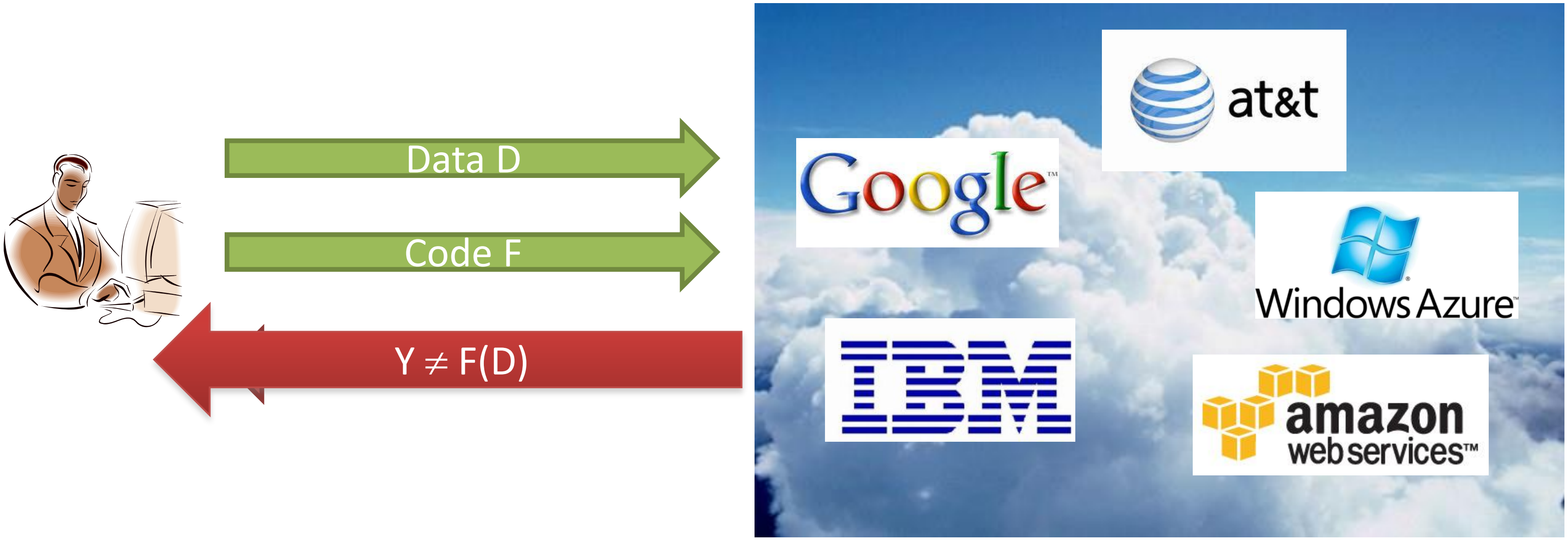
Verifiable Delegation of Computation over Large Datasets

Siavosh Benabbas
University of Toronto

Rosario Gennaro
IBM Research

Yevgeniy Vahlis
AT&T

Cloud Computing

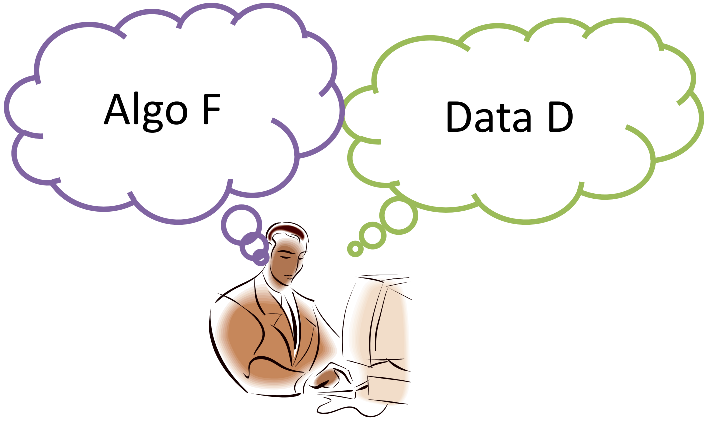


Cloud could be malicious or arbitrarily buggy (same as malicious)!

