Generalizing Homomorphic MACs for Arithmetic Circuits

Dario Catalano
Università di Catania
Italy

Dario Fiore
IMDEA Software Institute
Spain

Rosario Gennaro
CUNY
USA

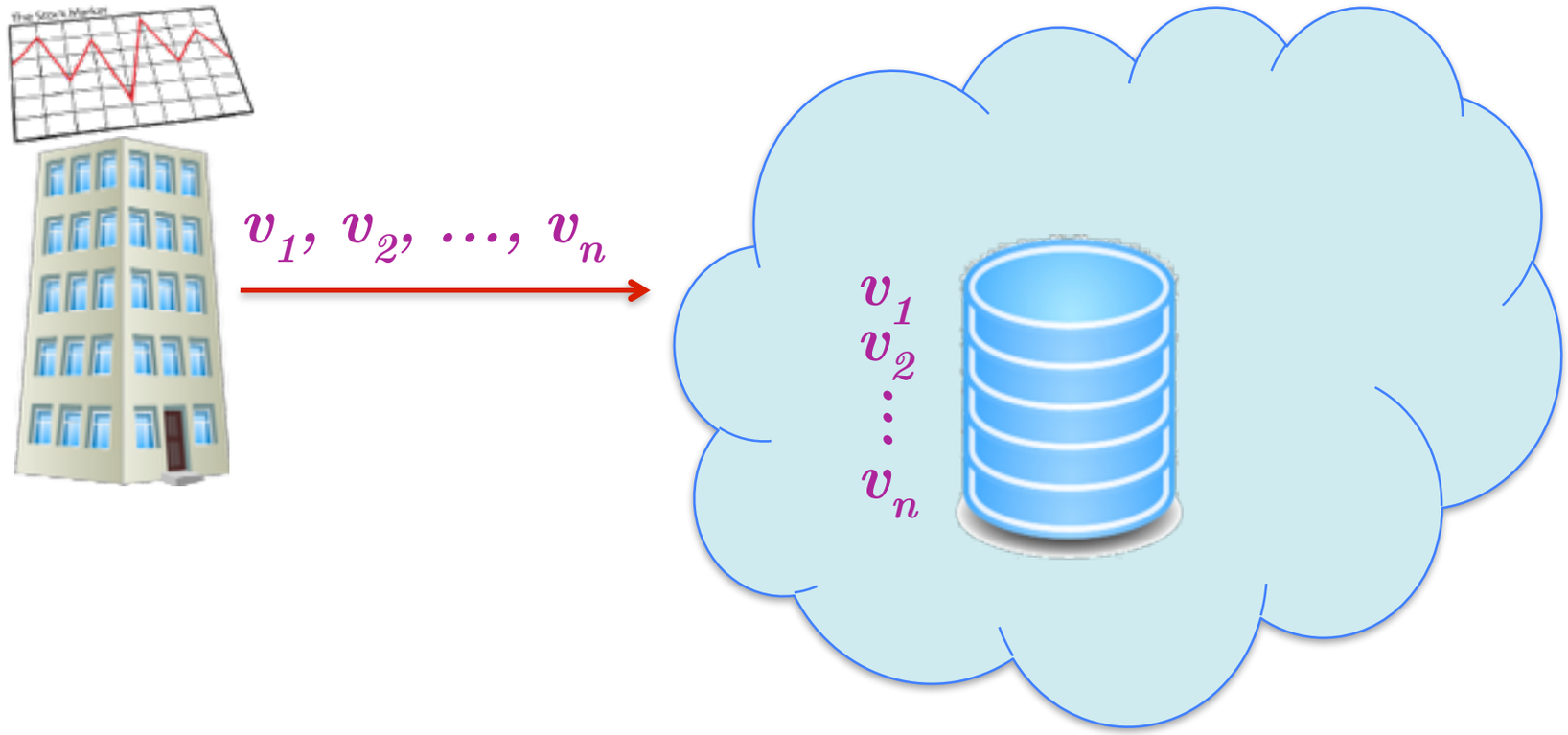
Luca Nizzardo*
Università di Milano-Bicocca
Italy
*work done while visiting CUNY

PKC'14 – Buenos Aires, March 28, 2014

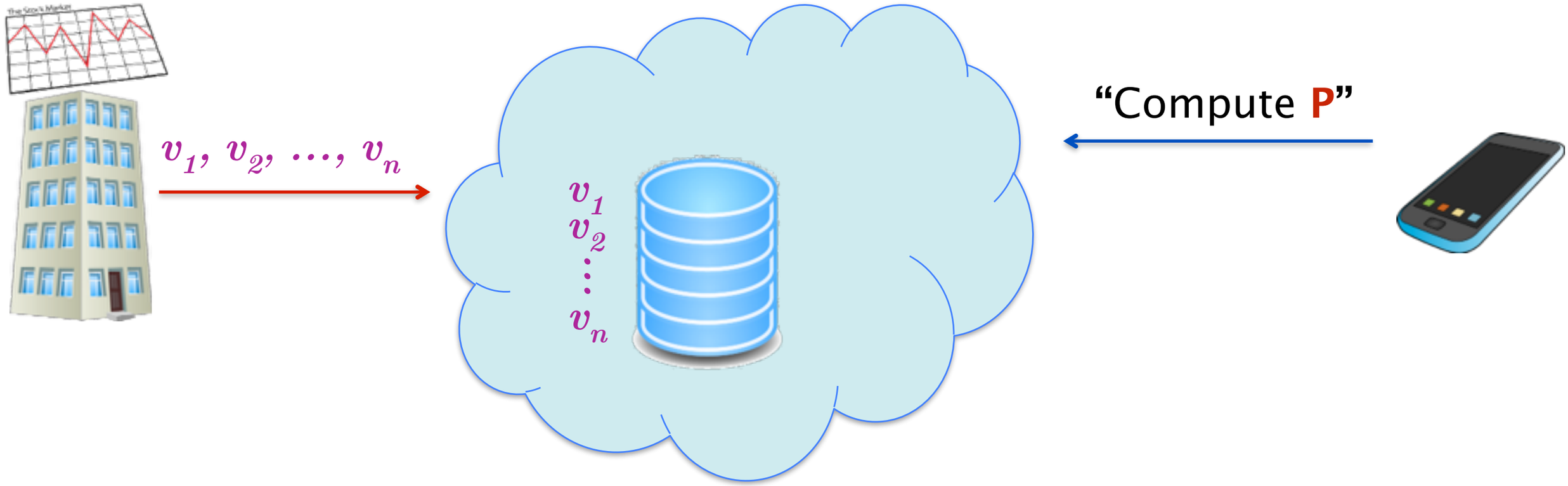
Outline

- Motivation
- Homomorphic MACs
  - Definition
  - Previous work
- Our results
- Summary & Open problems

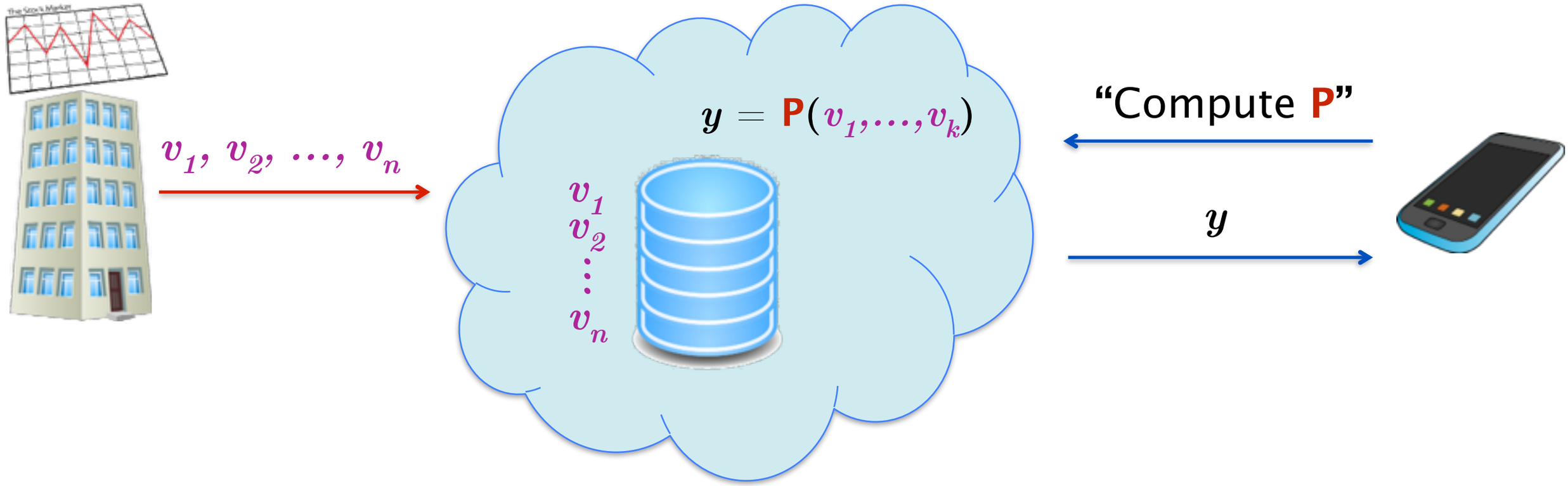
Delegating Computations on Outsourced Data



