CERIAS

The Center for Education and Research in Information Assurance and Security

# A Policy-Agnostic Language for Oblivious Computation

**Qianchuan Ye** (ye202@purdue.edu)
Benjamin Delaware (bendy@purdue.edu)
Department of Computer Science, Purdue University

## Abstract

Secure multiparty computation (**MPC**) techniques allow multiple parties to collaboratively compute functions over sensitive data in a privacy-preserving manner. MPC protocols use powerful **cryptographic techniques** to achieve these privacy guarantees, making them challenging for non-experts to directly use. To address this challenge, several high-level languages have been proposed to make writing such applications accessible. These languages typically require the programmers to **embed their privacy policies into the application logic**, making it hard to audit the policies, or experiment with different policies.

This poster presents our ongoing development of a privacy-preserving language, Taype, that **decouples privacy and functionality concerns**. Two key ingredients of this language are **oblivious algebraic data types** and **tape semantics**. Oblivious algebraic data types are a form of **dependent types** with oblivious constructs, that can be used to modularly encode complex privacy policies for structured data. Tape semantics then enforce these policies during execution, enabling applications to **modularly compose policies and programs** written in a conventional way without compromising privacy.

## Example: how to create a standard dating app

- Personal profile: include personal information like gender and income
- Preference: modeled as predicates over profiles of their own and their potential soulmate's. For example, the sum of their income is greater than a certain amount

DATA

```
data profile = ...
data feature = Gender | Age | Height | Income | ...
data exp = Const int | Var bool feature | Add exp exp | ...
data pred = Le exp exp | And pred pred
          | Or pred pred | Not pred | ...
```

- Main functionality: decide if the two are a good match

FUNCTIONALITY

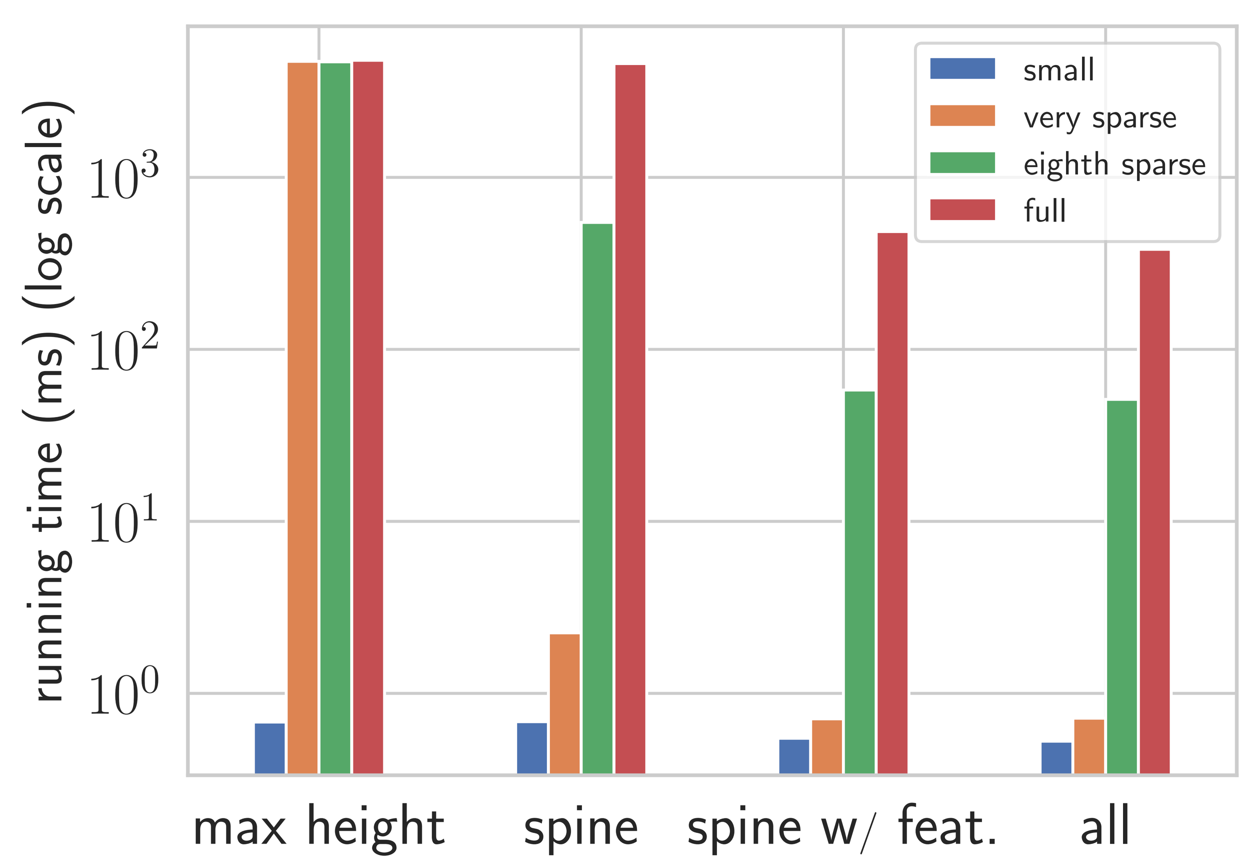
```
// Auxiliary functions
...
fn eval_exp : exp → profile → profile → int = ...
fn eval_pred : pred → profile → profile → bool = ...
// Main function
fn good_match : profile → pred → profile → pred → bool =
  λprof1 pred1 prof2 pred2 ⇒
    let b1 = eval_pred pred1 prof1 prof2 in
    let b2 = eval_pred pred2 prof2 prof1 in
    b1 && b2
```

