-deterministic input.
- C then checks the trace for correctness, and if so outputs whatever M outputs in the trace.
  - C must check two properties of the trace.
    - **Time consistency** (the claimed state at time t correctly follows from the claimed state at time t-1).
    - **Memory consistency** (whenever M reads a value from a memory location, the value that is returned is the last value that was written).
    - Time-consistency is easy to check: represent M's transition function as a small subcircuit, apply it to each entry t of the trace and check that it equals entry t+1.
    - Checking memory consistency is done by "re-sorting" the trasncript based on memory location, with ties broken by time.

Non-Deterministic Circuit Evaluation

- Given: An arithmetic circuit C over **F** of size S with explicit input x and non-deterministic input w, and claimed output(s) y.
- Goal: Determine if there exists a w such that C(x, w)=y.

Non-Deterministic Circuit Evaluation

- Given: An arithmetic circuit C over **F** of size S with explicit input x and non-deterministic input w, and claimed output(s) y.
- Goal: Determine if there exists a w such that C(x, w)=y.
- Assign each gate in C a (log S)-bit label.
- Call a function $W:\{0,1\}^{\log S} \rightarrow \mathbf{F}$ a *transcript* for C.
  - Say that W is *correct* on x if it satisfies the following properties:
    - The values W assigns to the explicit input gates equal x.
    - The value W assigns to the output gates is y.
    - The values W assigns to the intermediate gates correspond to the correct operation of the gates.
    - Clearly there is a w such that C(x, w)=1 iff there is a correct transcript for C.

Sketch of 2-Prover MIP for Non-Deterministic Circuit Evaluation

[Blumberg, Thaler, Vu, Walfish, unpublished]

- Protocol Sketch:
  - P_1 and P_2 claim to hold an extension Z of a correct transcript W for C.
  - Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \rightarrow \mathbf{F}$ (that depends on x and Z) such that:

    Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.

Sketch of 2-Prover MIP for Non-Deterministic Circuit Evaluation

[Blumberg, Thaler, Vu, Walfish, unpublished]

- Protocol Sketch:
  - P_1 and P_2 claim to hold an extension Z of a correct transcript W for C.
  - Identify a polynomial $g_{x,Z}: \{0,1\}^{3 \log S} \rightarrow \mathbf{F}$ (that depends on x and Z) such that:

    Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3 \log S}$.
  - V checks this by running sum-check protocol with P_1 to compute

$$0 \stackrel{?}{=} \sum_{(a,b,c) \in \{0,1\}^{3 \log S}} g_{x,Z}^2(a,b,c).$$

Sketch of 2-Prover MIP for Non-Deterministic Circuit Evaluation

[Blumberg, Thaler, Vu, Walfish, unpublished]

- Protocol Sketch:
  - P_1 and P_2 claim to hold an extension Z of a correct transcript W for C.
  - Identify a polynomial $g_{x,Z}: \{0,1\}^{3\log S} \to \mathbf{F}$ (that depends on x and Z) such that:
    Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.
  - V checks this by running sum-check protocol with P_1 to compute
    - $$0 \stackrel{?}{=} \sum_{(a,b,c) \in \{0,1\}^{3\log S}} g^2_{x,Z}(a,b,c).$$
  - To perform final check in sum-check protocol, V needs to evaluate $g^2_{x,Z}$ at a random point. But this requires evaluating Z at a random point, and Z only "exists" in P_1's head.
    - So V asks P_2 for the evaluation of Z.
    - Soundness analysis of sum-check is valid as long as P_2's claim about Z is consistent with a **low-degree** polynomial. So V also runs a **low-degree test** with P_1 and P_2.

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \rightarrow \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z}:\{0,1\}^{3\log S}\rightarrow \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow$ $g_{x,Z}(a,b,c)=0\ \forall\ (a,b,c)\in\{0,1\}^{3\log S}$.
- Let add$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})=(\text{in}_1(\mathbf{a}),\ \text{in}_2(\mathbf{a}))$ and gate **a** is an addition gate.
- Let mult$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})=(\text{in}_1(\mathbf{a}),\ \text{in}_2(\mathbf{a}))$ and gate **a** is a mult gate.
- Let io$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff gate **a** is in the explicit input x and $(\mathbf{b},\mathbf{c})=(\mathbf{0},\mathbf{0})$,

  or if **a** is an output gate and **b** and **c** are in-neighbors of **a**.
- Let $I_x(\mathbf{a})$ output $x_{\mathbf{a}}$ if **a** is an input gate, $y_{\mathbf{a}}$ if **a** is an output gate, and 0 otherwise.
- Key Lemma: For $G_{x,W}:\{0,1\}^{3\log S}\rightarrow \mathbf{F}$ defined below, W is a correct transcript on x iff $G_{x,W}(\mathbf{a},\mathbf{b},\mathbf{c})=0$ for all $(\mathbf{a},\mathbf{b},\mathbf{c})$ in $\{0,1\}^{3\log S}$.

$$G_{x,W}(\mathbf{a},\mathbf{b},\mathbf{c}) := \text{io}(\mathbf{a},\mathbf{b},\mathbf{c})\bullet(\text{I}_{\text{x}}(\mathbf{a})\text{-W}(\mathbf{a})) + \text{add}(\mathbf{a},\mathbf{b},\mathbf{c})(\text{W}(\mathbf{a})\text{-}(\text{W}(\mathbf{b})\text{+W}(\mathbf{c})) + \text{mult}(\mathbf{a},\mathbf{b},\mathbf{c})\ \bullet\ (\text{W}(\mathbf{a})\text{-W}(\mathbf{b})\bullet \text{W}(\mathbf{c}))$$

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z}:\{0,1\}^{3\log S}\to\mathbf{F}$ (that depends on x and Z) such that:

  $$Z \text{ extends a correct transcript } \Leftrightarrow g_{x,Z}(a,b,c)=0 \;\forall\; (a,b,c)\in\{0,1\}^{3\log S}.$$

- Let add$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})=(\text{in}_1(\mathbf{a}), \text{in}_2(\mathbf{a}))$ and gate $\mathbf{a}$ is an addition gate.
- Let mult$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff $(\mathbf{b},\mathbf{c})=(\text{in}_1(\mathbf{a}), \text{in}_2(\mathbf{a}))$ and gate $\mathbf{a}$ is a mult gate.
- Let io$(\mathbf{a},\mathbf{b},\mathbf{c})$ output 1 iff gate $\mathbf{a}$ is in the explicit input x and $(\mathbf{b},\mathbf{c})=(\mathbf{0},\mathbf{0})$,

  or if $\mathbf{a}$ is an output gate and $\mathbf{b}$ and $\mathbf{c}$ are in-neighbors of $\mathbf{a}$.
- Let $I_x(\mathbf{a})$ output $x_{\mathbf{a}}$ if $\mathbf{a}$ is an input gate, $y_{\mathbf{a}}$ if $\mathbf{a}$ is an output gate, and 0 otherwise.
- Key Lemma: For $G_{x,W}:\{0,1\}^{3\log S}\to\mathbf{F}$ defined below, W is a correct transcript on x iff $G_{x,W}(\mathbf{a},\mathbf{b},\mathbf{c})=0$ for all $(\mathbf{a},\mathbf{b},\mathbf{c})$ in $\{0,1\}^{3\log S}$.

$$G_{x,W}(\mathbf{a},\mathbf{b},\mathbf{c}) := \text{io}(\mathbf{a},\mathbf{b},\mathbf{c})\bullet(I_x(\mathbf{a})-W(\mathbf{a})) + \text{add}(\mathbf{a},\mathbf{b},\mathbf{c})(W(\mathbf{a})-(W(\mathbf{b})+W(\mathbf{c})) + \text{mult}(\mathbf{a},\mathbf{b},\mathbf{c}) \bullet (W(\mathbf{a})-W(\mathbf{b})\bullet W(\mathbf{c}))$$

Proof: A case analysis depending on whether a is an input gate, non-output gate, output addition gate, or output multiplication gate. Exploits non-trivial cancellation for output gates.