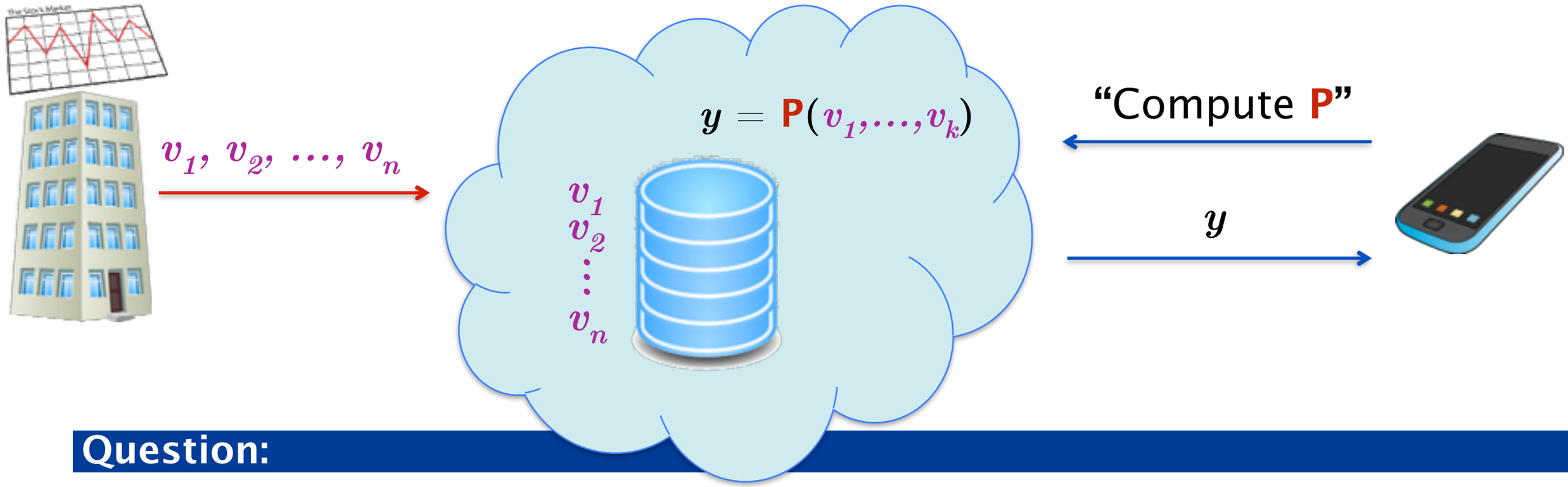
Delegating Computations on Outsourced Data



Delegating Computations on Outsourced Data



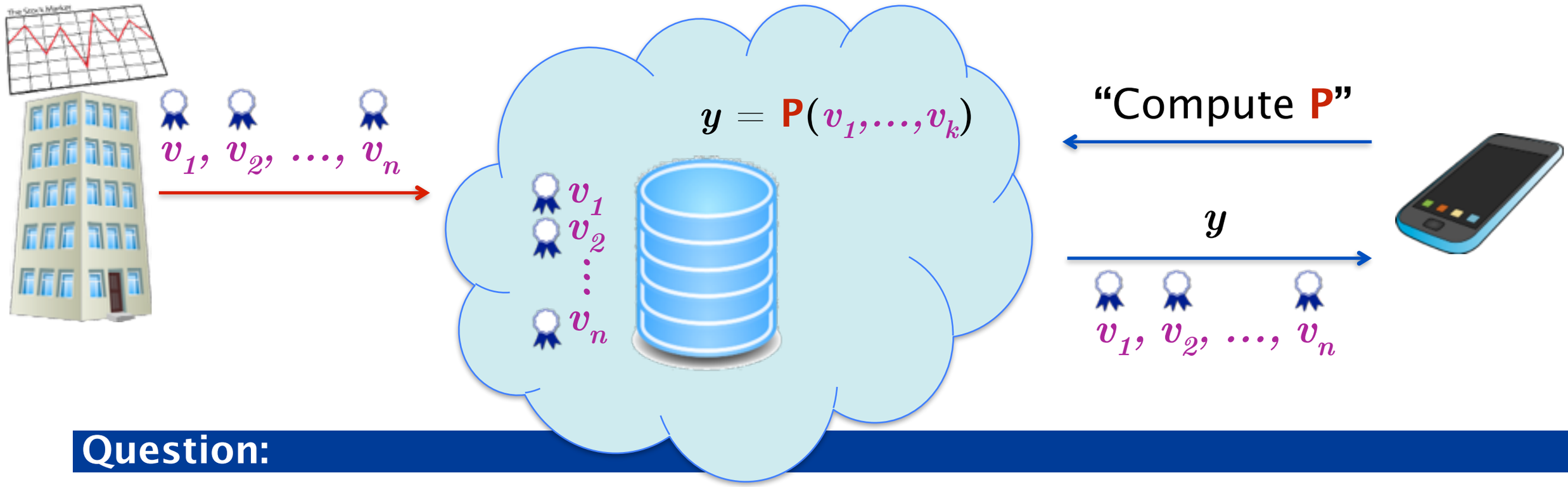
Delegating Computations on Outsourced Data



Question:

- **How can the client be sure that P is executed on the company's data?**

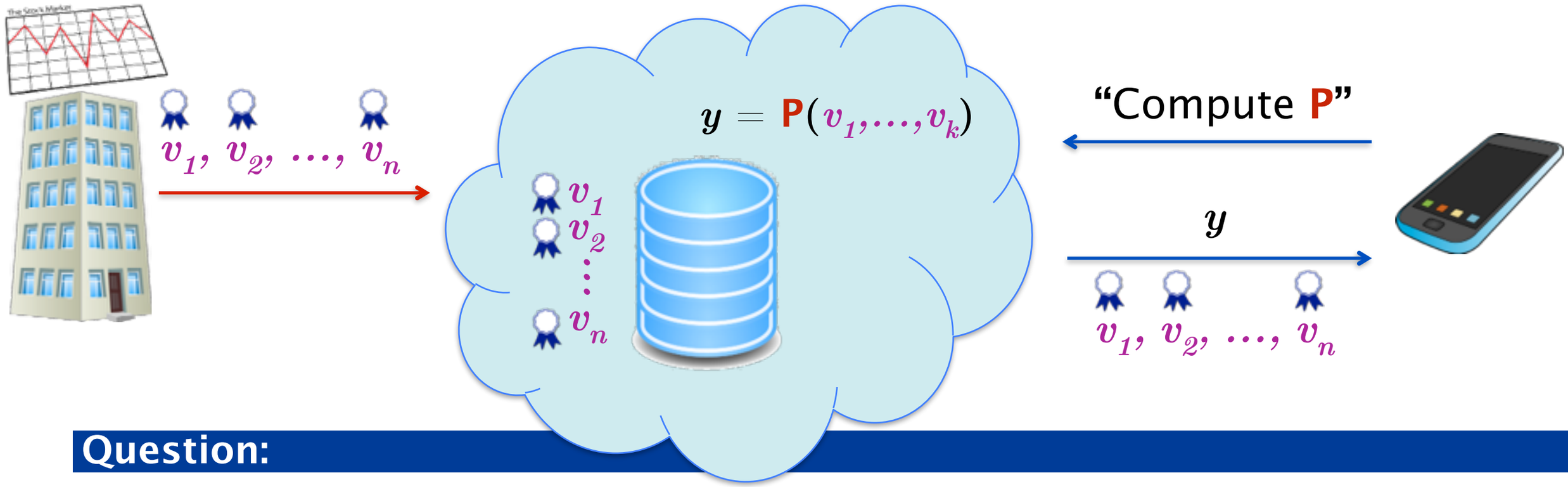
Delegating Computations on Outsourced Data



Question:

- **How can the client be sure that P is executed on the company's data?**
- **Trivial solution:** the cloud sends all the authenticated inputs.

Delegating Computations on Outsourced Data

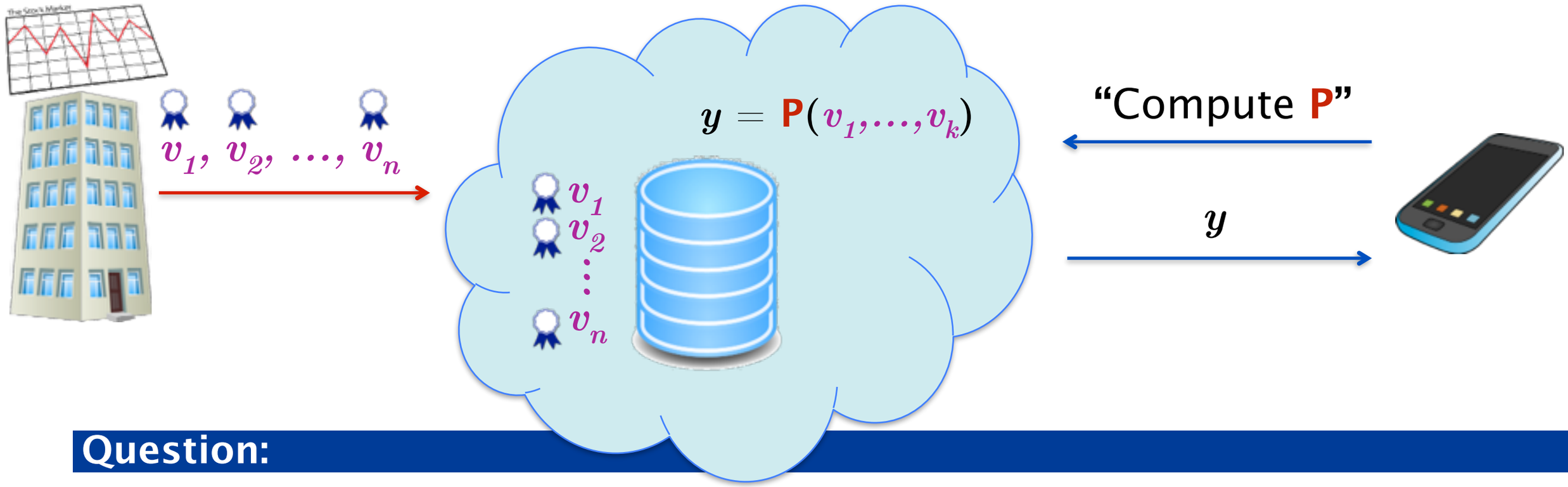


Question:

- **How can the client be sure that P is executed on the company's data?**
- **Trivial solution:** the cloud sends all the authenticated inputs.