Goal: efficiently verify that Y = F(D)

Cloud Computing

What is efficient verification?

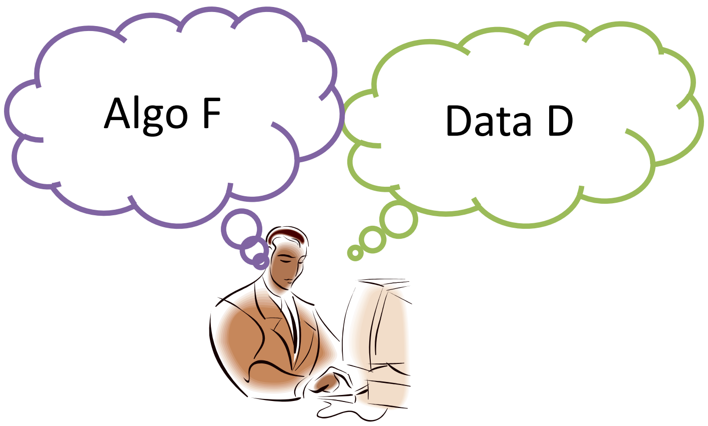


Option 1: |F|,|D| are small
but F(D) takes many steps

For example: D=N=pq, F tries all prime factors until p,q, are found

Efficient verification can be linear in |F|, |D|

Cloud Computing



What is efficient verification?

Option 2: |D| is very big
F(D) is almost linear in |D|

Plenty of examples:

- Mining medical records
- Looking up records (PIR)
- Making predictions based on trained machine learning models
- …

Linear verification is not good enough

➔ Need to be (very) sublinear in |D|

[GGP, CKV, AIK]: Any function can be verifiably delegated in the sense of option 2, assuming Fully Homomorphic Encryption

1. FHE will become practical any moment
In the mean time – can we do VC without it?

2. [GGP,CKV,AIK] require that a malicious server does not learn if it was successful in cheating – a significant restriction in practice

Our Results

- A new verifiable de
  - Delegate functions
  - The degree d is arbi
  - Extends* to multiv
  - Adaptive security –

- Verifiable databases
  - A client can outsource dictionaries $(i_1, v_1)\ldots(i_n, v_n)$
  - Make verifiable retrieval queries "Get i"
  - Update queries: "Add (i, v)", "Remove (i)", "Update (i, v)"

- Non-crypto applications
  - Keyword search
- Proofs of retrievability

- In the line of work on auth. data structures and memory checkers
- Constant communication overhead and client work (strict poly-time)
- "Constant size" assumption

Prior Work

- Long series of works related to this problem
- Interactive Proofs (B,GMR)
- Probabilistically Checkable Proofs
  - A computation can be associated with a (potentially very long) proof of correctness
  - Verifying an NP problem can take time indep. of size of statement
  - Verifier queries bits of the proof, assuming the Prover honestly provides them
- Efficient Arguments/CS Proofs [K,M]
  - Prover commits to the PCP proof
  - Verifier queries bits and verifies
  - Statement must be short “F(x) = y”. Does not deal well with large data.
- All schemes above are interactive
  - Except for Micali's CS proofs which are made non-interactive in the random oracle model
- Memory checkers [BlumEvansGemmellKannanNaor91,Ajtai02,GemmellNaor03,NaorRothblum05,DworkNaorRothVaik09,...]
  - Different model: server can only retrieve array values. The goal is to minimize the number of queries
  - Our solution is not a good memory checker (because the server works hard), but is much more efficient in communication and client work

VERIFIABLE DELEGATION OF POLYMOMIALS

Delegating a polynomial

- What does it mean to delegate a polynomial?