Definition of the Key Polynomial

- Identify a polynomial $g_{x,Z} : \{0,1\}^{3\log S} \to \mathbf{F}$ (that depends on x and Z) such that:

  Z extends a correct transcript $\Leftrightarrow$ $g_{x,Z}(a,b,c) = 0 \ \forall \ (a,b,c) \in \{0,1\}^{3\log S}$.
- Let add($\mathbf{a,b,c}$) output 1 iff ($\mathbf{b,c}$)=(in$_1$($\mathbf{a}$), in$_2$($\mathbf{a}$)) and gate $\mathbf{a}$ is an addition gate.
- Let mult($\mathbf{a,b,c}$) output 1 iff ($\mathbf{b,c}$)=(in$_1$($\mathbf{a}$), in$_2$($\mathbf{a}$)) and gate $\mathbf{a}$ is a mult gate.
- Let io($\mathbf{a,b,c}$) output 1 iff gate $\mathbf{a}$ is in the explicit input x and ($\mathbf{b,c}$)=($\mathbf{0,0}$),

  or if $\mathbf{a}$ is an output gate and $\mathbf{b}$ and $\mathbf{c}$ are in-neighbors of $\mathbf{a}$.
- Let $I_x(\mathbf{a})$ output $x_{\mathbf{a}}$ if $\mathbf{a}$ is an input gate, $y_{\mathbf{a}}$ if $\mathbf{a}$ is an output gate, and 0 otherwise.
- Key Lemma: For $G_{x,W} : \{0,1\}^{3\log S} \to \mathbf{F}$ defined below, W is a correct transcript on x iff $G_{x,W}(\mathbf{a,b,c}) = 0$ for all $(\mathbf{a,b,c})$ in $\{0,1\}^{3\log S}$.

$$G_{x,W}(\mathbf{a,b,c}) := \mathrm{io}(\mathbf{a,b,c}) \bullet (\mathrm{I_x}(\mathbf{a})\text{-}\mathrm{W}(\mathbf{a})) + \mathrm{add}(\mathbf{a,b,c})(\mathrm{W}(\mathbf{a})\text{-}(\mathrm{W}(\mathbf{b})+\mathrm{W}(\mathbf{c})) + \mathrm{mult}(\mathbf{a,b,c}) \bullet (\mathrm{W}(\mathbf{a})\text{-}\mathrm{W}(\mathbf{b}) \bullet \mathrm{W}(\mathbf{c}))$$

- So we define:

$$g_{x,Z}(\mathbf{a,b,c}) = \widetilde{\mathrm{io}}(\mathbf{a,b,c}) \bullet (\widetilde{\mathrm{I}}_{\mathrm{x}}(\mathbf{a})\text{-}\mathrm{Z}(\mathbf{a})) + \widetilde{\mathrm{add}}(\mathbf{a,b,c})(\mathrm{Z}(\mathbf{a})\text{-}(\mathrm{Z}(\mathbf{b})+\mathrm{Z}(\mathbf{c})) + \widetilde{\mathrm{mult}}(\mathbf{a,b,c}) \bullet (\mathrm{Z}(\mathbf{a})\text{-}\mathrm{Z}(\mathbf{b}) \bullet \mathrm{Z}(\mathbf{c}))$$

Costs of the 2-Prover MIP for Non-Deterministic Circuit Evaluation

Rounds	V Time	P_1 Time	P_2 Time
log S	$O(n + \log^2 S)$	O(S log S)	O(S log S)

Combining this MIP with the RAM $\Longrightarrow$ non-deterministic circuit reduction sketched before, we get an MIP that can simulate any RAM that runs in time T. In the MIP, V runs in time O(n + polylog(T)) and P_1 and P_2 run in time O(T*polylog(T)).

PCPs

The PCP Model For A Language L

- V is given oracle access to a static proof string π in Σ^{ℓ}.
  - Standard notions of completeness and soundness must hold.
    - If x is in L, then there must exist a proof string causing V to accept.
    - If x is not in L, there for all proof strings, V must reject w.h.p.
  - ℓ is called the **length** or **size** of the proof.
  - Σ is called the **alphabet**.
  - **Prover time** refers to the time required to generate π.
  - If V only looks at q entries of the proof string, then q is referred to as the **query cost**.

Relationship Between MIPs and PCPs

- Every MIP can be turned into a PCP and vice versa.
  - But the transformations can blow up costs (e.g., P time, V time, communication, query costs, etc.).

MIP $\Longrightarrow$ PCP Transformation

- Lemma: Suppose L has a k-prover MIP in which V sends one message to each prover, with each message consisting of at most r_Q bits, and each prover sends at most r_A bits in response. Then L has a k-query PCP over alphabet $\Sigma = [2^{r_A}]$ with proof size $k2^{r_Q}$. V's runtime, soundness error and completeness error are the same as in the MIP.
- Proof: For each prover P_i in the MIP, the PCP has an entry for every possible message to P_i. The PCP verifier simulates the MIP verifier, treating the proof string as the provers' answers in the MIP.

MIP $\Rightarrow$ PCP Transformation

- Lemma: Suppose L has a k-prover MIP in which V sends one message to each prover, with each message consisting of at most r_Q bits, and each prover sends at most r_A bits in response. Then L has a k-query PCP over alphabet $\Sigma = [2^{r_A}]$ with proof size $k2^{r_Q}$. V's runtime, soundness error and completeness error are the same as in the MIP.
- Proof: For each prover P_i in the MIP, the PCP has an entry for every possible message to P_i. The PCP verifier simulates the MIP verifier, treating the proof string as the provers' answers in the MIP.
- Highlights a key difference between MIPs and PCPs.
  - MIP provers only need to compute answers "on demand".
  - A PCP prover must "write down" an answer to every possible question V might ask.

PCP$\Longrightarrow$ MIP Transformation

- Lemma: Suppose L has a PCP system in which V makes k queries to a proof of length ℓ over an alphabet Σ with soundness error δ_s. Then L has a $(k+1)$-prover MIP in which P and V's runtimes are preserved, and the soundness error of the MIP is at most $\max\{1-1/k,\ \delta_s\}$.

- Proof: For each PCP query q_i that the PCP verifier makes, the MIP verifier poses q_i to a different prover P_i, then picks $i \in [k]$ at random and poses q_i to the remaining prover to make sure its answer matches that of P_i.

PCP $\Longrightarrow$ MIP Transformation

- Lemma: Suppose L has a PCP system in which V makes k queries to a proof of length ℓ over an alphabet Σ with soundness error δ_s. Then L has a $(k+1)$-prover MIP in which P and V's runtimes are preserved, and the soundness error of the MIP is at most $\max\{1-1/k,\ \delta_s\}$.

- Proof: For each PCP query q_i that the PCP verifier makes, the MIP verifier poses q_i to a different prover P_i, then picks $i \in [k]$ at random and poses q_i to the remaining prover to make sure its answer matches that of P_i.
- Highlights the two more key differences between MIPs and PCPs.
  - In an MIP, each prover can act **adaptively** if asked more than one question.
  - Even if the provers in an MIP don't act adaptively, they may not answer with respect to the same function π.

A First PCP, From an MIP

- In the MIP from earlier, it was sound to work over a field of size polylog(S), and V set O(log S) field elements to each prover, where S was the size of (non-deterministic) circuit we were simulating.
  - So total number of bits sent by V was r_Q=O(log(S) * loglog(S))
    $\Rightarrow$ PCP of length $O(2^{r_Q}) = S^{O(\text{loglog } S)}$.
- By tweaking parameters in the MIP itself, we can reduce r_Q to O(log S).

A First PCP, From an MIP

- In the MIP from earlier, it was sound to work over a field of size polylog(S), and V set O(log S) field elements to each prover, where S was the size of (non-deterministic) circuit we were simulating.
  - So total number of bits sent by V was r_Q=O(log(S) * loglog(S))

    $\Rightarrow$ PCP of length $O(2^{r_Q}) = S^{O(\text{loglog } S)}$.
- By tweaking parameters in the MIP itself, we can reduce r_Q to O(log S).
  - Don't assign gates binary labels and use multilinear extensions over log S variables.
  - Instead, assign them labels in base b for b=(log S)/(loglog S), so each label consists of b digits, since b^b=S.
  - Can use extensions of degree b in each variable, so it is still sound to work over a field **F** of size polylog(S).
  - So r_Q becomes O(b *log|**F**|)=O((log S)/(loglog S)*loglog S) = O(log S)

    $\Rightarrow$ PCP of size poly(S).

