Verifying computations without re-executing them

The GKR Protocol

Myungsun Kim

Department of Information Security
The University of Suwon

September 27, 2019

@Hanyang University

Contents

USW THE UNIVERSITY OF SUWON

1. Introduction

§1.1 Motivations

A classic problem in computation theory

The question

How can a single PC check a herd of supercomputers with unreliable software and untested hardware?

[BFLS91]

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

– Babai, Fortnow, Levin, and Szegedy: *Checking computations in polylogarithmic time*. STOC, 1991.

Due to Babai et al.

In an asymmetric setting (a PC vs. dozens of Supercomputers) with available resources,

Delegating some expensive tasks to the cloud:

- Pros
  - Fixed cost & depreciation savings
  - Supplies expenses savings

  . . .
- Cons
  - Black-box in faults: mis-config., corruption of data in transit, H/W problems, & mal. op.'s

  . . .

A central issue

Is **an output** from the cloud as a response to a service request **correct?**

Due to Babai et al.

Possible answers

1. Replicate of delegations
2. Audit: checking the responses in a small sample
3. Trust H/W (a.k.a TPM) & attestation

. . .

Another possible solution–($\star$)

- Supercomputers return some results along with a proof that the results were correctly computed
- Checking the proof is cheap (vs. locally redoing the computation)

A bit more realistic scenario

Consider a simple DB with a table [EN11], that

- keeps track of employee's name, social security number, address, salary, and her working department

EMPLOYEE

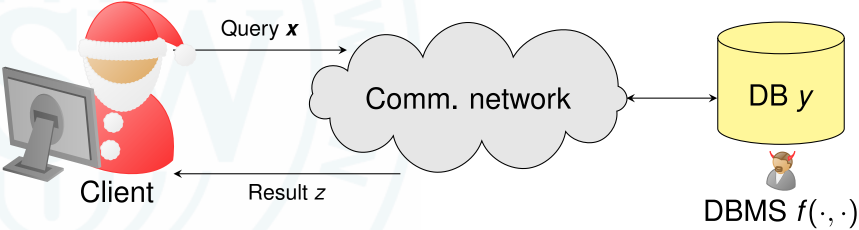
Fname	Lname	Ssn	Address	Salary	Dno
John	Smith	1234	73 Fondren, Houston, TX	30000	5
Franklin	Wong	3334	68, Voss, Houston, TX	40000	5
Alicia	Zelaya	9998	21, Castle, Spring, TX	25000	4
Jennifer	Wallace	9876	91, Berry, Bellaire, TX	43000	4
Ramesh	Narayan	6688	75, Oak, Humble, TX	38000	5
Joyce	English	4534	56, Rice, Houston, TX	25000	5
Almad	Jabbar	9879	80, Dallas, Houston, TX	31000	4
James	Borg	8886	50, Stone, Houston, TX	55000	1

A bit more realistic scenario

A standard DBMS provides DB query services

- Search (e.g., `select-from-where`)
- Update (e.g., `insert` or `delete`)

by running the DBMS



A bit more realistic scenario

When the client submits a query $\boldsymbol{x}$ to the DBMS

```sql
SELECT Fname, Lname, Salary, Address
FROM   EMPLOYEE
WHERE  Salary = (SELECT MAX(E1.Salary)
                 FROM EMPLOYEE E1, PROJECT P
                 WHERE P.Pname = "Design"
                 AND P.Dnum = E1.Dno
                 AND E1.Age >= 37)
ORDER BY Fname, Lname;
```

$z =$

Fname	Lname	Salary	Address
John	Smith	30000	73, Fondren, Houston, TX
Joyce	English	25000	56, Rice, Houston, TX

A bit more realistic scenario

Main security concerns

1. Correctness: $z = f_y(\boldsymbol{x})$?
2. Privacy: The DBMS learns my secret data $\boldsymbol{x}$, y and/or z

☺ From homomorphic encryption (E, D) (e.g., [CKK17]):

- encrypt a DB y into $\bar{y} \leftarrow E(y)$
- store them to the remote server;
- encrypt a query $\boldsymbol{x}$ into $\bar{\boldsymbol{x}} \leftarrow E(\boldsymbol{x})$ and send $\bar{\boldsymbol{x}}$ to the server
- evaluate f at $\bar{\boldsymbol{x}}, \bar{y}$ as $\bar{z}$

☞ How to verify if $z^* \leftarrow D(\bar{z})$ is a correct query result?