TOO INEFFICIENT

Delegating Computations on Outsourced Data



Question:

- **How can the client be sure that P is executed on the company's data?**
- **Trivial solution:** the cloud sends all the authenticated inputs.

TOO INEFFICIENT

Main Goals

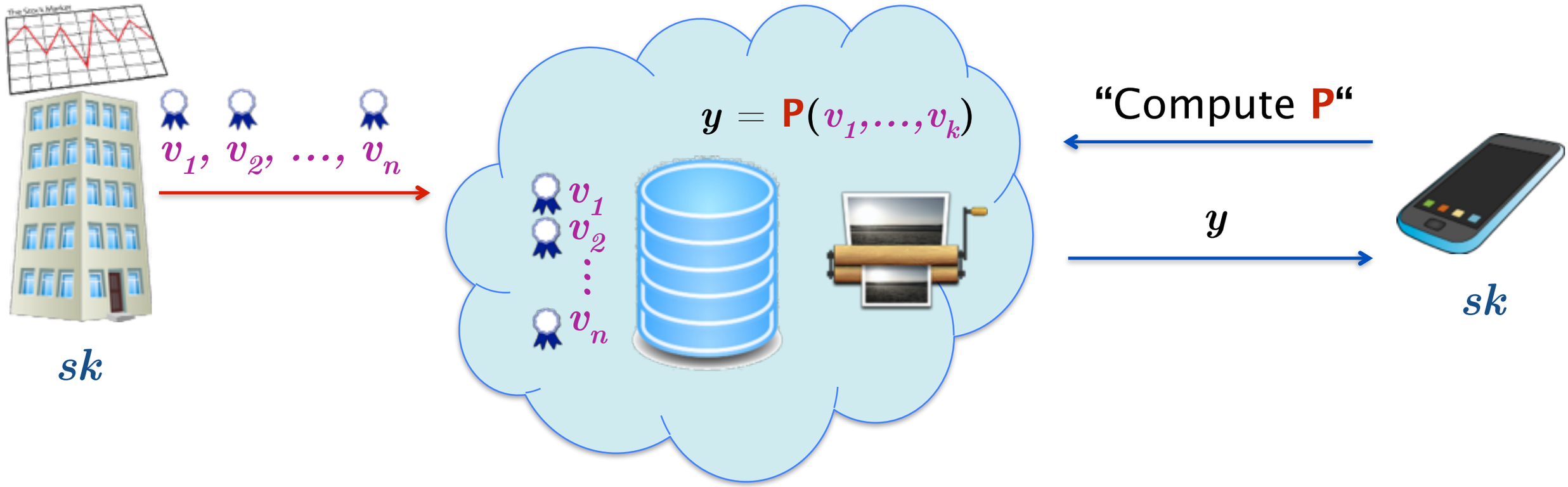
- **Integrity**

  Untrusted cloud must not be able to send incorrect y

- **Efficiency**

  Client's communication and storage must be minimized

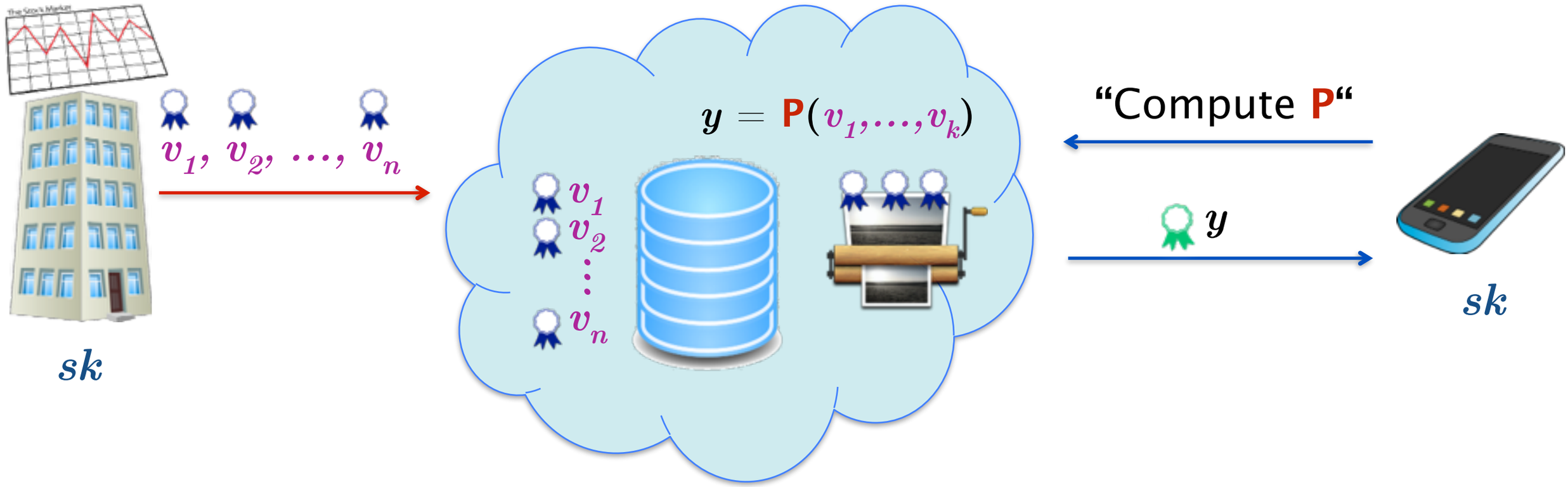
An approach to solve the problem: Homomorphic Message Authenticators [GW13]



Main Goals

- **Integrity**
  Untrusted cloud must not be able to send incorrect y
- **Efficiency**
  Client's communication and storage must be minimized

An approach to solve the problem: Homomorphic Message Authenticators [GW13]

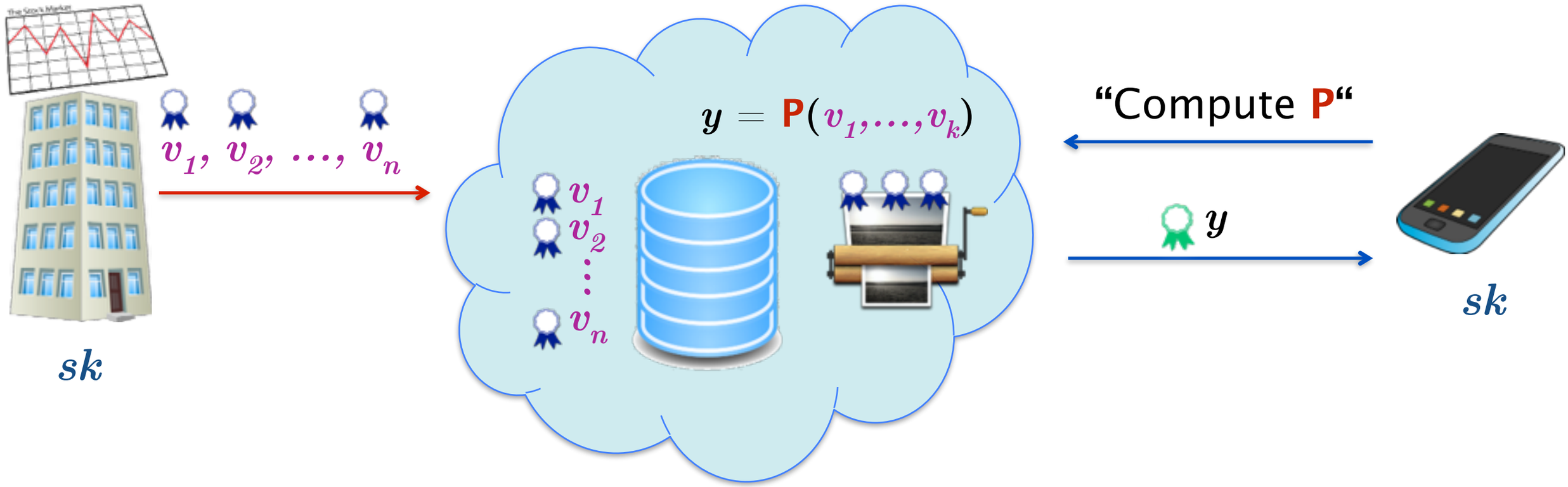


proves that "y is the output of **P** on authenticated data"

Main Goals

- **Integrity**
  Untrusted cloud must not be able to send incorrect y
- **Efficiency**
  Client's communication and storage must be minimized

An approach to solve the problem: Homomorphic Message Authenticators [GW13]



proves that "y is the output of **P** on authenticated data"

Main Goals

- **Integrity**
  - ✓ Untrusted cloud must not be able to send incorrect y
  - Cloud cannot forge MACs.
- **Efficiency**
  - ✓ Client's communication and storage must be minimized
  - |🎗| << size of k input values.