A State of the Art PCP

- [BSS 2005]: A PCP for simulating a RAM M running in time T, with proof length O(T*polylog(T)) and O(polylog(T)) queries by V.
- [BSGHSV 2005]: Reduced V's **time** in the PCP to O(n*polylog(T))
- [BSCGT 2013]: Improved constants, and showed how to generate the proof in time O(T*polylog(T)) using FFTs.
  - Still complicated, large hidden constants, must work over fields of characteristic 2.

Argument Systems

Argument Systems

- Argument systems are constructed in a 2-step process:
  1. Construct an information-theoretically secure protocol for a model in which cheating provers behave in a restricted model.
  2. Use crypto to force a single prover to behave in this model.

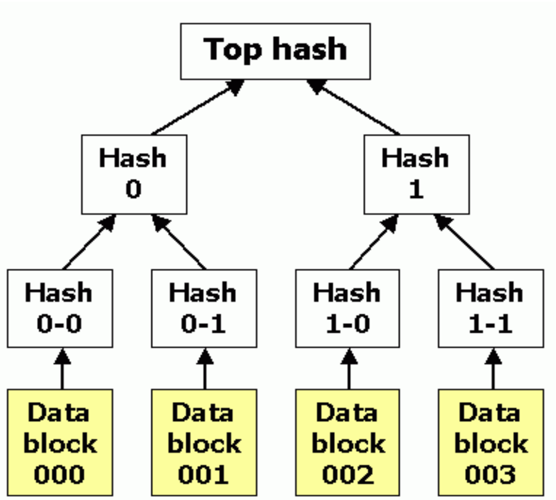
Argument Systems and Their Properties

Information-Theoretically Secure Model	Crypto Primitive	Argument System Properties	Reference
Polynomial size PCP	CRHF	4-message argument for NP Zero-Knowledge (ZK) Proof of Knowledge (PoK)	Kilian 1992
"linear" PCP of exponential size	Additively homomorphic encryption	Same as above, but **with pre-processing** (also, simpler w/better constants)	IKO 2007, GGPR 2013
------------------------------	"linear only" additively homomorphic encryption	2-message argument for NP **with pre-processing** ZK+PoK+public verifiability	GGPR 2013, BCIOP 2013
MIP	Fully-Homomorphic Encryption (FHE)	4-message "complexity-preserving" argument for NP with PoK	Bitansky-Chiesa 2012
No-signaling MIP	FHE or PRI	2-message argument for P Publicly verifiable [PR15]	KRR 2014

Kilian’s Argument System

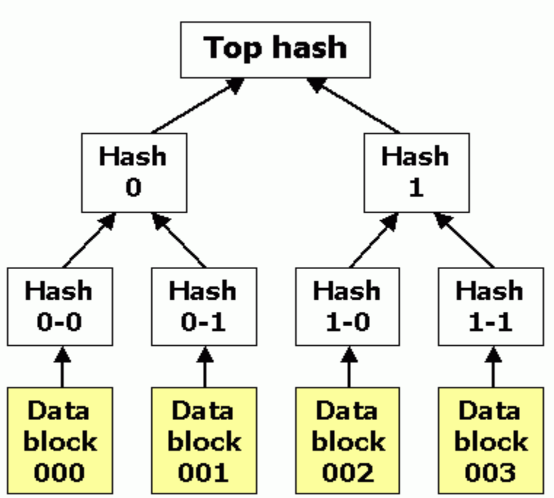
Merkle Trees

- A Merkle Hash Tree gives a way to outsource storage of a bunch of data to an untrusted "prover" P.
- Fix a collision resistant hash function $h:\{0,1\}^k \times \{0,1\}^k \rightarrow \{0,1\}^k$. The prover uses h to build a hash tree over the data.
- Suppose V knows the root hash.



Merkle Trees

- A Merkle Hash Tree gives a way to outsource storage of a bunch of data to an untrusted "prover" P.
- Fix a collision resistant hash function $h:\{0,1\}^k \times \{0,1\}^k \rightarrow \{0,1\}^k$. The prover uses h to build a hash tree over the data.
- Suppose V knows the root hash.
- If V wants to know a data block, she asks P to provide the data block, and all nodes on its path to the root, along with their siblings.
  - Called the **witness path** for the data block.
- V checks that all provided nodes actually equal the hash of the children, and that the claimed root hash is correct.
- For P to lie about the value of the data block, there must be a hash-collision somewhere on the real root-to-leaf path and the claimed root-to-leaf path.



Kilian's Argument System

- Combine a PCP with a Merkle tree.
- In more detail:

1. Commit Phase of the Argument System:

2. Reveal Phase of the Argument System:

Kilian's Argument System

- Combine a PCP with a Merkle tree.
- In more detail:

1. Commit Phase of the Argument System:
   - V sends a collision-resistant hash function h to P.
   - Let π be a PCP attesting to $x \in L$. P builds a Merkle tree over π using h and sends the root hash to V.
2. Reveal Phase of the Argument System:
   - Let $q_1, \ldots, q_k$ be the PCP verifier's queries to π. V sends these queries to P.
   - P sends back $\pi(q_1), \ldots, \pi(q_k)$ plus the witness path for each.

Kilian's Argument System

- Combine a PCP with a Merkle tree.
- In more detail:
  1. Commit Phase of the Argument System:
     - V sends a collision-resistant hash function h to P.
     - Let π be a PCP attesting to $x \in L$. P builds a Merkle tree over π using h and sends the root hash to V.
  2. Reveal Phase of the Argument System:
     - Let $q_1, \ldots, q_k$ be the PCP verifier's queries to π. V sends these queries to P.
     - P sends back $\pi(q_1), \ldots, \pi(q_k)$ plus the witness path for each.
- Soundness proof sketch: By security of the Merkle Tree, after the reveal phase P is "committed" to answer all k queries in the Reveal Phase using a single, fixed function π. Hence, by soundness of the PCP, if P can convince V to accept with non-negligible probability, then $x \in L$.

Costs of Kilian's Argument System When Instantiated with State-Of-The-Art PCP for Non-Deterministic Circuit Evaluation

Messages	Communication	V Time	P Time
4	polylog(S)	O(n + polylog S)	O(S*polylog S)

Downsides:

*State-of-the-art PCPs are complicated, concretely expensive.

*The argument system is interactive, not publicly verifiable (though it can be made ZK and PoK).

*State-of-the-art PCPs require a lot of space for the prover (who must perform FFTs over entire computation traces).

Argument Systems from Linear PCPs [Ishai, Kushilevitz, Ostrovsky, 2008]

Interactive Arguments from Linear PCPs

- The reason Kilian needs a polynomial-size PCP is that the prover must materialize the full proof π to commit to it.
- Can avoid this if π is structured (i.e., linear).
  - i.e., $\pi : \mathbf{F}^v \to \mathbf{F}$ and $\pi(q_1 + q_2) = \pi(q_1) + \pi(q_2)$.
- Step 1: Give commit/reveal protocol for linear functions $\pi : \mathbf{F}^v \to \mathbf{F}$.
  - Will use a semantically secure **additively homomorphic** encryption scheme.
  - i.e. Enc($q_1 + q_2$)=Enc(q_1) + Enc(q_2).
- Step 2: Give a linear PCP for non-deterministic circuit evaluation.
  - First, we give one of length $|\mathbf{F}|^{O(S^2)}$ due to [IKO, 2007].
  - Then, we give one of length $|\mathbf{F}|^{O(S)}$ due to [GGPR, 2013].

Interactive Arguments from Linear PCPs

- The reason Kilian needs a polynomial-size PCP is that the prover must materialize the full proof π to commit to it.
- Can avoid this if π is structured (i.e., linear).
  - i.e., $\pi : \mathbf{F}^v \rightarrow \mathbf{F}$ and $\pi(q_1 + q_2) = \pi(q_1) + \pi(q_2)$.
- Step 1: Give commit/reveal protocol for linear functions $\pi : \mathbf{F}^v \rightarrow \mathbf{F}$.
  - Will use a semantically secure **additively homomorphic** encryption scheme.
  - i.e. Enc($q_1 + q_2$)=Enc(q_1) + Enc(q_2).
- Step 2: Give a linear PCP for non-deterministic circuit evaluation.
  - First, we give one of length $|\mathbf{F}|^{O(S^2)}$ due to [IKO, 2007].
  - Then, we give one of length $|\mathbf{F}|^{O(S)}$ due to [GGPR, 2013].