- $z^* = f_y(\boldsymbol{x})$?
- E.g., vSQL in [ZGK+17], but over clear DB

A short history in verifiable computation

Purely theoretical

(e.g., [Sha90])

2007∼2008

Cloud computing

(e.g., Amazon AWS, Google AE) $\Leftarrow$ [IKO07,GKR08]

SNARG/SNARK

Implementations (e.g., Pepper, Pinocchio)

ZKB++, Bulletproof, Hyrax, & Libra, etc

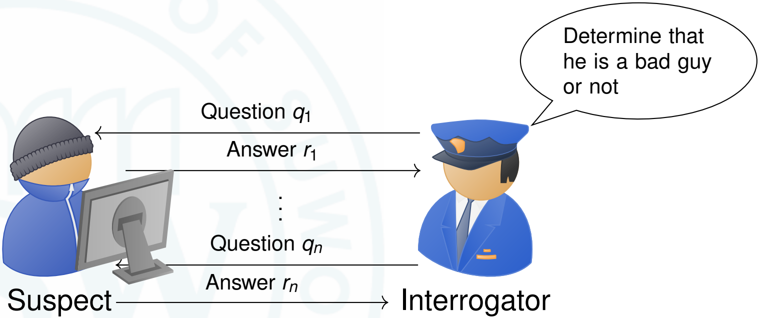
§1.2 The Abstract Model

A system model

A verifiable computation (VC) system:

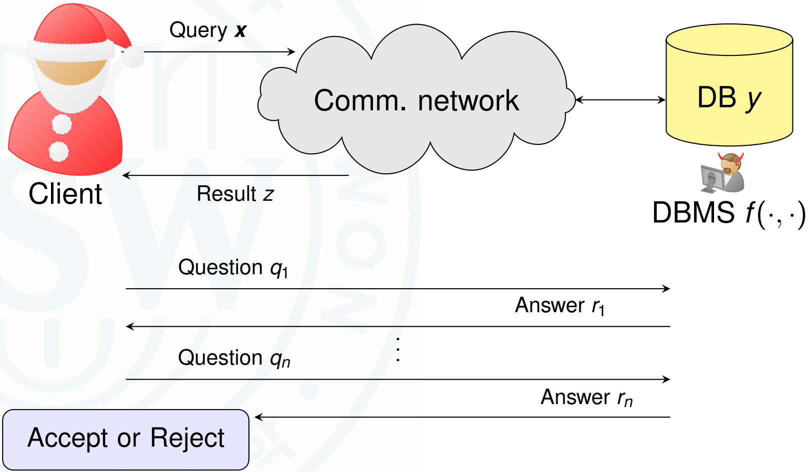
- Goal: Proving integrity to the verifier,
  - Prover's auxiliary input is secret $\Rightarrow$ privacy-preserving integrity
- Participating players
  - Verifier: Wish to know the output from a computation p on an input x
  - Prover: Wish to efficiently convince the verifier of his computing the output y, while returning y and a certificate of correctness π
- Adversarial model
  - The verifier is assumed to be honest, but the prover is dishonest

A way to enforce security



A way to enforce security

Applying to our scenario



A way to enforce security

The key to the powerfulness of 🗨 (the mechanic before) is the **combination** of V's randomness and (P, V)'s interaction, for the purpose of amplifying soundness guarantee by

- Tossing random coins: even a supercomputer does not know but just guess
  - The distinction bet. public & internal coins is not something [GS86]
- Interacting many times: reduce the small probability to be negligible

The system model

Interactive proof (IP) systems 🗨: Informal

- Participants
  - Prover P (modeling the DBMS or the suspect)
  - Verifier V (modeling the client or the interrogator)
- Activities
  1. P solves a problem on a given input, modeling the question "Is the result z correct?"
  2. P shows the answer
  3. P **proves** to V that the answer is correct
- Security requirements
  - **Completeness** states that an honest P always can convince V to accept
  - **Soundness** states that V is able to catch a cheating P with high probability