Homomorphic MACs & Labeled Programs

[GW13]

Homomorphic MACs & Labeled Programs

[GW13]

- **KeyGen(λ)→(*sk*,*ek*)** // private key *sk*, public evaluation key *ek*

Homomorphic MACs & Labeled Programs

[GW13]

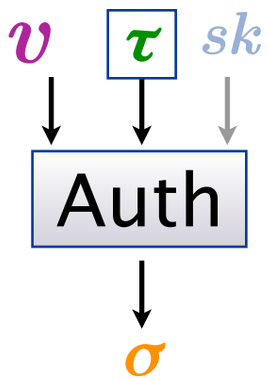
- **KeyGen(λ)→(*sk*,*ek*)** // private key *sk*, public evaluation key *ek*
- **Auth(*sk*,*v*,*τ*)→*σ*** which authenticates **value *v*** w.r.t. label *τ*
  - Idea of labels: uniquely "remember" the outsourced data

    $ 665.41 ~ "Jan, 3rd, 2012, Google stock price"

    $ 668.28 ~ "Jan, 4th, 2012, Google stock price"

    $ 659.01 ~ "Jan, 5th, 2012, Google stock price"

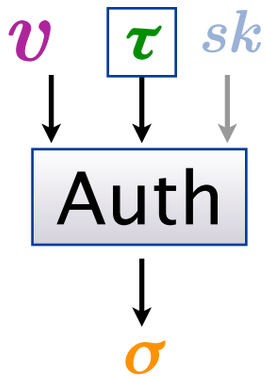
    … …



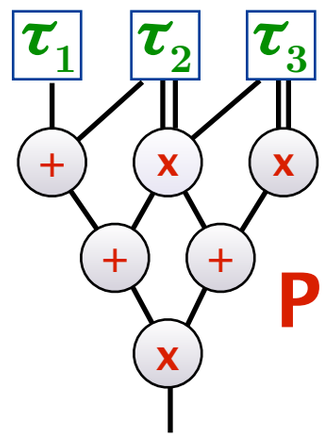
Homomorphic MACs & Labeled Programs [GW13]

- **KeyGen(λ)→(*sk*,*ek*)** // private key *sk*, public evaluation key *ek*
- **Auth(*sk*,*v*,*τ*)→*σ*** which authenticates **value *v*** w.r.t. label *τ*
  - Idea of labels: uniquely "remember" the outsourced data

    $665.41 ~ "Jan, 3rd, 2012, Google stock price"
    $668.28 ~ "Jan, 4th, 2012, Google stock price"
    $659.01 ~ "Jan, 5th, 2012, Google stock price"
    … …



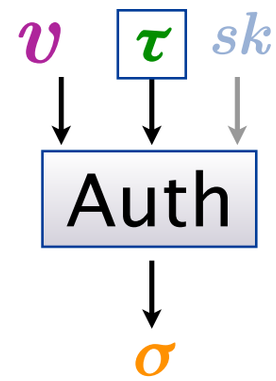
- **Eval(*ek*,P,$\sigma_1,\ldots,\sigma_n$)→*σ*** new tag authenticating "output of **labeled program P**"
  - A **labeled program P** is a circuit **f** with a **label *τ*** on each input wire
    - e.g., **P** computes the yearly average stock price for some days — each day labeled by some τ_i



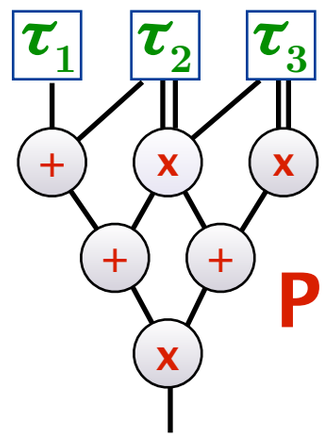
Homomorphic MACs & Labeled Programs [GW13]

- **KeyGen(λ)→(*sk*,*ek*)** // private key *sk*, public evaluation key *ek*
- **Auth(*sk*,*v*,*τ*)→*σ*** which authenticates **value *v*** w.r.t. label *τ*
  - Idea of labels: uniquely "remember" the outsourced data

    $665.41 ~ "Jan, 3rd, 2012, Google stock price"
    $668.28 ~ "Jan, 4th, 2012, Google stock price"
    $659.01 ~ "Jan, 5th, 2012, Google stock price"
    … …



- **Eval(*ek*,P,$\sigma_1,\ldots,\sigma_n$)→*σ*** new tag authenticating "output of **labeled program P**"
  - A **labeled program P** is a circuit **f** with a **label *τ*** on each input wire
    - e.g., **P** computes the yearly average stock price for some days — each day labeled by some τ_i

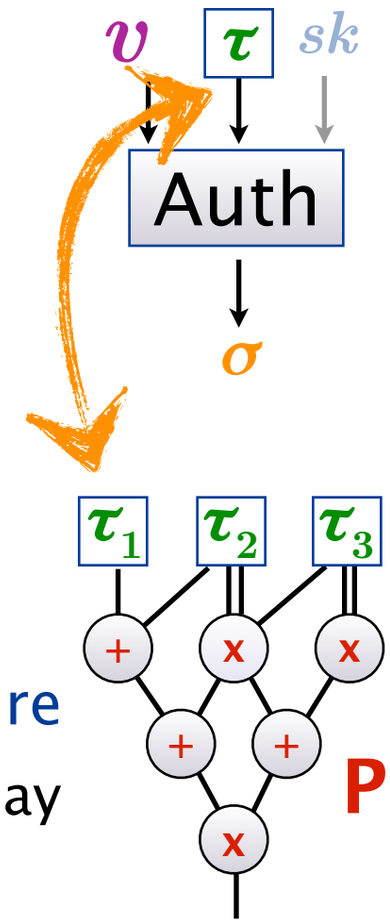


- **Ver(*sk*, P, *v*, *σ*)** checks whether *v* is output of **P**=(**f**,$\tau_1, \ldots, \tau_n$) on values authenticated with labels $\tau_1,\ldots,\tau_n$

Homomorphic MACs & Labeled Programs [GW13]

- **KeyGen(λ)→(*sk*,*ek*)** // private key *sk*, public evaluation key *ek*
- **Auth(*sk*,*v*,*τ*)→*σ*** which authenticates **value *v*** w.r.t. label *τ*
  - Idea of labels: uniquely "remember" the outsourced data

    $665.41 ~ "Jan, 3rd, 2012, Google stock price"
    $668.28 ~ "Jan, 4th, 2012, Google stock price"
    $659.01 ~ "Jan, 5th, 2012, Google stock price"
    … …



- **Eval(*ek*,P,$\sigma_1,\ldots,\sigma_n$)→*σ*** new tag authenticating "output of **labeled program P**"
  - A **labeled program P** is a circuit **f** with a **label *τ*** on each input wire
    - e.g., **P** computes the yearly average stock price for some days — each day labeled by some τ_i

- **Ver**(*sk*, P, *v*, *σ*) checks whether *v* is output of P=(f,$\tau_1, \ldots, \tau_n$) on values authenticated with labels $\tau_1,\ldots,\tau_n$

Properties of Homomorphic MACs

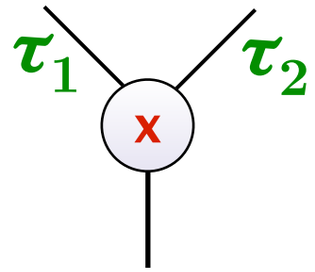
- **Security:** …in 2 slides
- **Succinctness:** size of tags (returned by **Eval**) does not depend on the number of inputs of the computation
- **Composition:** authenticated outputs can be further used as inputs to other circuits

Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output

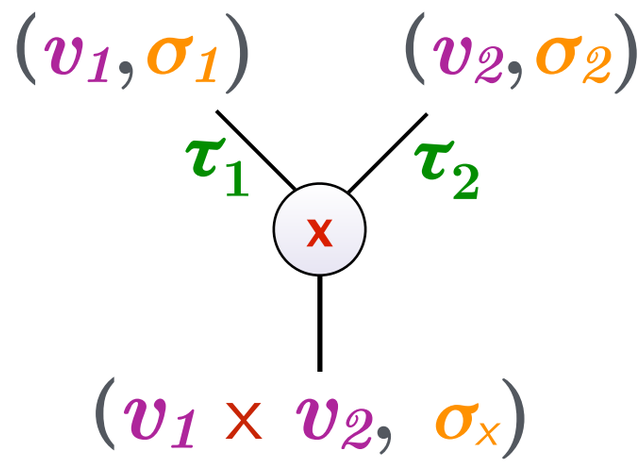
Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output