Costs of Resulting Argument System

Interactive Arguments from Linear PCPs

- The reason Kilian needs a polynomial-size PCP is that the prover must materialize the full proof π to commit to it.
- Can avoid this if π is structured (i.e., linear).
  - i.e., $\pi : \mathbf{F}^v \to \mathbf{F}$ and $\pi(q_1 + q_2) = \pi(q_1) + \pi(q_2)$.
- Step 1: Give commit/reveal protocol for linear functions $\pi : \mathbf{F}^v \to \mathbf{F}$.
  - Will use a semantically secure **additively homomorphic** encryption scheme.
  - i.e. Enc($q_1 + q_2$)=Enc(q_1) + Enc(q_2).
- Step 2: Give a linear PCP for non-deterministic circuit evaluation.
  - First, we give one of length $|\mathbf{F}|^{O(S^2)}$ due to [IKO, 2007].
  - Then, we give one of length $|\mathbf{F}|^{O(S)}$ due to [GGPR, 2013].

Costs of Resulting Argument System

Messages	Communication	V Time	P Time
4	V → P communication: O(S) field elements P → V communication: O(1) field elements	O(S), but amortizable over a **batch** of inputs to C	O(S *log² S)

Step 1: Commit/Reveal For Linear Functions $\pi : \mathbf{F}^v \rightarrow \mathbf{F}$

- Guarantee: At end of commit phase, there is some function π' (not necessarily linear) such that if P passes V's test with non-negligible probability, then answers in the reveal phase are consistent with π'.
- Commit phase:
- Reveal phase:

Step 1: Commit/Reveal For Linear Functions $\pi : \mathbf{F}^v \to \mathbf{F}$

- Guarantee: At end of commit phase, there is some function π' (not necessarily linear) such that if P passes V's test with non-negligible probability, then answers in the reveal phase are consistent with π'.
- Commit phase:
  - V chooses a random $r \in \mathbf{F}^v$, sends Enc(r_1),…, Enc(r_v) to P.
  - P sends e=Enc($\pi(r)$) to V using homomorphism of Enc.
  - V lets s=Dec(e).
- Reveal phase:
  - Given queries $q_1,\ldots,q_k$ to π, V picks $\alpha_1,\ldots,\alpha_k$ at random, sends $q_1,\ldots,q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
  - P sends claimed answers $a_1,\ldots,a_k, a^*$ to the queries.
  - V checks if $a^* = s + \sum_i \alpha_i a_i$.

Commit phase:

- V chooses a random $r \in \mathbf{F}^v$, sends $\text{Enc}(r_1), \ldots, \text{Enc}(r_v)$ to P.
- P sends e=Enc($\pi(r)$) to V using homomorphism of Enc.
- V lets s=Dec(e).

Reveal phase:

- Given queries $q_1, \ldots, q_k$ to π, V picks $\alpha_1, \ldots, \alpha_k$ at random, sends $q_1, \ldots, q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
- P sends claimed answers $a_1, \ldots, a_k, a^*$ to the queries.
- V checks if $a^* = s + \sum_i \alpha_i a_i$.

Commit phase:

- V chooses a random $r \in \mathbf{F}^v$, sends Enc(r_1),. . . , Enc(r_v) to P.
- P sends e=Enc($\pi(r)$) to V using homomorphism of Enc.
- V lets s=Dec(e).

Reveal phase:

- Given queries $q_1,\ldots,q_k$ to π, V picks $\alpha_1,\ldots,\alpha_k$ at random, sends $q_1,\ldots,q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
- P sends claimed answers $a_1,\ldots,a_k, a^*$ to the queries.
- V checks if $a^* = s + \sum_i \alpha_i a_i$.

- Proof of Binding:
  - Assume the number of queries V asks in reveal phase is k=1 for simplicity.
  - Consider two runs of the reveal phase, where:
    - In Run 1, V sends q_1 and $q^* := r + \alpha q_1$ and P responds with a_1 and a^*.
    - In Run 2, V sends q_1 and $q^{*\prime} := r + \alpha' q_1$ and P responds with $a_1' \neq a_1$ and $a^{*\prime}$.
    - And V accepts both runs.
  - Claim: In this case, P can solve for (α, α'). Hence, P can solve for r, breaking semantic security of the encryption scheme.

Commit phase:

- V chooses a random $r \in \mathbf{F}^v$, sends $\mathrm{Enc}(r_1), \dots, \mathrm{Enc}(r_v)$ to P.
- P sends e=Enc($\pi(r)$) to V using homomorphism of Enc.
- V lets s=Dec(e).

Reveal phase:

- Given queries $q_1, \dots, q_k$ to π, V picks $\alpha_1, \dots, \alpha_k$ at random, sends $q_1, \dots, q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
- P sends claimed answers $a_1, \dots, a_k, a^*$ to the queries.
- V checks if $a^* = s + \sum_i \alpha_i a_i$.

- Proof of Binding:
  - Assume the number of queries V asks in reveal phase is k=1 for simplicity.
  - Consider two runs of the reveal phase, where:
    - In Run 1, V sends q_1 and $q^* := r + \alpha q_1$ and P responds with a_1 and a^*.
    - In Run 2, V sends q_1 and $q^{*\prime} := r + \alpha' q_1$ and P responds with $a_1' \neq a_1$ and $a^{*\prime}$.
    - And V accepts both runs.
  - Claim: In this case, P can solve for (α, α'). Hence, P can solve for r, breaking semantic security of the encryption scheme.
  - Proof: Even though P doesn't know s, P knows by V's acceptance that

$$\begin{aligned} a^* &= s + \alpha a_1 \\ a^{*\prime} &= s + \alpha' a_1' \end{aligned} \implies \boxed{(a^* - a^{*\prime}) = \alpha a_1 - \alpha' a_1'}$$

Commit phase:

- V chooses a random $r \in \mathbf{F}^v$, sends Enc(r_1),..., Enc(r_v) to P.
- P sends e=Enc($\pi(r)$) to V using homomorphism of Enc.
- V lets s=Dec(e).

Reveal phase:

- Given queries $q_1,\ldots,q_k$ to π, V picks $\alpha_1,\ldots,\alpha_k$ at random, sends $q_1,\ldots,q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
- P sends claimed answers $a_1,\ldots,a_k, a^*$ to the queries.
- V checks if $a^* = s + \sum_i \alpha_i a_i$.

- Proof of Binding:
  - Assume the number of queries V asks in reveal phase is k=1 for simplicity.
  - Consider two runs of the reveal phase, where:
    - In Run 1, V sends q_1 and $q^* := r + \alpha q_1$ and P responds with a_1 and a^*.
    - In Run 2, V sends q_1 and $q^{*\prime} := r + \alpha' q_1$ and P responds with $a_1' \neq a_1$ and $a^{*\prime}$.
    - And V accepts both runs.
  - Claim: In this case, P can solve for (α, α'). Hence, P can solve for r, breaking semantic security of the encryption scheme.
  - Proof: Even though P doesn't know s, P knows by V's acceptance that

$$a^* = s + \alpha a_1$$
$$a^{*\prime} = s + \alpha' a_1' \implies \boxed{(a^* - a^{*\prime}) = \alpha a_1 - \alpha' a_1'}$$

Also, P doesn't know r, but he knows:

$$q^* = r + \alpha q_1$$
$$q^{*\prime} = r + \alpha' q_1 \implies \boxed{(q^* - q^{*\prime}) = \alpha q_1 - \alpha' q_1 \quad \text{(Equality of Vectors)}}$$

Commit phase:

- V chooses a random $r \in \mathbf{F}^v$, sends $\mathrm{Enc}(r_1), \ldots, \mathrm{Enc}(r_v)$ to P.
- P sends e=Enc($\pi(r)$) to V using homomorphism of Enc.
- V lets s=Dec(e).

Reveal phase:

- Given queries $q_1, \ldots, q_k$ to π, V picks $\alpha_1, \ldots, \alpha_k$ at random, sends $q_1, \ldots, q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
- P sends claimed answers $a_1, \ldots, a_k, a^*$ to the queries.
- V checks if $a^* = s + \sum_i \alpha_i a_i$.