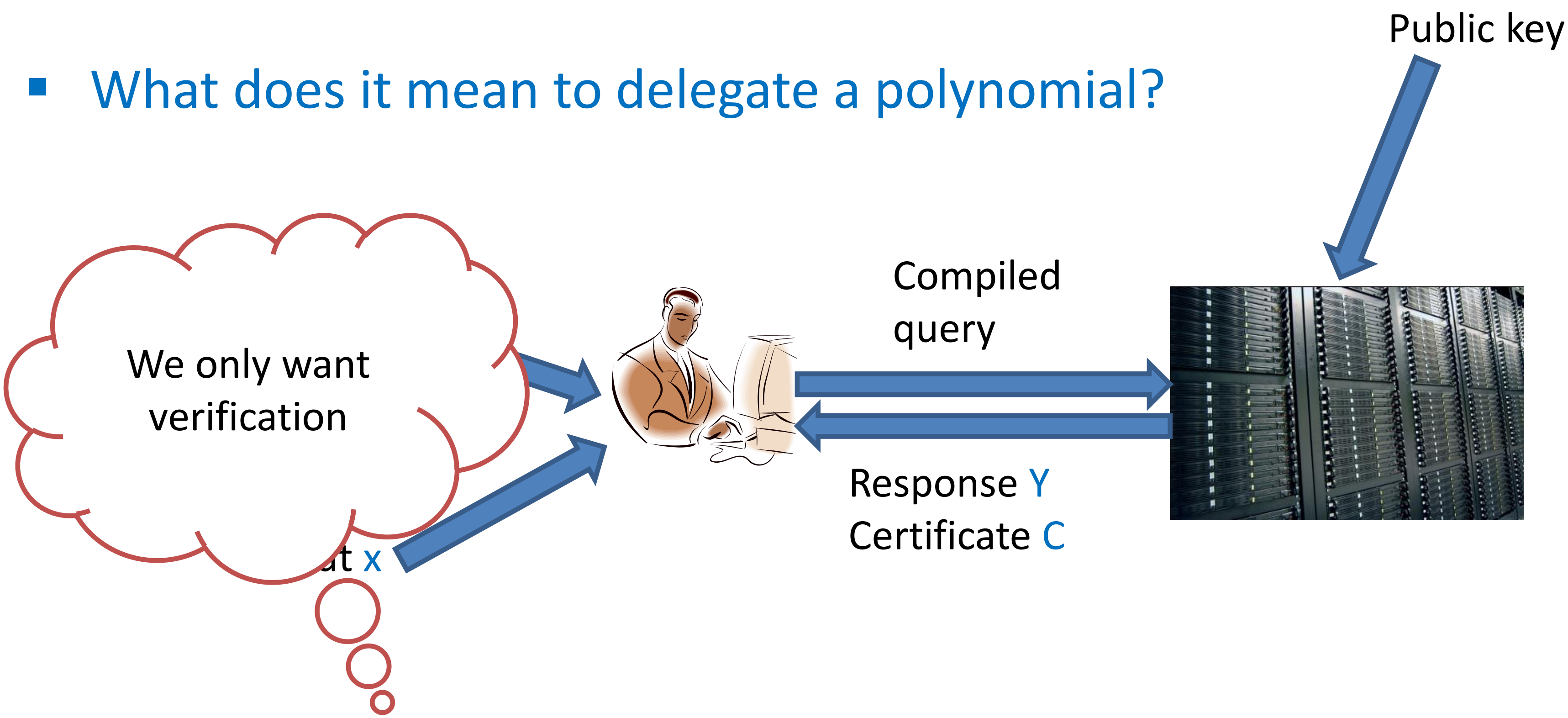
$p(x)=a_0 + a_1x + \ldots + a_dx^d$

Public key

Short secret $|SK| \ll d\,\lambda$

Delegating a polynomial

- What does it mean to delegate a polynomial?