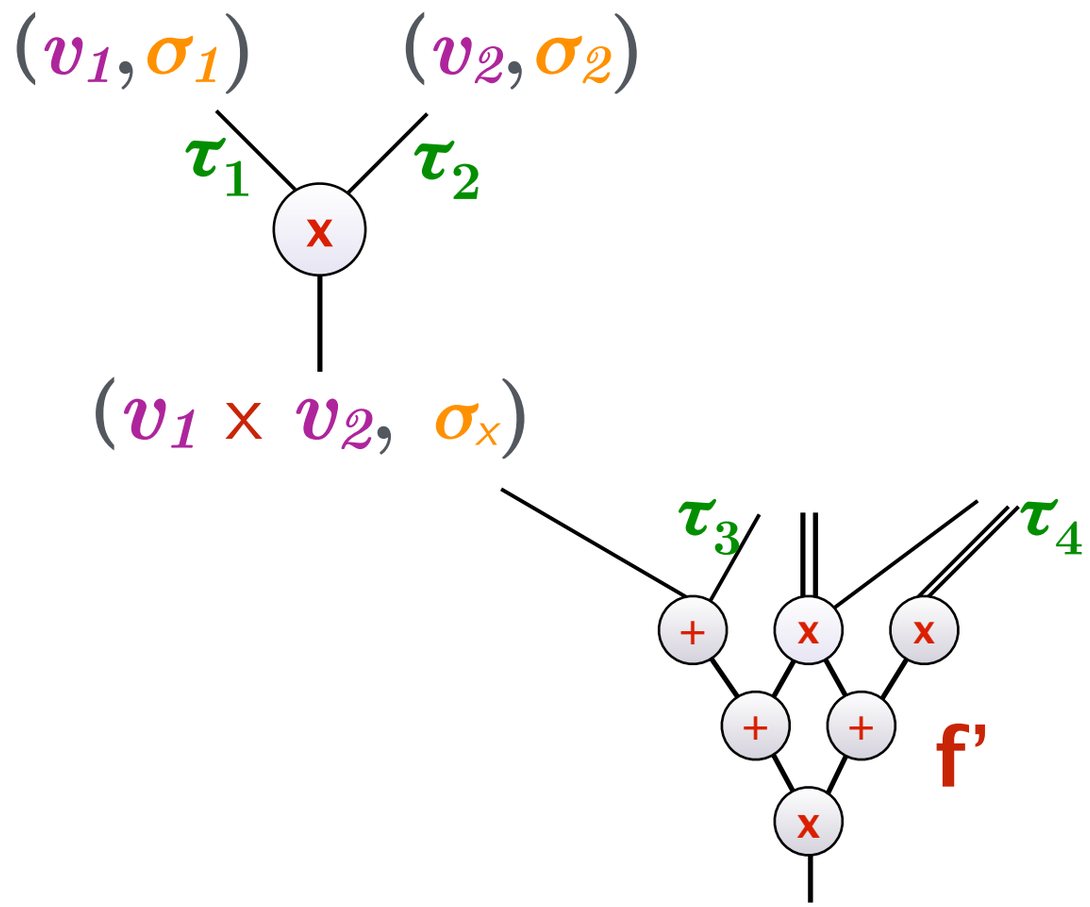
Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output



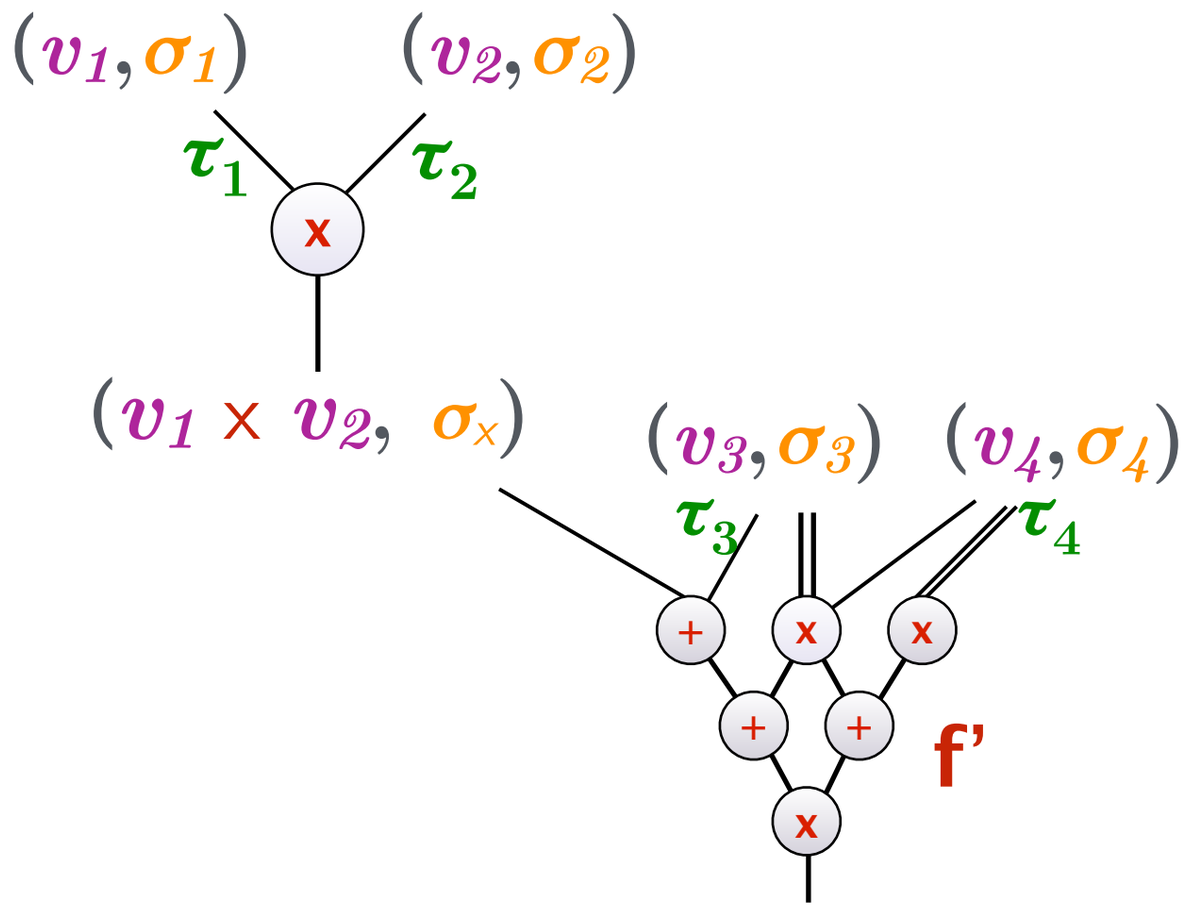
Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output



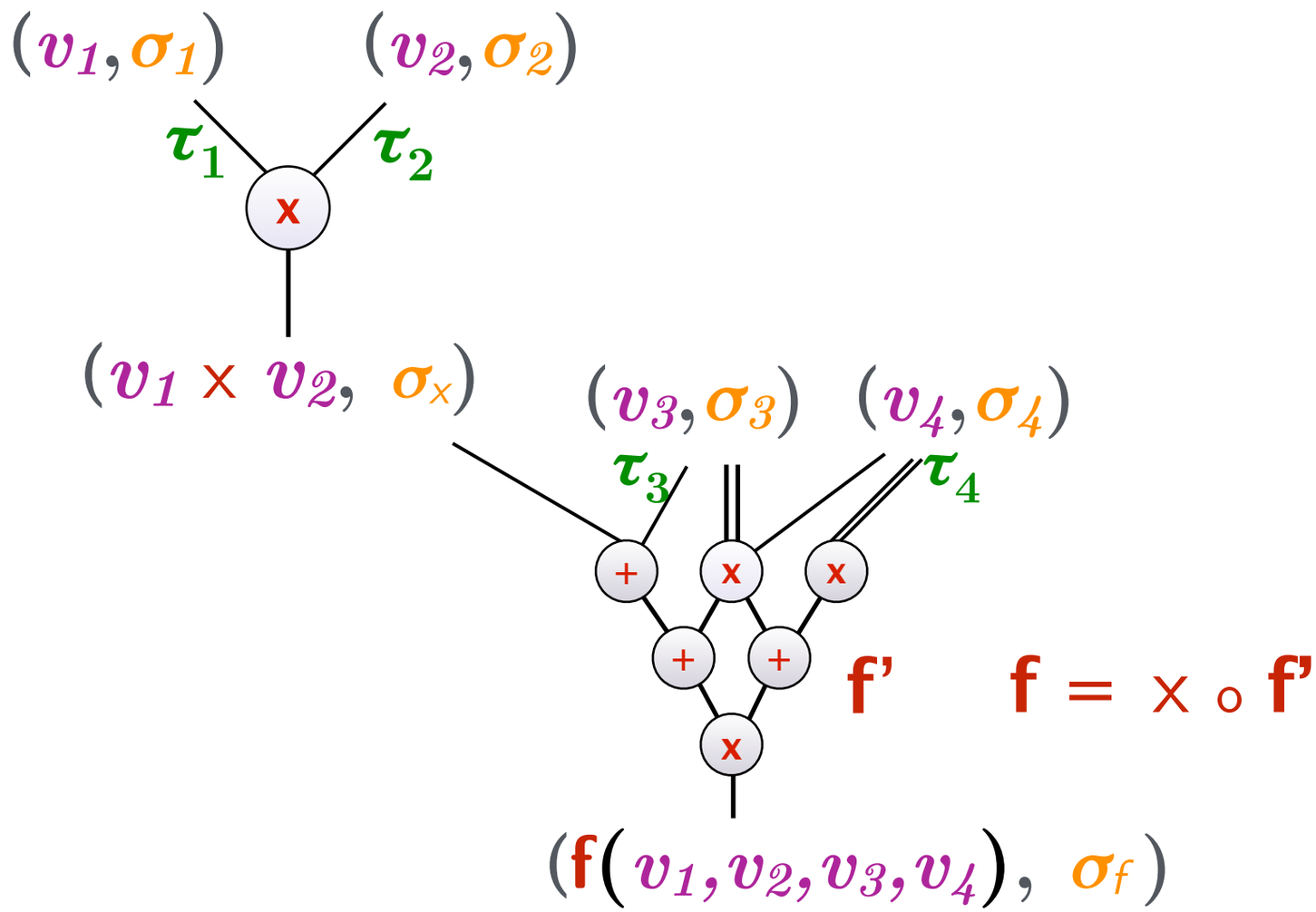
Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output