- Proof of Binding:
  - Assume the number of queries V asks in reveal phase is k=1 for simplicity.
  - Consider two runs of the reveal phase, where:
    - In Run 1, V sends q_1 and $q^* := r + \alpha q_1$ and P responds with a_1 and a^*.
    - In Run 2, V sends q_1 and $q^{*\prime} := r + \alpha' q_1$ and P responds with $a_1' \neq a_1$ and $a^{*\prime}$.
    - And V accepts both runs.
  - Claim: In this case, P can solve for (α, α'). Hence, P can solve for r, breaking semantic security of the encryption scheme.
  - Proof: Even though P doesn't know s, P knows by V's acceptance that

$$a^* = s + \alpha a_1, \quad a^{*\prime} = s + \alpha' a_1' \implies \boxed{(a^* - a^{*\prime}) = \alpha a_1 - \alpha' a_1'}$$

Also, P doesn't know r, but he knows:

$$q^* = r + \alpha q_1, \quad q^{*\prime} = r + \alpha' q_1 \implies (q^* - q^{*\prime}) = \alpha q_1 - \alpha' q_1$$

(Equality of Vectors)

Pick any j s.t. $q_{1,j} \neq 0$. Then:

$$\boxed{(q_j^* - q_j^{*\prime}) = (\alpha - \alpha') q_1}$$

Commit phase:

- V chooses a random $r \in \mathbf{F}^v$, sends $\text{Enc}(r_1), \ldots, \text{Enc}(r_v)$ to P.
- P sends $e=\text{Enc}(\pi(r))$ to V using homomorphism of Enc.
- V lets $s=\text{Dec}(e)$.

Reveal phase:

- Given queries $q_1, \ldots, q_k$ to π, V picks $\alpha_1, \ldots, \alpha_k$ at random, sends $q_1, \ldots, q_k$ and $q^* := r + \sum_i \alpha_i q_i$ to P.
- P sends claimed answers $a_1, \ldots, a_k, a^*$ to the queries.
- V checks if $a^* = s + \sum_i \alpha_i a_i$.

- Proof of Binding:
  - Assume the number of queries V asks in reveal phase is k=1 for simplicity.
  - Consider two runs of the reveal phase, where:
    - In Run 1, V sends q_1 and $q^* := r + \alpha q_1$ and P responds with a_1 and a^*.
    - In Run 2, V sends q_1 and $q^{*\prime} := r + \alpha' q_1$ and P responds with $a_1' \neq a_1$ and $a^{*\prime}$.
    - And V accepts both runs.
  - Claim: In this case, P can solve for (α, α'). Hence, P can solve for r, breaking semantic security of the encryption scheme.
  - Proof: Even though P doesn't know s, P knows by V's acceptance that

$$a^* = s + \alpha a_1 \quad a^{*\prime} = s + \alpha' a_1' \implies (a^* - a^{*\prime}) = \alpha a_1 - \alpha' a_1'$$

Also, P doesn't know r, but he knows:

$$q^* = r + \alpha q_1 \quad q^{*\prime} = r + \alpha' q_1 \implies (q^* - q^{*\prime}) = \alpha q_1 - \alpha' q_1$$

(Equality of Vectors)

Pick any j s.t. $q_{1,j} \neq 0$. Then:

$$(q_j^* - q_j^{*\prime}) = (\alpha - \alpha') q_1$$

Since $a_1 \neq a_1'$, P has two linearly independent equations in two unknowns, so P can solve for (α, α')

Step 2: A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{O(S^2)}$

- Fix a circuit C taking explicit input x and non-deterministic input w, with claimed outputs y.
- Call a vector $W \in \mathbf{F}^S$ a transcript for C.
  - Say W is a **correct** transcript for input x if:
    - $W_a - x_a = 0$ for all input gates a.
    - $W_a - y_a = 0$ for all output gates a.
    - $W_a - (W_{\text{in}_1(a)} + W_{\text{in}_2(a)}) = 0$ for all addition gates a.
    - $W_a - (W_{\text{in}_1(a)} \bullet W_{\text{in}_2(a)}) = 0$ for all multiplication gates a.
  - Note: S + 1 – w constraints in total. 2 for the output gate, 0 for non-deterministic witness gates, and 1 for all others.
  - Note: All constraints are of the form $Q_i(W) = 0$ for a polynomial Q_i of degree at most 2 in the entries of W.

Step 2: A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{O(S^2)}$

- Let $W \otimes W \in \mathbf{F}^{S^2}$ be the vector whose (i,j)'th entry equals $W_i \bullet W_j$.
- Define $(W, W \otimes W) \in \mathbf{F}^{S+S^2}$ as the concatenation of W and $W \otimes W$.
- For any vector $d \in \mathbf{F}^v$, define $\pi_d : \mathbf{F}^v \to \mathbf{F}$ via $\pi_d(x) = \langle x, d \rangle$.
  - The set of all $|\mathbf{F}|^v$ evaluations of π_d is called the **Hadamard encoding** of d.

Step 2: A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{O(S^2)}$

- Let $W \otimes W \in \mathbf{F}^{S^2}$ be the vector whose (i,j)'th entry equals $W_i \bullet W_j$.
- Define $(W, W \otimes W) \in \mathbf{F}^{S+S^2}$ as the concatenation of W and $W \otimes W$.
- For any vector $d \in \mathbf{F}^v$, define $\pi_d : \mathbf{F}^v \to \mathbf{F}$ via $\pi_d(x) = \langle x, d \rangle$.
  - The set of all $|\mathbf{F}|^v$ evaluations of π_d is called the **Hadamard encoding** of d.
- Honest proof π contains all evaluations of the function $\pi_{(W, W \otimes W)}$.
- V must check:
  1. π is linear.
  2. Assuming 1. holds, that π is of the form $\pi_{(W, W \otimes W)}$ for some W.
  3. Assuming 1. and 2. hold, that W also satisfies all constraints required for W to be a valid transcript.

Checking 1: Linearity Testing

- V picks $q_1, q_2 \in \mathbf{F}^{S+S^2}$ at random, and checks that $\pi(q_1)+\pi(q_2)=\pi(q_1+q_2)$.
  - [Blum, Luby, Rubinfeld, 1993]: Over a field of characteristic 2, if this test passes with probability δ then there exists a linear function π' such that $\pi'(x)=\pi(x)$ for a δ-fraction of inputs x.
  - Over other fields, weaker guarantees are known.
- From now on, let us assume for simplicity that if π passes the linearity test, then π is actually linear.

Checking 2: Assuming π is Linear, Check That it is of the Form $\pi_{(W, W\otimes W)}$ for some W.

- Since π is linear, $\pi = \pi_d = \langle \bullet, d \rangle$ for some d.
- To check that $d = (W, W \otimes W)$ for some W, V picks $q', q'' \in \mathbf{F}^S$ at random.
  - Let $a = (q', \overrightarrow{0})$, $\overrightarrow{0} \in \mathbf{F}^{S^2}$.
  - Let $b = (q'', \overrightarrow{0})$, $\overrightarrow{0} \in \mathbf{F}^{S^2}$.
  - Let $c = (\overrightarrow{0}, q' \otimes q'')$, $\overrightarrow{0} \in \mathbf{F}^{S^2}$.
- V checks that $\pi(a) \bullet \pi(b) = \pi(c)$.
- Proof of completeness of this check:
  - If $d = (W, W \otimes W)$ then the check will pass because:

$\pi(a) = \langle W, q' \rangle$ and $\pi(b) = \langle W, q'' \rangle$, so $\pi(a) \bullet \pi(b) = \sum_{i=1}^{s} \sum_{j=1}^{s} W_i q'_i W_j q''_j$

while $\pi(c) = \sum_{i=1}^{s} \sum_{j=1}^{s} W_i W_j q_i' q_j''$.

Checking 2: Assuming π is Linear, Check That it is of the Form $\pi_{(W, W\otimes W)}$ for some W.