## Privacy and performance tradeoff

Our language allows the programmers to explicitly make tradeoff between privacy and performance, by composing the functionality with different policies (i.e. public views).

Example: decision tree classification.

- Public views: maximum height, the spine, spine including the feature index of each node, and the whole tree
- The more information we are allowed to disclose, the better performance we get
- Quantify how much their performance differ, which may vary in different MPC protocols
- Quantify how performance varies in tree density
- Importantly, the decision algorithm is agnostic of the actual public views, allowing for swapping privacy policies without any changes to the program logic




## Acknowledgements

This work is partially supported by Cisco Systems and Intelligence Advanced Research projects Activity (IARPA).

## Example: how to create a private dating app

It turns out no one wants to use our dating app, because they are not willing to reveal their personal profiles or preferences! Thankfully, with our language Taype, we can turn a standard dating app into a private one in just a few simple steps!

**Step 1:** Encode a private version of the data types, with the desired policy, as oblivious algebraic data types (OADT).

- An oblivious algebraic data type is a dependent type that takes a **public view**, specifying what information can be disclosed
- Type body represents the shape of the private data
- Example: oblivious predicate. Public view is the maximum depth of the AST
- We can use other public views too, because oblivious types are independent of the functionality

```
obliv pred (k : int) =
  if k = 0 then exp 0 × exp 0          // Le
  else exp k × exp k +                 // Le
       pred (k−1) × pred (k−1) +       // And
       pred (k−1) × pred (k−1) +       // Or
       pred (k−1)                      // Not
```

**Step 2:** Define **section** and **retraction** functions for the oblivious types.

- They are essentially conversion functions, similar to encryption and decryption

```
fn pred#s : (k : int) → pred → pred k = ...
fn pred#r : (k : int) → pred k → pred = ...
```

**Step 3:** Compose the functionality and the desired privacy policy. Voilà, now we have a private soulmate matching function!

- Key idea: first "decrypt" all private input, and then run the public functionality, and finally encrypt the result
- Worry not! This does not compromise privacy thanks to our tape semantics, even though we seemingly have revealed the private input

```
fn good_match : (k : int) → profile → pred k → profile → pred k → bool =
  λk x1 p1 x2 p2 ⇒ bool#s (good_match (profile#r x1) (pred#r k p1)
                                       (profile#r x2) (pred#r k p2))
```

**Step 4:** Profit!

**Even Better:** A new version of this language (to appear in OOPSLA 2024) allows for simpler policy specifications and better performance. We can do this now:

```
fn good_match : Ψprofile → Ψpred → Ψprofile → Ψpred → bool =
  %lift good_match
```

## Obliviousness theorem

Our security type system and execution model guarantee privacy by construction.

*If* $e_1 \approx e_2$ *and* $\cdot \vdash e_1 :_{l_1} \tau_1$ *and* $\cdot \vdash e_2 :_{l_2} \tau_2$, *then*

*(1)* $e_1 \longrightarrow^n e_1'$ *if and only if* $e_2 \longrightarrow^n e_2'$ *for some* $e_2'$.

*(2) if* $e_1 \longrightarrow^n e_1'$ *and* $e_2 \longrightarrow^n e_2'$, *then* $e_1' \approx e_2'$.

## References

[1] Qianchuan Ye and Benjamin Delaware. Oblivious Algebraic Data Types. POPL 2022. https://doi.org/10.1145/3498713

[2] Qianchuan Ye and Benjamin Delaware. Taype: A Policy-Agnostic Language for Oblivious Computation. PLDI 2023. https://doi.org/10.1145/3591261

[3] Qianchuan Ye and Benjamin Delaware. Taypsi: Static Enforcement of Privacy Policies for Policy-Agnostic Oblivious Computation. OOPSLA 2024. https://doi.org/10.1145/3649861

PURDUE UNIVERSITY

CERIAS