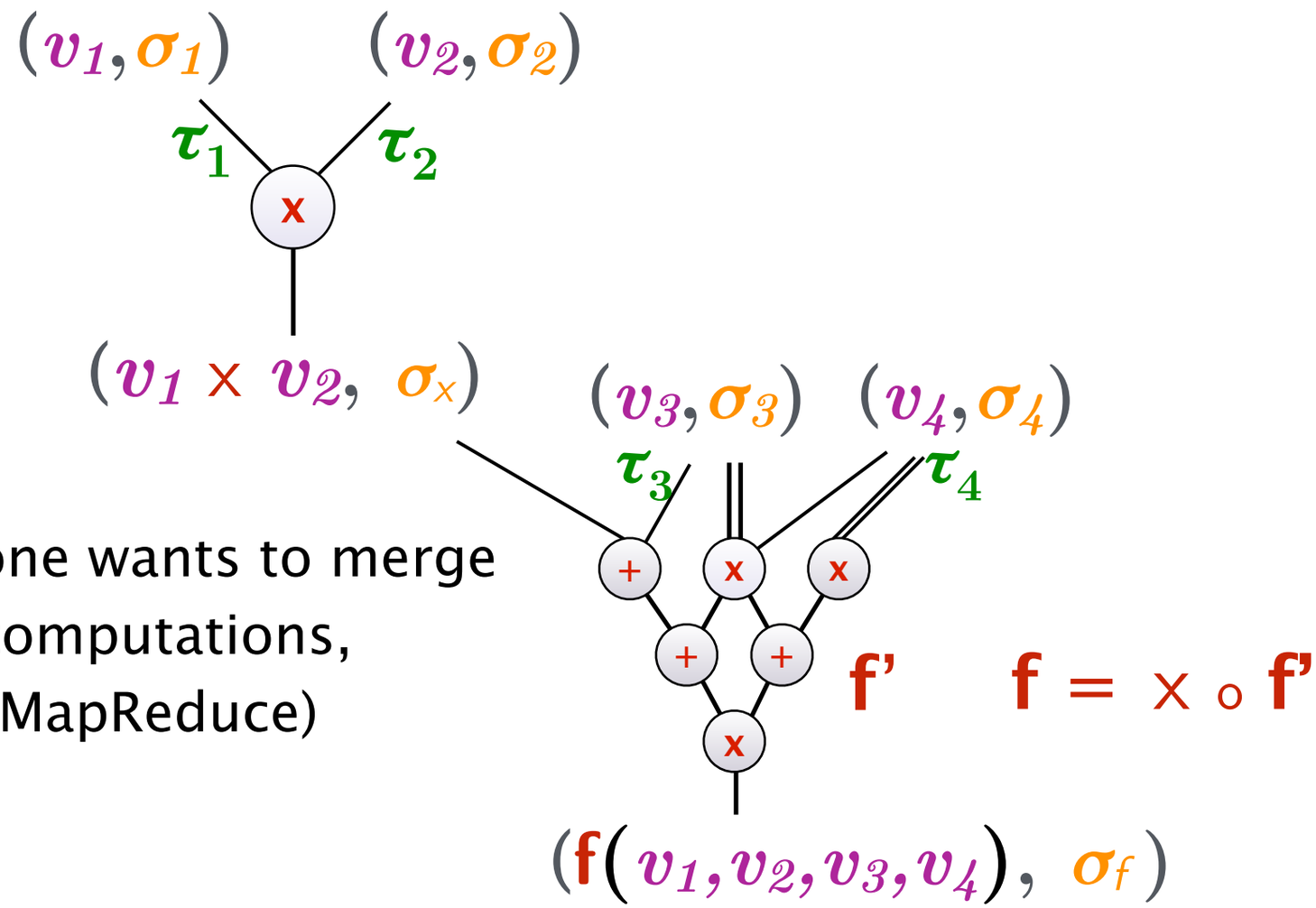
Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output



Composition

- At gate level: for every pair of authenticated inputs, obtain an authenticated output



Very useful property if one wants to merge partially authenticated computations, e.g., for parallelization (MapReduce)

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

ek

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

τ_i , v_i

σ_i=**Auth**(sk, τ_i, v_i)

ek

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

τ_i , v_i

σ_i=**Auth**(sk, τ_i, v_i)

P, v, σ

b=**Ver**(sk, **P**, v, σ)

ek

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

τ_i , v_i

σ_i=**Auth**(sk, τ_i, v_i)

P, v, σ

b=**Ver**(sk,**P**, v, σ)

Each τ_i can be queried only once

ek

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

τ_i , v_i

σ_i=**Auth**(sk, τ_i, v_i)

P, v, σ

b=**Ver**(sk,**P**,v,σ)

Each τ_i can be queried only once

ek

- Adversary wins if it makes a verification query (**P**,v^*,σ^*) such that, for **P**=(**f**,τ_1, …, τ_n): **Ver**(sk,**P**,v^*,σ^*)=accept and

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

τ_i , v_i

σ_i=**Auth**(sk, τ_i, v_i)

P, v, σ

b=**Ver**(sk,**P**,v,σ)

Each τ_i can be queried only once

ek

- Adversary wins if it makes a verification query (**P**,v^*,σ^*) such that, for **P**=(**f**,τ_1, …, τ_n): **Ver**(sk,**P**,v^*,σ^*)=accept and
  - **Type-1:** $\exists \tau_j$ that has never been queried, and τ_j "does contribute" to **f**

Security

Unforgeability against chosen-message attacks

Basic idea: nobody, without sk, can create a "valid" MAC

sk

τ_i , v_i

σ_i=**Auth**(sk, τ_i, v_i)

P, v, σ

b=**Ver**(sk,**P**,v,σ)

Each τ_i can be queried only once

ek

- Adversary wins if it makes a verification query (**P**,v^*,σ^*) such that, for **P**=(**f**,τ_1, …, τ_n): **Ver**(sk,**P**,v^*,σ^*)=accept and
  - **Type-1:** ∃τ_j that has never been queried, and τ_j "does contribute" to **f**
  - **Type-2:** all labels have been queried and v^*≠**f**(v_1,…,v_n)

Realizations: Previous Work

Realizations: Previous Work

- **Homomorphic Signatures** [JMSW02] (more flexible – public verification)
  - Many realizations for linear functions [BFKW09, GKKR10, CFW11, AL11, CFW12, Freeman12, ALP13, …]
  - Beyond linear: only one scheme [BF11] for constant-degree polynomials

Realizations: Previous Work

- **Homomorphic Signatures** [JMSW02] (more flexible – public verification)
  - Many realizations for linear functions [BFKW09, GKKR10, CFW11, AL11, CFW12, Freeman12, ALP13, …]
  - Beyond linear: only one scheme [BF11] for constant-degree polynomials
- **Homomorphic MACs** (beyond linear)**:**

Realizations: Previous Work

- **Homomorphic Signatures** [JMSW02] (more flexible – public verification)
  - Many realizations for linear functions [BFKW09, GKKR10, CFW11, AL11, CFW12, Freeman12, ALP13, …]
  - Beyond linear: only one scheme [BF11] for constant-degree polynomials
- **Homomorphic MACs** (beyond linear)**:**

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no verif. queries	Arbitrary	O(1)	✔

Realizations: Previous Work

- **Homomorphic Signatures** [JMSW02] (more flexible – public verification)
  - Many realizations for linear functions [BFKW09, GKKR10, CFW11, AL11, CFW12, Freeman12, ALP13, …]
  - Beyond linear: only one scheme [BF11] for constant-degree polynomials
- **Homomorphic MACs** (beyond linear)**:**

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no verif. queries	Arbitrary	O(1)	✔
[CF13] (1)	OWF	full	degree-d arithmetic circuits, d=O(1)	O(d)	✔

Realizations: Previous Work

- **Homomorphic Signatures** [JMSW02] (more flexible – public verification)
  - Many realizations for linear functions [BFKW09, GKKR10, CFW11, AL11, CFW12, Freeman12, ALP13, …]
  - Beyond linear: only one scheme [BF11] for constant-degree polynomials
- **Homomorphic MACs** (beyond linear)**:**

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no verif. queries	Arbitrary	O(1)	✔
[CF13] (1)	OWF	full	degree-d arithmetic circuits, d=O(1)	O(d)	✔
[CF13] (2)	d-DHI	full	degree-D arithmetic circuits for D=poly(k)	O(1)	✘

Our Results

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no ver. queries	Arbitrary	O(1)	✔
[CF13] (1)	OWF	full	degree-d arithmetic circuits, d=O(1)	O(d)	✔
[CF13] (2)	d-DHI	full	degree-D arithmetic circuits for D=poly(k)	O(1)	✘
This work	Encoding w/ limited malleability	full	degree-D arithmetic circuits for D=poly(k)	O(1)	✘
This work	(D,k)-MDHI on multilinear maps	full	degree-(D+k) arithmetic circuits	$O(k^2)$	✔ (k)

Our Results

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no ver. queries	Arbitrary	O(1)	✔
[CF13] (1)	OWF		degree-d arithmetic		✔
[CF13] (2)	d-DHI				✘
This work	Encoding w/ limited malleability	full	degree-D arithmetic circuits for D=poly(k)	O(1)	✘
This work	(D,k)-MDHI on multilinear maps	full	degree-(D+k) arithmetic circuits	$O(k^2)$	✔ (k)

Basic idea: additively homomorphic but not multiplicative homomorphic (similar to [BCIOP13]). Possible instantiations: Paillier, BV11.