Interactive Proof Systems

Classifications of proof systems:

- Interactive proofs (IPs) [GMR89,Bab85]: ensure information-theoretically soundness
- Argument systems [BCC88]: ensure soundness just against polynomial time P's
- Multi-prover IPs (MIPs)
  - MIP [BGKW88]: similar to IP, but there are 2 or more P's Note. In PCP, the answer (formally a proof) is static, but may have super-polynomial size
- Zero-knowledge proofs and its argument variants: ensure that no information are revealed to V except the validity of answer being given

Interactive Proof Systems

Zero-knowledge proofs (ZKPs)

- Some IP and argument systems $\rightleftarrows$ ZKPs (or ZK arguments)
  - *P* with an ingenious know-how to persuade *V*
  - *P* in such IPs (and arguments) reveals no knowledge to *V* other than the correctness of his proof
- Additional applications from IPs of achieving ZK
  - The sum-check protocol [LFKN92]: a building block in efficient delegating computation
    - during execution, reveals to *V* partial sums over *V*'s randomness
    - the partial sums $\not\Rightarrow$ ZK-sum-check

☺ with higher costs than expected

Definition

A formal definition of IP

Definition. *For a function* $f : \{0,1\}^n \to R$*, an* *interactive proof system for* f *consists of a PPT verifier* V *and a prover* P *with a common input* $\boldsymbol{x} \in \{0,1\}^n$*. After poly. many comm.'s, denoted by* $\boldsymbol{t} := (V(\boldsymbol{r}), P)(\boldsymbol{x})$*,* V *outputs* $\{0,1\}$*. The IP system has* δ_c, δ_s *if the two conditions hold.*

1. *(Completeness)* $\exists P$ *such that* $\forall \boldsymbol{x} \in \mathcal{L}$*:*
$$\Pr[\mathtt{out}_V(V(\boldsymbol{r}), P)(\boldsymbol{x}) \to 1)] \geq 1 - \delta_c$$
2. *(Soundness)* $\forall \boldsymbol{x} \neq \mathcal{L}$ *and* P^**:*
$$\Pr[\mathtt{out}_V(V(\boldsymbol{r}), P^*)(\boldsymbol{x}) \to 1)] \leq \delta_s$$

where $\boldsymbol{r}$ *is* V*'s internal random; the IP for* f *is equivalent to the IP for language* $\mathcal{L}_f = \{(\boldsymbol{x}, y) | y \leftarrow f(\boldsymbol{x})\}$

USW THE UNIVERSITY OF SUWON

2. The GKR Protocol

§2.1 Warming-up

A warmup example

Freivalds' algorithm

- For $A, B \in \mathcal{M}_{n \times n}$ over $\mathbb{F}_p$, the fastest algorithm to compute $A \cdot B$ runs in $O(n^{2.3728639})$ [LG14]
- The Freivalds algorithm [Fre77]
  1. P opens $C \leftarrow A \cdot B$
  2. V chooses $\boldsymbol{r} = (r, r^2, \ldots, r^n)$ where $r \in \mathbb{F}_p$
  3. Compute $y \leftarrow C\boldsymbol{r}$ and $z \leftarrow A \cdot (B\boldsymbol{r})$
  4. Check $y = z$; if yes, accept, reject otherwise
- V of the Freivalds algorithm runs in $O(n^2)$
- If $A \cdot B \neq C$, then for some i: $C_i \neq (A \cdot B)_i$ and

$$\Pr[(C\boldsymbol{r})_i = (A \cdot (B\boldsymbol{r}))_i] \leq n/p$$

A warmup example

Analysis

Fact

For any two distinct polynomial p_a and p_b of degree at most n with coefficients in $\mathbb{F}_p$, $p_a(x) = p_b(x)$ for at most n values of x in $\mathbb{F}_p$ where $p_a(x) = \sum a_i x^i$.