Goal: be convinced that Y=P(x), or output "reject"

Our main tool

- Algebraic PRFs with "trapdoor" efficient algebraic operations
- A pseudorandom function **F** is a family of functions where
  - $\mathbf{F_K(\bullet)}$ is indistinguishable from a random function $\mathbf{R(\bullet)}$
- **Algebraic PRF:** the range of $\mathbf{F_K(\bullet)}$ forms an abelian group
  - **F** is *not* a homomorphism!
  - But, given $F_K(x)$, $F_K(y)$, can compute $F_K(x) \bullet F_K(y)$
  - A public generator g
  - (This is trivial)

Trapdoor Efficiency

Given a range (0,…,n) and values $(x,x^2,…,x^n)$ can compute:

$$Y = F_K(0)F_K(1)^{x}F_K(2)^{x^2}\cdots F_K(n)^{x^n}$$

using the algebraic property

Trapdoor efficiency: given (K,x) easy to compute Y (sublinear in n)

More generally: other functions of $F_K(0),…,F_K(n)$

Back to VC

Given coefficients $a_0,\ldots,a_d$

Want to delegate $p(x) = a_0 + a_1x + \ldots + a_dx^d$

Secrecy of $a_0,\ldots,a_d$ can be achieved using(singly) homomorphic encryption

Construction

- Choose random c, compute

$$g^{r_0} = F_K(0), \ldots, g$$

- Upload

$$t_0 = g^{ca_0}F_K(0), \ldots,$$
$$t_d = g^{ca_d}F_K(d)$$

and $a_0, \ldots, a_d$

- To answer query x the server computes:

$$C = t_0 \cdot t_1^x \cdot t_1^{x^2} \cdot \ldots \cdot t_1^{x^d}$$

and returns (C, P(x))

Verification

Verifier's key: PRF key K, masking coefficient c

Recall that the server is given $t_i = g^{ca_i + r_i}$

The server has (in the exponent) coefficients of

$$(cP + R)(x) = (ca_0 + r_0) + (ca_1 + r_1)x + \cdots + (ca_d + r_d)x^d$$

An honest server sends:

$$C = t_0 \cdot t_1^{x} \cdot t_1^{x^2} \cdot \ldots \cdot t_1^{x^d}$$

Verifier checks: $C = g^{cY + R(}$

If R was random,
this breaks a secure MAC

To cheat adversary has to find $g^{cW + R(x)}$, $W \neq Y$

Efficiency

- If R was random the client would have to remember $r_0, \ldots, r_d$
  - Easy to solve using any PRF (in fact, we already did that)
    Now the client only remembers the PRF key
- Even if a PRF is used, the verifier needs to check *efficiently*:

$$C = g^{cY+R(x)}$$

- Trapdoor efficiency allows exactly that!
  - Given (K, x) can compute R(x) is time sublinear in d

How?

- From strong-DDH: $g, g^{x}, g^{x^2}, \dots, g^{x^d}$ is ind. from random
- The PRF is: $F_K(x) = g^{k^x}$
- Efficiency: $F_K(0) \cdot F_K(1)^x \cdot \dots \cdot F_K(d)^{x^d} = g^{\Sigma k^i x^i}$

  Need only one exponentiation because:

  $\sum k^i x^i = (1 - k^d x^d)/(1 - kx)$
- Multivariate: $F_K(x_1, \dots, x_n) = g^{k_0 k_1^{x_1} k_2^{x_2} \dots k_n^{x_n}}$

  Generalizes Naor-Reingold

How?

- From DDH
  - Local state size is *log(d)*
- We use the Naor-Reingold PRF

$$F_K(x) = g^{k \dots}$$

- Efficiency:

$$F_K(0) \cdot F_K(1)^x \cdot \ldots \cdot F_K(d \dots)$$

$$= g^{k_0(1+k_1x_1)(1+k_2x_2)\ldots(1+k_{\lceil \lg d \rceil}x^{2^{\lceil \lg d \rceil}})}$$

In the paper:
Polynomials with logarithmic number of variables (tradeoff degree/# variables)

To summarize…

- Based on DDH/Strong-DDH we obtian an adaptively secure scheme for delegating high degree polynomials.
- Can be used for keyword search:
  - To outsource a set of keywords $\{w_1,\dots,w_n\}$ outsource the polynomial $p(x) = (x-w_1)\ (x-w_2)\bullet$⑤$\bullet(x-w_n)$
- Proofs of retrievability
  - Want to make sure that server keeps a large file F
  - Break F into blocks $F_0,\dots,F_n$
  - Outsource the polynomial $P(x) = F_0 + F_1\ x + \dots + F_n\ x^n$
  - Audit check: verifiably evaluate $P(r)$ for random r