Our Results

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no ver. queries	Arbitrary	O(1)	✔
[CF13] (1)	OWF	full	degree-d arithmetic circuits, d=O(1)	O(d)	✔
[CF13] (2)	d-DHI		degree-D arithmetic		✘
This work	Encoding w/ limited malleability		D=poly(k)		✘
This work	(D,k)-MDHI on multilinear maps	full	degree-(D+k) arithmetic circuits	$O(k^2)$	✔ (k)

We use graded k-linear maps [GGH13, CLT13] and support composition circuits of bounded degree k.

Our Results

	Assumption	Security	Computations	Size of tags	Comp.
[GW13]	FHE	no ver. queries	Arbitrary	O(1)	✔
[CF13] (1)	OWF	full	degree-d arithmetic circuits, d=O(1)	O(d)	✔
[CF13] (2)	d-DHI		degree-D arithmetic		✘
This work	Encoding w/ limited malleability		D=poly(k)		✘
This work	(D,k)-MDHI on multilinear maps	full	degree-(D+k) arithmetic circuits	$O(k^2)$	✔ (k)

We use graded k-linear maps [GGH13, CLT13] and support composition circuits of bounded degree k.

This Talk

Graded k-Linear maps

- **Gen**(1^λ, k) generates k groups of prime order p

$$G_1, G_2, \ldots, G_k$$

- with a collection of bilinear maps

$$e_{ij}: G_i \times G_j \to G_{i+j} : \quad e_{ij}(g_i^{\,a}, g_j^{\,b}) = g_{i+j}^{\,ab}$$

- **Notation:** $g \in G_1$, $g_i = e(g, \ldots, g)$ i times
- "Approximate" realizations via graded encodings [GGH13, CLT13]

Our Homomorphic MAC

Our Homomorphic MAC

- **KeyGen(1^λ, D, k):**
  - Generate leveled k-linear groups of prime order p, e_{ij}: $G_i \times G_j \rightarrow G_{i+j}$
  - Take random generator g in G_1, sample $x, a \leftarrow Z_p$,
  - Compute $g^{x\hat{}i}$, $g^{ax\hat{}i}$, for i=1…D
  - Sample a seed K of a PRF F_K: $\{0,1\}^* \rightarrow Z_p$
  - sk=(K, g, x, a), ek=(g^a, $\{g^{x\hat{}i}, g^{ax\hat{}i}\}_i$)

Our Homomorphic MAC

- **KeyGen(1^λ, D, k):**
  - Generate leveled k-linear groups of prime order p, e_{ij}: $G_i \times G_j \to G_{i+j}$
  - Take random generator g in G_1, sample $x, a \leftarrow Z_p$,
  - Compute g^{x^i}, g^{ax^i}, for i=1…D
  - Sample a seed K of a PRF F_K: $\{0,1\}^* \to Z_p$
  - sk=(K, g, x, a), ek=(g^a, $\{g^{x^i}, g^{ax^i}\}_i$)
- **Auth**(sk, v, τ): the tag is a degree-1 polynomial $y(X) \in Z_p[X]$ s.t.

$$y(0)=v \quad \text{and} \quad y(x)=r_\tau=F_K(\tau)$$

Our Homomorphic MAC

- **KeyGen(1^λ, D, k):**
  - Generate leveled k-linear groups of prime order p, e_{ij}: $G_i \times G_j \to G_{i+j}$
  - Take random generator g in G_1, sample $x, a \leftarrow Z_p$,
  - Compute g^{x^i}, g^{ax^i}, for i=1…D
  - Sample a seed K of a PRF F_K: $\{0,1\}^* \to Z_p$
  - sk=(K, g, x, a), ek=(g^a, $\{g^{x^i}, g^{ax^i}\}_i$)
- **Auth**(sk, v, τ): the tag is a degree-1 polynomial $y(X) \in Z_p[X]$ s.t.

$$y(0)=v \quad \text{and} \quad y(x)=r_\tau=F_K(\tau)$$

- **Eval**(ek,f): compute $y(X) \leftarrow f(y_1(X), \ldots, y_n(X))$ over $Z_p[X]$, $|y(X)| \le D$

Our Homomorphic MAC

- **KeyGen(1^λ, D, k):**
  - Generate leveled k-linear groups of prime order p, e_{ij}: $G_i \times G_j \to G_{i+j}$
  - Take random generator g in G_1, sample $x, a \leftarrow Z_p$,
  - Compute g^{x^i}, g^{ax^i}, for i=1…D
  - Sample a seed K of a PRF F_K: $\{0,1\}^* \to Z_p$
  - sk=(K, g, x, a), ek=(g^a, {g^{x^i},g^{ax^i}}$_i$)
- **Auth**(sk, v, τ): the tag is a degree-1 polynomial $y(X) \in Z_p[X]$ s.t.

$$y(0)=v \quad \text{and} \quad y(x)=r_\tau=F_K(\tau)$$

- **Eval**(ek,f): compute $y(X) \leftarrow f(y_1(X), \ldots, y_n(X))$ over $Z_p[X]$, $|y(X)| \le D$
- **Compress**(ek, $y(X)$): $\Lambda \leftarrow \prod^d_{i=1} (g^{x^i})^{yi} = g^{y(x)-y(0)}$
  - Similarly, compute $\Gamma \leftarrow g^{a[y(x)-y(0)]} = \Lambda^a$. Output σ=($y(0)$, Λ, Γ)

Our Homomorphic MAC cont'd

Our Homomorphic MAC cont'd

- **CompositionEval(**ek, ϕ, $\sigma_1=(v_1, \Lambda_1, \Gamma_1)$, $\sigma_2=(v_2, \Lambda_2, \Gamma_2)$ **)**$\rightarrow \sigma=(v, \Lambda, \Gamma)$

(simplified description for ϕ single gate and elements in G_1)

Our Homomorphic MAC cont'd

- **CompositionEval(**ek, ϕ, $\sigma_1=(v_1, \Lambda_1, \Gamma_1)$, $\sigma_2=(v_2, \Lambda_2, \Gamma_2)$ **)**$\rightarrow \sigma=(v, \Lambda, \Gamma)$

(simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1 \Lambda_2$, $\Gamma=\Gamma_1 \Gamma_2$

Our Homomorphic MAC cont'd

- **CompositionEval(**ek, ϕ, $\sigma_1=(v_1,\Lambda_1,\ \Gamma_1)$, $\sigma_2=(v_2,\ \Lambda_2,\ \Gamma_2)$ **)**$\rightarrow\sigma=(v,\ \Lambda,\ \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2,\quad \Lambda=\Lambda_1\ \Lambda_2,\quad \Gamma=\Gamma_1\ \Gamma_2$
  - **Multiplication:** $v=v_1v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\ e(\Lambda_1, g^a)^{v_2}\ e(g^a,\Lambda_2)^{v_1} = g_2^{\ a[y(x)\ -\ v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\ \ e(\Gamma_1, g^a)^{v_2}\ e(g^a,\Gamma_2)^{v_1} = g_2^{\ a^2[y(x)\ -\ v]}$$

Our Homomorphic MAC cont'd

- **CompositionEval(ek, ϕ, $\sigma_1=(v_1,\Lambda_1,\Gamma_1)$, $\sigma_2=(v_2,\Lambda_2,\Gamma_2)$)$\rightarrow\sigma=(v,\Lambda,\Gamma)$**

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1\,\Lambda_2$, $\Gamma=\Gamma_1\,\Gamma_2$
  - **Multiplication:** $v=v_1v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\, e(\Lambda_1, g^a)^{v_2}\, e(g^a,\Lambda_2)^{v_1} = g_2^{\,a[y(x) - v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\; e(\Gamma_1, g^a)^{v_2}\, e(g^a,\Gamma_2)^{v_1} = g_2^{\,a^2[y(x) - v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots,\Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