- Let $D = A \cdot B$ and observe only the case $\exists i$ s.t. $C_i \neq D_i$
- View $(C\boldsymbol{r})_i = p_{C_i}(\boldsymbol{r})$ and $A \cdot (B \cdot \boldsymbol{r}) = p_{D_i}(\boldsymbol{r})$
- $\Pr[(C\boldsymbol{x})_i \neq A \cdot (B \cdot \boldsymbol{x})_i] \geq 1 - \frac{n}{p}$
- Apply the fact

§2.3 The Sum-check Protocol

The sum-check protocol

- Goal: Given a v-variate polynomial g over $\mathbb{F}$, to compute the sum

$$H := \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} g(x_1, x_2, \ldots, x_v)$$

- The strategy [LFKN92]
  - $H = \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} g(0, x_2, \ldots, x_v) + \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} g(1, x_2, \ldots, x_v)$
  - Each partial sum is the evaluation of $g(t, x_2, \ldots, x_v)$ at 0 or 1
  - Proceed evaluations incrementally while binding the variable t_j to a random $r_j \in \mathbb{F}$

The sum-check protocol

1. Fix an $H \in \mathbb{F}$
2. 1^{st}-round.
   - P sends the univariate polynomial

$$g_1(t_1) = \sum_{x_2,\ldots,x_v \in \{0,1\}^{v-1}} g(t_1, x_2, \ldots, x_v)$$

   - V checks if $g_1(t_1)$ has the claimed degree and $H = g_1(0) + g_1(1)$; if not rejecting
   - V sends a random $r_1 \in \mathbb{F}$ to P

The sum-check protocol

3. j^{th}-round $(1 < j < v)$
   - P sends the univariate polynomial

$$g_j(t_j) = \sum_{x_{j+1},\ldots,x_v \in \{0,1\}^{v-j}} g(r_1, r_2, \ldots, r_{j-1}, t_j, x_{j+1}, \ldots, x_v)$$

   - V checks if $g_j(t_j)$ has the claimed degree and $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$; if not rejecting
   - V sends $r_j \xleftarrow{\$} \mathbb{F}$ to P

USW THE UNIVERSITY OF SUWON

The sum-check protocol

The sum-check protocol

4. v^{th}-round.
   - P sends the univariate polynomial

$$g_v(t_v) = g(r_1, \ldots, r_{v-1}, t_v)$$

   - V checks if $g_v(t_v)$ has the claimed degree and $g_{v-1}(r_{v-1}) = g_v(0) + g_v(1)$; if not rejecting
   - V picks $r_v \xleftarrow{\$} \mathbb{F}$ and check if $g_v(r_v) = \underbrace{g(r_1, r_2, \ldots, r_v)}_{\text{How efficiently?}}$; if not rejecting
5. V halts or accepts if it has not yet rejected

The sum-check protocol

Efficiency of the SC protocol

- Costs of the sum-check protocol

Communication	Rounds	V's time	P's time
$O(\sum_{i=1}^{v} \deg(g_i))$ $\mathbb{F}$ elt's	v	$O(\sum_{i=1}^{v} \deg(g_i)) + T$	$O(2^v \cdot T)$

- Soundness error $\leq \frac{dv}{|\mathbb{F}|}$ by induction on v & the Fact

§2.4 The Main Protocol

An example

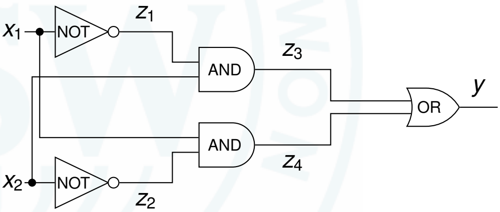
Consider a piece of a C code where x_1, x_2, y are bits

```
/* y= x1 ^ x2 */
void foo(const char* x)
{
    if (x1 != x2) {
        y = 1;
    }
    else {
        y = 0;
    }
    /* do something more */
    return y;
}
```

An example

Representing the program `foo` as a circuit in terms of AND, OR and NOT

- $y = x_1 \oplus x_2$
- $y = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$
- A circuit $\mathcal{C}_{\texttt{foo}}$



