Verifiable Delegation of Computation on Outsourced Data

Dario Fiore
IMDEA Software Institute
Madrid, Spain
Cloud Computing

**Benefits**

- **Pay-per-use**
  - No need to maintain expensive infrastructures

- **Easy Access**
  - From multiple clients
  - Wherever you are
Cloud Computing

Benefits
- Pay-per-use
  - No need to maintain expensive infrastructures
- Easy Access
  - From multiple clients
  - Wherever you are

Expensive computation

How about Security?
Security Issues in Cloud Computing

- **Integrity**
  - What if data is tampered with?
  - Are the results correct?
  - Can we verify that the Cloud operates correctly?

- **Privacy**
  - What if data is sensitive? credit cards, medical records, etc.
  - Can we prevent the Cloud from learning the data?

- Malicious Cloud?
  - External attacks
  - Coercion by governments
A Motivating Example

The company might rely on (untrusted) **Cloud Services** ...

...but wants some guarantees (privacy, integrity, ...)

- Data **too large** for being stored on small devices
Delegating Computations on Outsourced Data

**Goals**

- **Integrity**
  Untrusted cloud must **not** be able to send incorrect $y$

- **Efficiency**
  Client’s communication, storage and computation must be **minimized**

- **Open-endedness**
  Client shall continuously outsource its data

- **Program-independence**
  Client shall **not need to fix** $P$ in the outsourcing stage

**Why challenging?**

Client does not know the inputs $v_1,\ldots,v_k$

(most noticeable difference to verifiable computation)

$y = P(v_1,\ldots,v_k)$

"Compute $P$"

$y$

Is $y$ correct?
A First Attempt to Solve the Problem

Goals

- **Integrity**
  Untrusted cloud must **not** be able to send incorrect $y$

- **Efficiency**
  Client’s communication, storage and computation must be minimized

- **Open-endedness**
  Client shall continuously outsource its data

- **Program-independence**
  Client shall **not** need to fix $P$ in the outsourcing stage

Solution: homomorphic MACs

$y = P(v_1, v_2, \ldots, v_k)$

Is $y$ correct?
Our solution: Homomorphic MACs

 Goals

- **Integrity**
  Untrusted cloud must not be able to send incorrect $y$

- **Efficiency**
  Client’s communication, storage and computation must be minimized

- **Open-endedness**
  Client shall continuously outsource its data

- **Independence**
  Client shall not need to fix $P$ in the outsourcing stage

**Diagram**

- $v_1, v_2, \ldots, v_n$ are inputs to the cloud server.
- The cloud server computes $y = P(v_1, \ldots, v_k)$.
- The client verifies if $y$ is correct.

“Compute $P$“
- $v_1, v_2, \ldots, v_k$ are inputs to the client.
- The client checks for the correctness of $y$. 

**Is $y$ correct?**
Our solution: Homomorphic MACs

Goals

- **Integrity**
  - Untrusted cloud must not be able to send incorrect $y$.
  - Cloud cannot forge signatures.

- **Efficiency**
  - Client’s communication, storage, and computation should be minimized.

- **Open-endedness**
  - Client shall continuously outsource its data.

- **Independence**
  - Client shall not need to fix $P$ in the outsourcing stage.
The model: Labeled Programs [GW12, CF13]

- Problem: what does correctness mean?
- Each value \( v \) is authenticated wrt a multi-label \( L = (\Delta, \tau) \)
  - Idea: uniquely “remember” the outsourced data

Correct output means

“\( P \) executed on valid values with labels \( \Delta, \tau_1, \ldots, \tau_n \)”

= authenticated by the company

- Each program variable gets a name: a label \( \tau \)
- E.g., \( P \) computes the yearly average stock price for any company, for any year, etc.
The model: Labeled Programs

Idea: evaluate program $P$ on different data sets $\Delta_1, \Delta_2, \Delta_3, \ldots$