Open directions

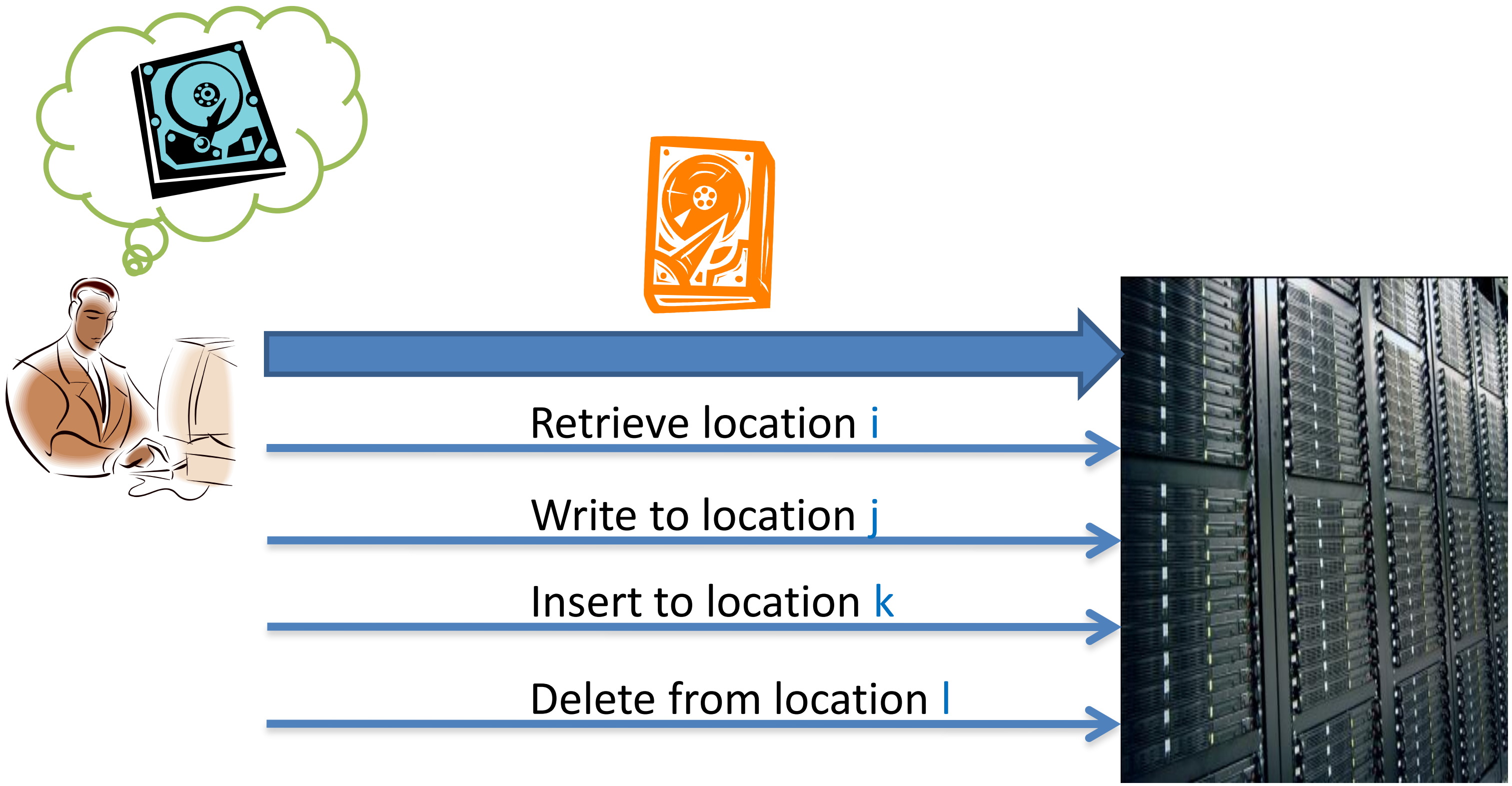
- Adaptive security for general functions
- Other efficient constructions for restricted classes of functions
- Better support for multi-variate polynomials

Thank you!