Background

Some background: Low degree & multilinear extensions

Notation:
$\mathbb{F}$ is a finite field, and $f : \{0,1\}^v \to \mathbb{F}$ is a function

Definition. *A polynomial $g \in \mathbb{F}[x_1, \ldots, x_v]$ is said to be an* extension *of f if $\forall \boldsymbol{x} \in \{0,1\}^v : g(\boldsymbol{x}) = f(\boldsymbol{x})$.*

Definition. *A multivariate polynomial g is* multilinear *if the degree of g in each variable ≤ 1.*

Lemma. *Any function $f : \{0,1\}^v \to \{0,1\}$ has a* unique multilinear extension *(MLE) over $\mathbb{F}$, denoted by $\widetilde{f}$*

Proof.

Background

Usefulness of low-degree extensions.

- An MLE g of f: a distance-amplifying encoding of f
- $f \neq f'$ at a single input $\Rightarrow \widetilde{g} \neq \widetilde{g}'$ almost everywhere

Example: $f : \{0,1\}^2 \to \mathbb{F}_5$ and $f(0,0) = 1, f(0,1) = 2,$
$f(1,0) = 1, f(1,1) = 4$

- $\widetilde{f}(x_1, x_2) = 2x_1x_2 + x_2 + 1$

	0	1
0	1	2
1	1	4

f

$\xrightarrow{\text{extension}}$

	0	1	2	3	4
0	1	2	3	4	0
1	1	4	2	0	3
2	1	1	1	1	1
3	1	3	0	2	4
4	1	0	4	3	2

$g = \widetilde{f}$

The GKR Protocol

Protocol overview:

- Goal: Compute the value of the output gate(s) of $\mathcal{C}$ for a logspace arithmetic circuit $\mathcal{C}$ whose size S and depth d over n variables
- Costs of the GKR protocol [GKR08]

Communication	Rounds	V's time	P's time
$d \cdot \text{polylog}(S)$ field elt's	$d \cdot \text{polylog}(S)$	$O(n + d \cdot \text{polylog}(S))$	$\text{poly}(S)$

- Protocol activities:
  1. P sends to V the claimed output of $\mathcal{C}$
  2. Bind a claim about level $i \geq 1$ and a claim about level $i+1$ by V's random
  3. At level $d+1$, V locally check if the last claim is consistent with the public input $\boldsymbol{x}$

The GKR Protocol

A bit more details of the GKR protocol

- At the first round, V is given a claim about the value(s) if the output gate(s) of $\mathcal{C}$
  - ☹ V cannot check this claim by herself
- V reduces the claim about the outputs of $\mathcal{C}$ to a claim about the gate values at level 2,
  - using the sum-check protocol
- Do the same thing by level d
- At the $(d+1)$ level, V is given a claim about the inputs $\boldsymbol{x}$ of $\mathcal{C}$
  - ☺ V locally checks the claim using the inputs $\boldsymbol{x}$

The GKR Protocol

Notation

- $\mathcal{C}$: a layered arith. circuit of size s, depth d, and fan-in 2 where 1: the output level & $d+1$: the input level
- s_i : the number of gates at level i and $s_i = 2^{\sigma_i}$ for some $\sigma_i \geq 0$
- $W_i : \{0,1\}^{s_i} \to \mathbb{F}$: take a binary gate label, output the corr. gate's value $\widetilde{W}_i$ which is W_i's MLE version
- $\text{in}_{1,i}, \text{in}_{2,i} : \{0,1\}^{s_i} \to \{0,1\}^{s_{i+1}}$: take the label $\boldsymbol{a}$ of a gate at level i, output the label of each in-neighbor of gate $\boldsymbol{a}$
- $\text{add}_i : \{0,1\}^{s_i+2s_{i+1}} \to \{0,1\}$: a wiring predicate with its MLE $\widetilde{\text{add}}_i$
- $\text{mul}_i : \{0,1\}^{s_i+2s_{i+1}} \to \{0,1\}$: a wiring predicate with its MLE $\widetilde{\text{mul}}_i$

  where a wiring predicate **encodes**
  - which pairs of wires from level $i+1$ are connected to
  - a given gate at level i in $\mathcal{C}$