> reuse programs
Realization of Hom. MACs

Values $v_i \in \mathbb{Z}_p$

Computations expressed by arithmetic circuits

$P : \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p$
Our Realization [BFR13]

- Homomorphic MACs \textit{w/efficient verification} for
  - values \( v_i \in \mathbb{Z}_p \)
  - computations expressible by \textit{degree–2 arithmetic circuits} over \( \mathbb{Z}_p \)

\textbf{Applications}

- \textbf{Statistical functions}: counting, summation, (weighted) average, arithmetic mean, standard deviation, variance, covariance, least–squares, various correlation errors.
- \textbf{Similarity distance} (euclidean distance/Pearson error) between vectors/populations
Basic ingredients

- Polynomial algebra
- Pseudorandom functions PRF
  - \( \text{PRF}_K : \{0,1\}^* \rightarrow \mathbb{Z}_p \)
  - Without knowing \( K \), \( r = \text{PRF}_K(x) \) looks like sampling a random \( r \leftarrow \mathbb{Z}_p \)
How to create a MAC?
Encode value $v_i$ (an integer) with multi-label $L_i$ as a random polynomial $\sigma_i$ of degree 1.

The data owner stores a key $K$ of a pseudo-random function $\text{PRF}_K$ and a secret line $\alpha$.

How to verify a MAC?
Check the “guard” point, i.e., recompute $\text{PRF}(L_i)$ and evaluate $\sigma_i$ on 0 and $\alpha$. 
Practical Homomorphic MACs (ctd)

How to evaluate a program $P$?

- Point-wise execution of arithmetic operations
  \[ \sigma^* = P(\sigma_1, \ldots, \sigma_k) \]

  - **Addition** → addition of coefficients
  - **Multiplication** → convolution of polynomials

How to verify a result $\sigma^*$?

Compute $P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k))$ and compare to $\sigma^*(\alpha)$

Observations:

- $\sigma^*(0) = P(\sigma_1(0), \ldots, \sigma_k(0))$ = $P(v_1, \ldots, v_k)$
- $\sigma^*(\alpha) = P(\sigma_1(\alpha), \ldots, \sigma_k(\alpha))$ = $P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k))$

- **integrity**
- **communication**: succinct tags
- **verification time**: $O(P)$
Amortized Closed-Form Efficient PRFs

[BFR’13]

How to evaluate a program $P$?

- Point-wise execution of arithmetic operations
  \[ \sigma^* = P(\sigma_1, \ldots, \sigma_k) \]
  - Addition $\rightarrow$ addition of coefficients
  - Multiplication $\rightarrow$ convolution of polynomials

- Observations:
  \[ \sigma^*(0) = P(\sigma_1(0), \ldots, \sigma_k(0)) = P(v_1, \ldots, v_k) \]
  \[ \sigma^*(\alpha) = P(\sigma_1(\alpha), \ldots, \sigma_k(\alpha)) = P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k)) \]

How to verify a result $\sigma^*$?

Compute $P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k))$ and compare to $\sigma^*(\alpha)$

- **✓** integrity
- **✓** communication: succinct tags
- **✗** verification time: $O(P)$

[CF13]
Amortized Closed-Form Efficient PRFs

[BFRR’13]

How to verify a result $\sigma^*$?
Compute $P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k))$
and compare to $\sigma^*(a)$

$\Rightarrow$ Need efficient way to evaluate $P$ on PRF values

\[ P(\text{PRF}(\Delta, \tau_1), \ldots, \text{PRF}(\Delta, \tau_k)) \]

\[ P(g^{au_1+bv_1}, \ldots, g^{au_k+bv_k}) \]

isom. of $g$ for arithmetic operations

\[ g \ P(au_1+bv_1, \ldots, au_k+bv_k) \]

re-arrange

\[ g \ P^*(a,b) \]

\[ \text{PRF}_K(\Delta, \tau_i) = g^{au+bv} \]

where $(a,b) = F_{K_2}(\Delta) \in \mathbb{Z}_p^2$
and $(u,v) = F_{K_1}(\tau_i) \in \mathbb{Z}_p^2$
and $g$ generates a bilinear group $G$

based on the Decision Linear Assumption (DLin)

offline: precompute $P^*$ independent from $\Delta$
online: evaluate $P^*$ on different $\Delta$
Amortized
Closed-Form Efficient PRFs