Our Homomorphic MAC cont'd

- **CompositionEval**(ek, ϕ, $\sigma_1=(v_1, \Lambda_1, \Gamma_1)$, $\sigma_2=(v_2, \Lambda_2, \Gamma_2)$)$\rightarrow \sigma=(v, \Lambda, \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1 \Lambda_2$, $\Gamma=\Gamma_1 \Gamma_2$
  - **Multiplication:** $v=v_1 v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\, e(\Lambda_1, g^a)^{v_2}\, e(g^a,\Lambda_2)^{v_1} = g_2^{a[y(x) - v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\, e(\Gamma_1, g^a)^{v_2}\, e(g^a,\Gamma_2)^{v_1} = g_2^{a^2[y(x) - v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots, \Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

- **Ver**(sk, P, v, σ)→0/1 Let P=(f,$\tau_1, \ldots, \tau_n$) and $\sigma=(v, \Lambda, \Gamma)$

Our Homomorphic MAC cont'd

- **CompositionEval**(ek, ϕ, $\sigma_1=(v_1,\Lambda_1,\ \Gamma_1)$, $\sigma_2=(v_2,\ \Lambda_2,\ \Gamma_2)$)$\rightarrow\sigma=(v,\ \Lambda,\ \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1\ \Lambda_2$, $\Gamma=\Gamma_1\ \Gamma_2$
  - **Multiplication:** $v=v_1v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\ e(\Lambda_1, g^a)^{v_2}\ e(g^a,\Lambda_2)^{v_1} = g_2^{\ a[y(x)\ -\ v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\ \ e(\Gamma_1, g^a)^{v_2}\ e(g^a,\Gamma_2)^{v_1} = g_2^{\ a^2[y(x)\ -\ v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots,\Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

- **Ver**(sk, P, v, σ)→0/1 Let P=(f,$\tau_1,\ \ldots,\ \tau_n$) and $\sigma=(v,\ \Lambda,\ \Gamma\ )$
  - Derive $r_i\leftarrow F_K(\tau_i)$ i=1…n and compute $r\leftarrow f(r_1,\ \ldots,\ r_n)$

Our Homomorphic MAC cont'd

- **CompositionEval**(ek, ϕ, $\sigma_1=(v_1,\Lambda_1,\ \Gamma_1)$, $\sigma_2=(v_2,\ \Lambda_2,\ \Gamma_2)$)$\rightarrow\sigma=(v,\ \Lambda,\ \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1\ \Lambda_2$, $\Gamma=\Gamma_1\ \Gamma_2$
  - **Multiplication:** $v=v_1v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\ e(\Lambda_1, g^a)^{v_2}\ e(g^a,\Lambda_2)^{v_1} = g_2^{\ a[y(x)\ -\ v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\ \ e(\Gamma_1, g^a)^{v_2}\ e(g^a,\Gamma_2)^{v_1} = g_2^{\ a^2[y(x)\ -\ v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots,\Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

- **Ver**(sk, **P**, v, σ)→0/1 Let **P**=(**f**,$\tau_1,\ \ldots,\ \tau_n$) and $\sigma=(v,\ \Lambda,\ \Gamma\ )$
  - Derive $r_i\leftarrow F_K(\tau_i)$ i=1…n and compute $r\leftarrow f(r_1,\ \ldots,\ r_n)$
  - Verify the invariant $\Lambda= g_d^{\ a\hat{}(d-1)[r\ -\ v]}$

Our Homomorphic MAC cont'd

- **CompositionEval**(ek, ϕ, $\sigma_1=(v_1, \Lambda_1, \Gamma_1)$, $\sigma_2=(v_2, \Lambda_2, \Gamma_2)$)$\rightarrow \sigma=(v, \Lambda, \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1 \Lambda_2$, $\Gamma=\Gamma_1 \Gamma_2$
  - **Multiplication:** $v=v_1 v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\, e(\Lambda_1, g^a)^{v_2}\, e(g^a,\Lambda_2)^{v_1} = g_2^{a[y(x) - v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\, e(\Gamma_1, g^a)^{v_2}\, e(g^a,\Gamma_2)^{v_1} = g_2^{a^2[y(x) - v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots,\Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

- **Ver**(sk, P, v, σ)→0/1 Let P=(f,$\tau_1, \ldots, \tau_n$) and $\sigma=(v, \Lambda, \Gamma)$
  - Derive $r_i \leftarrow F_K(\tau_i)$ i=1…n and compute $r \leftarrow f(r_1, \ldots, r_n)$
  - Verify the invariant $\Lambda = g_d^{a\hat{}(d-1)[r - v]}$
- **Correctness**

Our Homomorphic MAC cont'd

- **CompositionEval**(ek, ϕ, $\sigma_1=(v_1, \Lambda_1, \Gamma_1)$, $\sigma_2=(v_2, \Lambda_2, \Gamma_2)$)$\rightarrow \sigma=(v, \Lambda, \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1 \Lambda_2$, $\Gamma=\Gamma_1 \Gamma_2$
  - **Multiplication:** $v=v_1 v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\, e(\Lambda_1, g^a)^{v_2}\, e(g^a,\Lambda_2)^{v_1} = g_2^{a[y(x) - v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\, e(\Gamma_1, g^a)^{v_2}\, e(g^a,\Gamma_2)^{v_1} = g_2^{a^2[y(x) - v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots, \Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

- **Ver**(sk, P, v, σ)→0/1 Let P=(f,$\tau_1, \ldots, \tau_n$) and $\sigma=(v, \Lambda, \Gamma)$
  - Derive $r_i \leftarrow F_K(\tau_i)$ i=1…n and compute $r \leftarrow f(r_1, \ldots, r_n)$
  - Verify the invariant $\Lambda = g_d^{a\hat{}(d-1)[r - v]}$
- **Correctness**
  - $y(x) = f(y_1(x), \ldots, y_n(x)) = f(r_1, \ldots, r_n) = r$

Our Homomorphic MAC cont'd

- **CompositionEval**(ek, ϕ, $\sigma_1=(v_1, \Lambda_1, \Gamma_1)$, $\sigma_2=(v_2, \Lambda_2, \Gamma_2)$)$\rightarrow \sigma=(v, \Lambda, \Gamma)$

  (simplified description for ϕ single gate and elements in G_1)

  - **Addition:** $v=v_1+v_2$, $\Lambda=\Lambda_1 \Lambda_2$, $\Gamma=\Gamma_1 \Gamma_2$
  - **Multiplication:** $v=v_1 v_2$,

$$\Lambda_1 = e(\Lambda_1,\Gamma_2)\, e(\Lambda_1, g^a)^{v_2}\, e(g^a,\Lambda_2)^{v_1} = g_2^{a[y(x) - v]}$$

$$\Gamma_2 = e(\Gamma_1,\Gamma_2)\, e(\Gamma_1, g^a)^{v_2}\, e(g^a,\Gamma_2)^{v_1} = g_2^{a^2[y(x) - v]}$$

  - Basic idea: use the graded maps to compute $\phi(\Lambda_1, \ldots, \Lambda_n)\rightarrow\Lambda$, with deg(ϕ)≤k

- **Ver**(sk, P, v, σ)→0/1 Let P=(f,$\tau_1, \ldots, \tau_n$) and $\sigma=(v, \Lambda, \Gamma)$
  - Derive $r_i \leftarrow F_K(\tau_i)$ i=1…n and compute $r \leftarrow f(r_1, \ldots, r_n)$
  - Verify the invariant $\Lambda = g_d^{a\hat{}(d-1)[r - v]}$
- **Correctness**
  - $y(x) = f(y_1(x), \ldots, y_n(x)) = f(r_1, \ldots, r_n) = r$
  - Homomorphic properties of the graded maps

Result

- **Security** under the (*D, k*)-**MDHI assumption**
  - Given (**g**, $\mathbf{g}^{x}$, …, $\mathbf{g}^{x\hat{}D}$) in $\mathbf{G}_1$, hard to compute $\mathbf{g}_k^{x\hat{}(Dk+1)}$ in $\mathbf{G}_k$
  - It can be shown hard in the generic multilinear group model, by extending the Uber assumption of [BBG05]
- **Theorem**. *If the (D,k)-MDHI assumption holds and* **F** *is a PRF, then the scheme is a secure homomorphic MAC with tags of size* $O(k^2)$ *and supports arithmetic circuits of degree* $\leq D$ *and composition circuits of degree* $\leq k$.

Comparison to other approaches

- Another approach to solve the problem is to leverage SNARKs
  - The homomorphic signature is a SNARK proof about the existence of valid signatures on the inputs
- However, by following this approach:
  - composition is achieved via recursive composition of proofs (proofs about validity of other proofs) [BCCT13]
  - function independence achieved via universal circuits
- Overall, less natural approach and likely to require non-falsifiable (knowledge) assumptions [GW11]
- In contrast, our solutions can be based on falsifiable assumptions

Conclusions & Open Problems

- We proposed new homomorphic MAC schemes
  - Based on encoding w/limited malleability
  - Multilinear maps – trading succinctness vs. composition
- **Main open questions:**
  - Can we achieve **Fully Homomorphic MACs with unbounded verification queries** ?
  - How about **Fully-Homomorphic Signatures?**

Conclusions & Open Problems

- We proposed new homomorphic MAC schemes
  - Based on encoding w/limited malleability
  - Multilinear maps – trading succinctness vs. composition
- **Main open questions:**
  - Can we achieve **Fully Homomorphic MACs with unbounded verification queries** ?
  - How about **Fully-Homomorphic Signatures?**

Interesting observation: if we assume ideal compact k-linear maps with k<p exponential, our scheme is homomorphic for all circuits of bounded depth and secure against unbounded verification queries.

Thanks