Thank you!

VERIFIABLE DATABASES!

Verifiable databases?



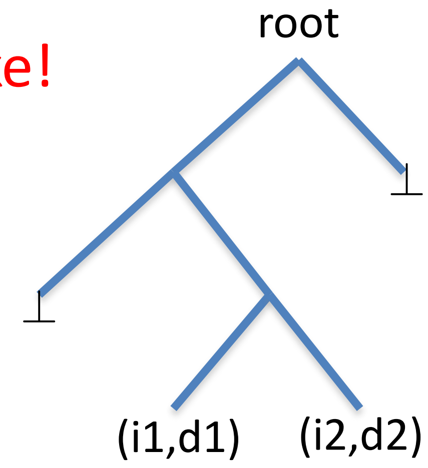
Think: SVN with untrusted repository

Very abridged history

- Merkle trees
  - Data is in stored as leaves of a tree
  - Client keeps a hash of the root
  - Queries/updates are relatively easy – log n operations each
  - Insertion/deletion is not good – based on amortization
    Too slow over a network for large storages
- Memory checkers
  - Different model: server is a RAM
  - Efficiency is counted in # of RAM queries
  - We allow server to work hard
- Authenticated Data Structures
  - Different model: trusted party has a large secret

Folklore solution without updates

- For every populated location i
  - Give the server MAC(i, data[i])
- For all other locations j
  - Upload a MAC of the shortest prefix w of j that does not extend to a populated i
- But, hard to do updates – can't revoke!



Simple Construction

- Upload $g^{ai+bv_i+F_K(i)}$ to authenticate (i, v_i)
  - This is a MAC
- Can update (insecurely):
  - To change value to u_i, send $g^{b(u_i-v_i)}$
  - Now server can find g^a, g^b
- Insertion is easy
- Efficient deletion not possible
  - Server always has certificate for (i, v_i)

- Can we fix it?
  - Need to tie all the elements together without growing client state

Composite Order Bilinear Groups

Subgroup membership assumption:

$G = G_1 \text{ x } G_2 \qquad |G_1|=p \qquad |G_2|=q$

Given $g \rangle\backslash G$, $g_2 \rangle\backslash G_2$ hard to distinguish:

(Random from G) $\approx_c$ (Random from G_2)

Back to verifiable DB

- Instead of uploading $g^{ai+bv_i+F_K(i)}$

  The client sends $g_1^{ai+bv_i+F_K(i)} g_2^{w_i}$ for a random w_i

  The key is a,b,K, and $w = \sum w_i$
- The server now sends* $g_1^{ai+bv_i+F_K(i)} g_2^{\sum w_j}$
- To update location i to value u_i client sends $g_1^{b(u_i-v_i)} g_2^{w_i'}$ and updates w

- Proof of security: the update token is indistinguishable from

  $g_1^{b(u_i-v_i)+r} g_2^{w_i'}$. (Actually, there are CCA issues)

Back to verifiable DB

- But server can't compute $g_1^{ai+bv_i+F_K(i)} g_2^{\sum w_j}$!
- All he has is $t_i = g_1^{ai+bv_i+F_K(i)} g_2^{w_i}$
- Upload additional "hints"
  $h_1 \rangle\backslash G$, $h_0 \rangle\backslash G_2$
- To respond to query "i" the server sends back:

$$C = e(t_i, h_1) \prod_{j \neq i} e(t_j, h_0)$$

- The client performs the check in the target group of the pairing

Open directions

- Adaptive security for general functions is still open
- Support higher degree polynomials
- Obtain constructions based on Lattice assumptions
- Make verifiable DB publicly checkable
- Extend VDB to support wider range of queries

Thank you!