[BFR’13]

How to verify a result $\sigma^*$?
Compute $P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k))$
and compare to $\sigma^*(\alpha)$

→ Need efficient way to evaluate $P$ on PRF values

$P(\text{PRF}(\Delta, \tau_1), \ldots, \text{PRF}(\Delta, \tau_k))$

from $k$ to 2

$g P'(a, b)$

$O(P)$
precomputation of $P'$

$O(1)$
$P(\text{PRF}(\Delta, \tau_1), \ldots, \text{PRF}(\Delta, \tau_k))$

✓ integrity
✓ communication: succinct tags
✓ amortized verification time: $O(1)$
Homomorphic MACs w/ Eff. Ver.

How to verify a result $\sigma^*$?
Compute $P(\text{PRF}(L_1), \ldots, \text{PRF}(L_k))$
and compare to $\sigma^*(a)$

→ Need efficient way to evaluate $P$ on PRF values

$P(\text{PRF}(\Delta, \tau_1), \ldots, \text{PRF}(\Delta, \tau_k))$

from $k$ to 2

Observation: our PRF maps to $\mathbb{G}_1$.
But, CF’13 MACs are encoded in $\mathbb{Z}_p$.

→ We re-encode CF’13 into $\mathbb{G}_1$.

✓ integrity
✓ communication: succinct tags
✓ amortized verification time: $O(1)$
Our Contribution: EVH–MAC

<table>
<thead>
<tr>
<th>MAC encoding</th>
<th>Computations</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hom–MAC [CF13] Eurocrypt’13</td>
<td>ring ( \mathbb{Z}_p )</td>
<td>circuits of poly–bounded degree</td>
</tr>
</tbody>
</table>

**Statistical functions:** counting, summation, (weighted) average, arithmetic mean, standard deviation, variance, covariance, least–squares, various correlation errors.

**Similarity distance** (euclidean/Pearson error) between vectors/populations
Performances: EVH–MAC

<table>
<thead>
<tr>
<th>Client operations</th>
<th>Time</th>
<th>Size of tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data outsourcing</td>
<td>1.3 ms</td>
<td>0.2 kB</td>
</tr>
<tr>
<td>Verification</td>
<td>8.8 ms</td>
<td>0.6 kB</td>
</tr>
</tbody>
</table>

Samples taken on a standard laptop, 2.5 GHz Intel i5, with 128 bits of security using the PBC library.

Server overhead (non-optimized, 128 bits): constant factor ($\sim 10^5$)

In 2007 (for non-interactive verifiable computation): server
Summary

- **Verifiable computation** over outsourced data
  - outsource your data
  - execute many functions
  - verify w/o having the inputs
  - in time independent of the input!
- Realization via **homomorphic MACs**
  - for degree-2 circuits (many important functions)
  - **first** construction with **efficient verification**
    (constant time, independent of input size)
Adding privacy

- Cloud does not learn the outsourced data: integrate with homomorphic encryption. Verifiable queries on encrypted DBs
- Verifiers (e.g. external auditors) do not learn the inputs: study notions of zero-knowledge proofs in this context

Challenges

- More expressive computations
- Reducing server’s overhead (for a significant speed-up we need different encodings)

- Not only about Cloud... (verifying computations as a tool for more applications)
Thanks!

Based on joint work with

D. Catalano [Eurocrypt’13]
M. Backes [ACM CCS’13]
R. Gennaro and V. Pastro [submitted]

Questions?