- Since π is linear, $\pi = \pi_d = \langle \bullet, d \rangle$ for some d.
- To check that $d = (W, W \otimes W)$ for some W, V picks $q', q'' \in \mathbf{F}^S$ at random.
  - Let $a = (q', \overrightarrow{0}),\ \overrightarrow{0} \in \mathbf{F}^{S^2}$.
  - Let $b = (q'', \overrightarrow{0}),\ \overrightarrow{0} \in \mathbf{F}^{S^2}$.
  - Let $c = (\overrightarrow{0}, q' \otimes q''),\ \ \overrightarrow{0} \in \mathbf{F}^{S^2}$.
- V checks that $\pi(a) \bullet \pi(b) = \pi(c)$.
- Proof of soundness of this check:
  - If $d \neq (W, W \otimes W)$ for any W, then $\pi(a) \bullet \pi(b)$ and $\pi(c)$ are both multilinear polynomials in the entries of q' and q'', and these polynomials are not equal.
  - So Schwartz-Zippel implies that the test will fail with probability at least $1 - 2S/|\mathbf{F}|$.

Checking 3: Assuming $\pi = \pi_{(W,W\otimes W)}$, Check That W Satisfies All Constraints Required By A Valid Transcript.

- V needs to check that $Q_i(W) = 0$ for all constraints i.
- V picks $\alpha_1, \alpha_2, \ldots$ at random from $\mathbf{F}$ and checks whether $\sum_i \alpha_i Q_i(W) = 0$.

> This is a degree 2 polynomial in the entries of W, i.e., a linear combination of the entries of $(W, W \otimes W)$. So it can be evaluated with a single query to $\pi = \pi_{(W,W\otimes W)}$.

- Completeness of this step is obvious (if W satisfies all constraints, the test will pass).
- Proof of Soundness: If W does not satisfy all constraints, then $\sum_i \alpha_i Q_i(W)$ is a degree 1 polynomial in the α_i's, so by Schwartz-Zippel, $\sum_i \alpha_i Q_i(W) \neq 0$ with probability at least $1 - 1/|\mathbf{F}|$ over the random choice of the α_i's.

A Linear PCP of Size $|F|^{O(S)}$ [Gennaro, Gentry, Parno, Raykova, 2013]

A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{o(S)}$ AKA Quadratic Arithmetic Programs (QAPs)

- Same setup as [IKO 2007]. Recall:
- Fix a circuit C taking explicit input x and non-deterministic input w, with claimed outputs y.
- Call a vector $W \in \mathbf{F}^S$ a transcript for C.
  - Say W is a **correct** transcript for input x if:
    - $W_a - x_a = 0$ for all input gates a.
    - $W_a - y_a = 0$ for all output gates a.
    - $W_a - (W_{\text{in}_1(a)} + W_{\text{in}_2(a)}) = 0$ for all addition gates a.
    - $W_a - (W_{\text{in}_1(a)} \bullet W_{\text{in}_2(a)}) = 0$ for all multiplication gates a.

A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{o(S)}$ AKA Quadratic Arithmetic Programs (QAPs)

- Same setup as [IKO 2007]. Recall:
- Fix a circuit C taking explicit input x and non-deterministic input w, with claimed outputs y.
- Call a vector $W \in \mathbf{F}^S$ a transcript for C.
  - Say W is a **correct** transcript for input x if:
    - $W_a - x_a = 0$ for all input gates a.
    - $W_a - y_a = 0$ for all output gates a.
    - $W_a - (W_{\text{in}_1(a)} + W_{\text{in}_2(a)}) = 0$ for all addition gates a.
    - $W_a - (W_{\text{in}_1(a)} \bullet W_{\text{in}_2(a)}) = 0$ for all multiplication gates a.
  - In all cases, constraint is of the form:

$$f_{1,i}(W) \bullet f_{2,i}(W) - f_{3,i}(W) = 0$$

    for some linear functions $f_{1,i}(W), f_{2,i}(W),$ and $f_{3,i}(W)$.
  - Next two slides are devoted to the following goals.
    - Given any transcript W, identify a polynomial $g_{x,W}(t)$ such that

$$W \text{ satisfies all constraints } \Leftrightarrow g_{x,W}(t) \text{ vanishes on } H.$$

    - Develop an efficient **proof** that $g_{x,W}(t)$ vanishes on H.

A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{o(S)}$ AKA Quadratic Arithmetic Programs (QAPs)

- Let $H = \{\sigma_1, \ldots, \sigma_m\}$ be an arbitrary set of distinct values in $\mathbf{F}$.

- Lemma (**): Let $h_H(t) = \prod_{i=1}^{m}(t - \sigma_i)$. Let $g(t)$ be any univariate polynomial of degree d over $\mathbf{F}$. Then:

$$g(\sigma_i) = 0 \text{ for all } \sigma_i \in H$$

$$\Leftrightarrow$$

$\exists$ a polynomial h^* of degree at most $d - m$ such that $g(t) = h_H(t) \bullet h^*(t)$.

A Linear PCP For Non-Deterministic Circuit Evaluation of Size $|\mathbf{F}|^{O(S)}$ AKA Quadratic Arithmetic Programs (QAPs)

- Recall: Constraint i is of the form:
  $f_{1,i}(W) \bullet f_{2,i}(W) - f_{3,i}(W) = 0.$
- For each gate a in C, define three univariate polynomials $A_a, B_a,$ and C_a of degree m-1 through interpolation:

$$A_a(\sigma_i) = \text{coefficient of } W_a \text{ in } f_{1,i}.$$
$$B_a(\sigma_i) = \text{coefficient of } W_a \text{ in } f_{2,i}.$$
$$C_a(\sigma_i) = \text{coefficient of } W_a \text{ in } f_{3,i}.$$

- Similarly, define 3 final polynomials of degree m-1 through interpolation:

$$A_a^{'}(\sigma_i) = \text{constant coefficient in } f_{1,i}.$$
$$B_a^{'}(\sigma_i) = \text{constant coefficient in } f_{2,i}.$$
$$C_a^{'}(\sigma_i) = \text{constant coefficient in } f_{3,i}.$$

$A_a(\sigma_i) =$ coefficient of W_a in $f_{1,i}$.
$B_a(\sigma_i) =$ coefficient of W_a in $f_{2,i}$.
$C_a(\sigma_i) =$ coefficient of W_a in $f_{3,i}$.

$A_a'(\sigma_i) =$ constant coefficient in $f_{1,i}$.
$B_a'(\sigma_i) =$ constant coefficient in $f_{2,i}$.
$C_a'(\sigma_i) =$ constant coefficient in $f_{3,i}$.

- Recall: Constraint i is of the form:

$$f_{1,i}(W) \bullet f_{2,i}(W) - f_{3,i}(W) = 0.$$

$A_a(\sigma_i) =$ coefficient of W_a in $f_{1,i}$.
$B_a(\sigma_i) =$ coefficient of W_a in $f_{2,i}$.
$C_a(\sigma_i) =$ coefficient of W_a in $f_{3,i}$.

$A_a'(\sigma_i) =$ constant coefficient in $f_{1,i}$.
$B_a'(\sigma_i) =$ constant coefficient in $f_{2,i}$.
$C_a'(\sigma_i) =$ constant coefficient in $f_{3,i}$.

- Recall: Constraint i is of the form:

$$f_{1,i}(W) \bullet f_{2,i}(W) - f_{3,i}(W) = 0.$$

- Define:

$$g_{x,W}(t) = \Big(\big(\sum_{\text{gates } a} W_a \bullet A_a(t)\big) + A'(t)\Big) \bullet \Big(\big(\sum_{\text{gates } a} W_a \bullet B_a(t)\big) + B'(t)\Big) - \Big(\big(\sum_{\text{gates } a} W_a \bullet C_a(t)\big) + C'(t)\Big)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on $H \overset{\text{Lemma (**)}}{\Leftrightarrow}$
(Key Condition): $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- Define:

$$g_{x,W}(t) = \left(\left(\sum_{\text{gates } a} W_a \bullet A_a(t)\right) + A'(t)\right) \bullet \left(\left(\sum_{\text{gates } a} W_a \bullet B_a(t)\right) + B'(t)\right) - \left(\left(\sum_{\text{gates } a} W_a \bullet C_a(t)\right) + C'(t)\right)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on H $\Leftrightarrow$

(Key Condition): $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- Define:

$$g_{x,W}(t) = \Big(\big(\sum_{\text{gates } a} W_a \bullet A_a(t)\big) + A'(t)\Big) \bullet \Big(\big(\sum_{\text{gates } a} W_a \bullet B_a(t)\big) + B'(t)\Big) - \Big(\big(\sum_{\text{gates } a} W_a \bullet C_a(t)\big) + C'(t)\Big)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on $H \Leftrightarrow$

(Key Condition): $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- To check (Key Condition), it suffices for V to pick a random $r \in \mathbb{F}$ and check that $g_{x,W}(r) = h_H(r) \bullet h^*(r)$.