The GKR Protocol

Details of the GKR protocol

- with d iterations;
- each iteration i starts with P claiming that a value for $\widetilde{W}_i(\boldsymbol{r}_i)$ for V's random $\boldsymbol{r}_i \in \mathbb{F}^{s_i}$

1. 1-round: check the claim about the outputs ($s_0 = 2^{\sigma_0}$) in $\mathcal{C}$
   - V
     - uses $D : \{0, 1\} \to \mathbb{F}$, a function mapping the label of the output gate to the claimed value of that output
     - picks $\boldsymbol{r}_0 \in \mathbb{F}^{\sigma_0}$ and evaluates $\widetilde{D}(\boldsymbol{r}_0)$ in time $O(\sigma_0)$ [VSBW13]
     - checks if $\widetilde{D}(\boldsymbol{r}_0) = \widetilde{W}_0(\boldsymbol{r}_0)$, where $\widetilde{D}(\boldsymbol{r}_0) = \widetilde{W}_0(\boldsymbol{r}_0)$: the MLE of claimed output = the MLE of the corrected outputs when evaluated at a random point $\boldsymbol{r}_0$
   - V can evaluate $\widetilde{W}_0(\boldsymbol{r}_0)$ only with P's interactions

The GKR Protocol

2. i-round ($i \le d$):
   - Goal: reduce the claim about the value of $\widetilde{W}_i(\boldsymbol{r}_i)$ to a claim about $\widetilde{W}_{i+1}(\boldsymbol{r}_{i+1})$ for V's random $\boldsymbol{r}_{i+1} \in \mathbb{F}^{\sigma_{i+1}}$
   - Run the sum-check protocol to a polynomial $f^{(i)}_{\boldsymbol{r}_i}$ derived from $\widetilde{W}_{i+1}, \widetilde{\mathrm{add}}_i$ and $\widetilde{\mathrm{mul}}_i$
   - To check P's claim about $\widetilde{W}_i(\boldsymbol{r}_i)$:
     (a) Express $\widetilde{W}_i(\boldsymbol{r}_i)$ as

$$\boldsymbol{W}_i(\boldsymbol{a}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{\sigma_{i+1}}} \widetilde{\mathrm{add}}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mul}}_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

     (b) Apply to the sum-check protocol to $f^{(i)}_{\boldsymbol{r}_i}$

$$f^{(i)}_{\boldsymbol{r}_i}(\boldsymbol{b},\boldsymbol{c}) = \widetilde{\mathrm{add}}_i(\boldsymbol{r}_i,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) + \widetilde{W}_{i+1}(\boldsymbol{c})) + \widetilde{\mathrm{mul}}_i(\boldsymbol{r}_i,\boldsymbol{b},\boldsymbol{c})(\widetilde{W}_{i+1}(\boldsymbol{b}) \cdot \widetilde{W}_{i+1}(\boldsymbol{c}))$$

The GKR Protocol

3. The final round:
   - V
     - The vector of gate values at level $d + 1$ is the input $\boldsymbol{x}$ of $\mathcal{C}$
     - Locally evaluate $\boldsymbol{W}_{d+1}(\boldsymbol{r}_{d+1})$ in time $O(n)$

The GKR Protocol

Costs of the GKR protocol

- V's time
  - Run the sum-check to evaluate $f^{(i)}_{\boldsymbol{r}_i}(\boldsymbol{b}, \boldsymbol{c})$ which is $2\sigma_{i+1}$-variate & of deg. at most 2 in each var. $\Rightarrow$ $2\sigma_{i+1}$-round at level i
  - the total time cost is $O(n + d \log S + \mathrm{m})$ where
    - processing time $d \log S$ messages between V and P
    - m for evaluating $\widetilde{\mathrm{add}}_i, \widetilde{\mathrm{mul}}_i$ & n for evaluating $\boldsymbol{W}_{d+1}(\boldsymbol{r}_{d+1})$
- P's time: d times running of the sum-check protocol for MLEs

Thank you & Questions?