- Define:

$$g_{x,W}(t) = \Big(\big(\sum_{\text{gates } a} W_a \bullet A_a(t)\big) + A'(t)\Big) \bullet \Big(\big(\sum_{\text{gates } a} W_a \bullet B_a(t)\big) + B'(t)\Big) - \Big(\big(\sum_{\text{gates } a} W_a \bullet C_a(t)\big) + C'(t)\Big)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on $H \Leftrightarrow$

(Key Condition) : $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- To check (Key Condition), it suffices for V to pick a random $r \in \mathbf{F}$ and check that $g_{x,W}(r) = h_H(r) \bullet h^*(r)$.
- Define honest proof π to be $\pi_W = \langle \bullet, W \rangle$ and π_d, where $d =$ (the $S+1$ coefficients of h^*)

- Define:

$$g_{x,W}(t) = \Big(\big(\sum_{\text{gates } a} W_a \bullet A_a(t)\big) + A'(t)\Big) \bullet \Big(\big(\sum_{\text{gates } a} W_a \bullet B_a(t)\big) + B'(t)\Big) - \Big(\big(\sum_{\text{gates } a} W_a \bullet C_a(t)\big) + C'(t)\Big)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on H $\Leftrightarrow$

(Key Condition): $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- To check (Key Condition), it suffices for V to pick a random $r \in \mathbb{F}$ and check that $g_{x,W}(r) = h_H(r) \bullet h^*(r)$.

- Define honest proof π to be $\pi_W = \langle \bullet, W \rangle$ and π_d, where $d =$ (the $S+1$ coefficients of h^*)

V can compute from π by evaluating π_d at the point $(1, r, r^2, \ldots, r^S)$.

- Define:

$$g_{x,W}(t) = \left(\left(\sum_{\text{gates } a} W_a \bullet A_a(t)\right) + A'(t)\right) \bullet \left(\left(\sum_{\text{gates } a} W_a \bullet B_a(t)\right) + B'(t)\right) - \left(\left(\sum_{\text{gates } a} W_a \bullet C_a(t)\right) + C'(t)\right)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on H $\Leftrightarrow$

(Key Condition): $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- To check (Key Condition), it suffices for V to pick a random $r \in \mathbf{F}$ and check that $g_{x,W}(r) = h_H(r) \bullet h^*(r)$.

- Define honest proof π to be $\pi_W = \langle \bullet, W \rangle$ and π_d, where d = (the $S+1$ coefficients of h^*)

V can compute from π by evaluating π_d at the point $(1, r, r^2, \ldots, r^S)$.

V can compute from π by querying π_W at three points: $(A_1(r), \ldots, A_S(r)), (B_1(r), \ldots, B_S(r))$, and $(C_1(r), \ldots, C_S(r))$.

- Define:

$$g_{x,W}(t) = \Big(\big(\sum_{\text{gates } a} W_a \bullet A_a(t)\big) + A'(t)\Big) \bullet \Big(\big(\sum_{\text{gates } a} W_a \bullet B_a(t)\big) + B'(t)\Big) - \Big(\big(\sum_{\text{gates } a} W_a \bullet C_a(t)\big) + C'(t)\Big)$$

- Then: W satisfies all constraints $\Leftrightarrow g_{x,W}(t)$ vanishes on $H \Leftrightarrow$

(Key Condition): $g_{x,W}(t) = h_H(t) \bullet h^*(t)$ for some h^* of degree at most S.

- To check (Key Condition), it suffices for V to pick a random $r \in \mathbf{F}$ and check that $g_{x,W}(r) = h_H(r) \bullet h^*(r)$.
- Define honest proof π to be $\pi_W = \langle \bullet, W \rangle$ and π_d, where $d =$ (the $S+1$ coefficients of h^*)

V can compute on her own in time O(S).

V can compute from π by evaluating π_d at the point $(1, r, r^2, \ldots, r^S)$.

V can compute from π by querying π_W at three points: $(A_1(r), \ldots, A_S(r)), (B_1(r), \ldots, B_S(r))$, and $(C_1(r), \ldots, C_S(r))$.

1- and 2-Message Arguments

Micali's Argument System in RO Model

- [Micali 1994] gave a one-message argument system in the Random Oracle model using the Fiat-Shamir heuristic to remove interaction from Kilian's protocol.
  - Idea: Use random oracle in place of CRHF when building the Merkle tree, and to choose V's PCP queries.

Micali's Argument System in RO Model

- [Micali 1994] gave a one-message argument system in the Random Oracle model using the Fiat-Shamir heuristic to remove interaction from Kilian's protocol.
  - Idea: Use random oracle in place of CRHF when building the Merkle tree, and to choose V's PCP queries.
- But one-message argument systems are impossible (for non-trivial languages) in the standard model.
  - At least, if you want security against non-uniform cheating provers.
- Attention therefore turns to 2-message argument systems in standard model.

Micali's Argument System in RO Model

- [Micali 1994] gave a one-message argument system in the Random Oracle model using the Fiat-Shamir heuristic to remove interaction from Kilian's protocol.
  - Idea: Use random oracle in place of CRHF when building the Merkle tree, and to choose V's PCP queries.
- But one-message argument systems are impossible (for non-trivial languages) in the standard model.
  - At least, if you want security against non-uniform cheating provers.
- Attention therefore turns to 2-message argument systems in standard model.
  - Goal: obtain same efficiency as 4-message argument systems obtained by combining commit/reveal protocol for linear functions from [IKO 2007] with GGPR's linear PCP.
    - Such arguments are called SNARGs (Succinct Non-interactive ARGuments).
    - "Succinct" refers to efficient support for non-determinism, i.e., P can convince V it holds a non-deterministic witness w that x in L, without sending w to V.
- There are obstacles to basing such 2-message arguments on standard (i.e., falsifiable) assumptions. Existing constructions use non-falsifiable ones.

2-Message Arguments from Linear PCPs

- Idea: Replace the 4-message commit/reveal protocol for linear functions of [IKO 2007] with a 2-message one.
  - Rather than use an additively homomorphic encryption scheme, use a stronger primitive: "linear-only" encryption.
  - Roughly, this is an encryption scheme that is:
    - Semantically secure
    - Additively Homomorphic
    - "linear-only" i.e., P is forced to behave in a linear manner.
      - More formally, given ciphertexts $c_1 = \mathrm{Enc}(a_1),\ldots,c_k = \mathrm{Enc}(a_k)$, it is assumed that the only way to efficiently compute a new ciphertext c' in the image of Enc is to "know" $\beta,\alpha_1,\ldots,\alpha_k$ such that $c' = \mathrm{Enc}(\beta+\alpha_1 \bullet a_1 + \ldots + \alpha_k \bullet a_k)$.
      - Actually formalized with an extractability guarantee.

2-Message Arguments from Linear PCPs: Protocol Details

- V simulates the linear PCP verifier, sending queries $q_1,\ldots,q_k$ to P **encrypted under a linear-only encryption scheme.**
- P uses the homomorphism property to compute $\mathrm{Enc}(\pi(q_1)),\ldots,\mathrm{Enc}(\pi(q_k))$.
- P sends these values to V, who decrypts them and simulates the PCP verifier's accept/reject process.
- Soundness proof sketch: By linear-only property, when P convinces V to accept, P must "know" an affine function Ω such that $\Omega(q_1),\ldots,\Omega(q_k)$ convinces the PCP verifier. By semantic security of Enc, Ω must be independent of the queries V sent to P. Hence, Ω must actually be a linear PCP proof. Soundness now follows from soundness of the linear PCP.
- Completeness is obvious.

Argument Systems Satisfying Additional Properties

- Proof of Knowledge (PoK): Whenever P can convince V that input x is in language L, there must exist a polynomial time extractor algorithm E that, given access to P, can output a witness w that x is in L.
  - Important in crypto settings where there may be many valid solutions/witnesses, but only one "correct" one.
  - E.g. Suppose V knows only a Merkle-hash h(x) of an input x, and wants to make sure P correctly executed some computation C on x to produce some output y.
  - A SNARG without PoK can only guarantee that there **exists** a x' such that C(x')=y and h(x')=h(x). This is not meaningful since such an x' could always exist, as there are collisions under h (even though they are hard to find).
  - But if the argument systems also satisfies PoK, then P must **know** such a x', not just that such an x' exists. And by collision-resistance of h, x' actually must equal to true input x.
- The arguments systems I've described do satisfy PoK.

Argument Systems Satisfying Additional Properties

- Public Verifiability. The linear-only encryption-based SNARG from before is not publicly verifiable, since V's secret key is required to decrypt P's messages and thereby execute the PCP verifier's checks.
- Instead, use a "linear-only one-way encoding" scheme. This satisfies 4 rough properties.
  - Additive homomorphism, to enable P to compute the "encoding" of $\text{Enc}(\pi(q_1)),\ldots,\text{Enc}(\pi(q_k))$ from $q_1,\ldots,q_k$.
  - Linear-only.
  - Allows any party to execute the linear PCP verifier's decision predicate on encoded answers, **without** decoding.
    - The candidate linear-only encodings in the literature only support a limited class of verifier decision predicates. The PCP verifier's test must be of the form "Test if $Q(\pi(q_1),\ldots,\pi(q_k))=0$", where Q is a quadratic polynomial. Fortunately, GGPR's PCP verifier satisfies this property.
  - "One-Way" Property: ensures that, given encodings of any set of queries $q_1,\ldots,q_k$, P cannot "learn" a set of answers that would cause V's check to pass, unless P actually knows a linear PCP proof π causing the PCP verifier to accept.
- Candidate linear-only one-way encoding schemes are based on knowledge of exponent assumptions in bilinear groups.

Implementations

Implementations of Argument Systems

- 4-message argument system based on [IKO 2007]'s commit/reveal protocol for linear functions and linear PCP of size $|\mathbf{F}|^{O(S^2)}$ implemented and refined by [SMBW 2012] and [SVPBBW 2012] ("Pepper" and "Ginger").
- 4-message argument system based on [IKO 2007]'s commit/reveal protocol for linear functions and [GGPR 2013]'s linear PCP of size $|\mathbf{F}|^{O(S)}$ implemented by [SBVBPW 2013] ("Zaatar").
- 2-message argument system based on the above theory works (but with stonger cryptographic primitives) implemented by [PHGR 2013] ("Pinocchio"), and also by [BSCGTV13] ("SNARKs for C").
- Many subsequent refinements: [VSBW 2013], [BFRSBW 2013], [WSRBW 2015], [BSCTV14], [BSCGTV15], [CFHKKNPZ15], etc.
- See [Blumberg and Walfish, CACM 2016] for a now-slightly-out-of-date comparison of implementations (including interactive proofs).

References

- [Babai 1985] L. Babai. Trading Group Theory for Randomness. STOC, 1985.
- [BC 2012] N. Bitansky and A. Chiesa. Succinct Arguments from Multi-prover Interactive Proofs and Their Efficiency Benefits. CRYPTO 2012.
- [BSCGT 2013] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer. On the Concrete Efficiency Threshold of Probabilistically Checkable Proofs. STOC 2013.
- [BSCGT 2013B] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer. Fast Reductions from RAMs to Delegatable Succinct Constraint Satisfaction Problems. ITCS 2013.
- [BSCGTV 2013] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, M. Virza. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. CRYPTO 2013.
- [BSCGTV 2015] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, M. Virza. Secure Sampling of Public Parameters for Succinct Zero-Knowledge Proofs. Oakland 2015.
- [BSCTV 2014A] E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza. Succinct Non-interactive Zero Knowledge For a Von Neumann Architecture. USENIX Security 2014.
- [BSCTV 2014B] E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza. Scalable Zero Knowledge via Cycles of Eliptic Curves. CRYPTO 2014.
- [BCIOP 2013] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, O. Paneth, Succinct Non-Interactive Arguments via Linear Interactive Proofs, TCC 2013.

References

- [BFL 1991] L. Babai, L. Fortnow, C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. Computational Complexity, 1991. Preliminary version in FOCS, 1990.
- [BSVRBW 2013]. B. Braun, A. Feldman, Z. Ren, S. Setty, A.J. Blumberg, M. Walfish. Verifying Computations with State. SOSP 2013.
- [BSGHSV 2005] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, S. Vadhan. Short PCPs Verifiable in Polylogarithmic Time. CCC 2005.
- [BSS 2008] E. Ben-Sasson, M. Sudan, Short PCPs with Polylog Query Complexity. SICOMP 2008. Preliminary version in STOC 2005.
- [BTVW 2014] A. J. Blumberg, J. Thaler, V. Vu, M. Walfish. Verifiable Computation Using Multiple Provers. Manuscript, 2014.
- [BW 2015]. A. J. Blumberg and M. Walfish. Verifying Computations Without Reexecuting Them. CACM, 2015.
- [CFHKKNPZ 2015]. C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, S. Zahur. Geppetto: Versatile Verifiable Computation. Oakland 2015.
- [CMT 2012] G Cormode, M. Mitzenmacher, J. Thaler. Practical Verified Computation with Streaming Interactive Proofs. ITCS 2012.
- [FRS 1990] L. Fortnow, R. Rompel, M. Sipser. On the Power of Multi-Prover Interactive Interactive Protocols. Theoretical Computer Science, 1994. Preliminary version in Structure in Complexity Theory, 1990.

References

- [GGPR 2013] R. Gennaro, C. Gentry, B. Parno, M. Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. Eurocrypt 2013.
- [GKR 2015] S. Goldwasser. Y. Tauman Kalai, G. Rothblum. Delegating Computation: Interactive Proofs for Muggles. J. ACM, 2015. Preliminary version in STOC 2008.
- [GMR 1989] S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. SICOMP, 1989. Preliminary version in STOC 1985.
- [GS 1989] Y. Gurevich and S. Shelah. Nearly Linear Time. In Symposium on Logical Foundations of Computer Science, 1989.
- [IKO 2007] Y. Ishai, E. Kushilevitz, R. Ostrovsky. Efficient Arguments without Short PCPs. CCC 2007.
- [Kilian 1992] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). STOC 1992.
- [KRR 2014] Y. Tauman Kalai, R. Raz, R. Rothblum. How to Delegate Computations: The Power of No-Signaling Proofs. STOC 2014.
- [LFKN 1992] C. Lund, F. Fortnow, H. Karloff, N. Nisan. Algebraic Methods for Interactive Proof Systems. J. ACM 1992. Preliminary version in FOCS, 1990.
- [PHGR 2013]. B. Parno, J. Howell, C. Gentry, M. Raykova. Pinnochio: Nearly Practical Verifiable Computation. Oakland 2013.

References

- [Robson 1991] JM Robson. An O(T log T) Reduction from RAM Computations to Satisfiability. Theoretical Computer Science, 1991.
- [SBVBPW 2013]. S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, M. Walfish. Resolving the conflict between generality and plausibility in verified computation. EuroSys 2013.
- [SMBW 2012] S. Setty, R. McPherson, A. J. Blumberg, M. Walfish, Making argument systems for outsourced computation practical (sometimes). NDSS 2012.
- [SVPBBW 2012] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, M. Walfish, Taking proof-based verified computation a few steps closer to practicality. USENIX Security 2012.
- [Shamir 1992] A. Shamir. IP=PSPACE. J ACM. 1992.
- [Thaler 2013] J. Thaler. Time-Optimal Interactive Proofs for Circuit Evaluation. CRYPTO 2013.
- [TRMP 2012] J. Thaler, M. Roberts, M. Mitzenmacher, H. Pfister. Verifiable Computation with Massively Parallel Interactive Proofs. HotCloud 2012.
- [WHGsW 2016]. R. Wahby, M. Howald, S. Garg, a shelat, M. Walfish. Verifiable ASICs. Oakland 2016.
- [WSRBW 2015]. R. Wahby, S. Setty, Z. Ren, A. J. Blumberg, M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. NDSS 2015.
- [VSBW 2013]. V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. Oakland 2013.