Secure Data Types: A Simple Abstraction for Confidentiality-Preserving Data Analytics

Savvas Savvides, Julian Stephen, Masoud Saeida Ardekani, Vinaitheerthan Sundaram, Patrick Eugster

Purdue University

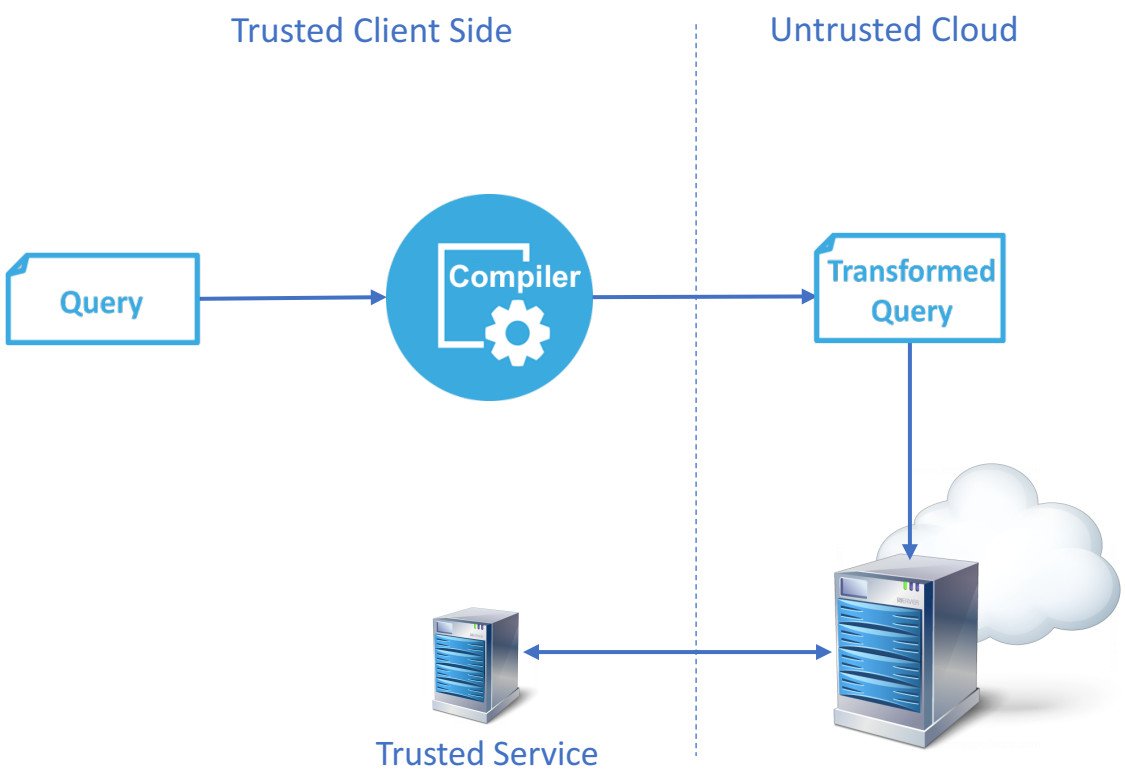
Introduction

Requirement: Confidentiality–preserving query execution

Preserving Confidentiality

- Fully homomorphic encryption (FHE)
  - Can express arbitrary computations
  - High overhead for complex queries
- Partially homomorphic encryption (PHE)
  - Allows *specific* operations over encrypted data
  - E.g., addition, multiplication, comparisons, pattern match
  - Mutually incompatible (limited expressiveness)

Current PHE-based Solutions



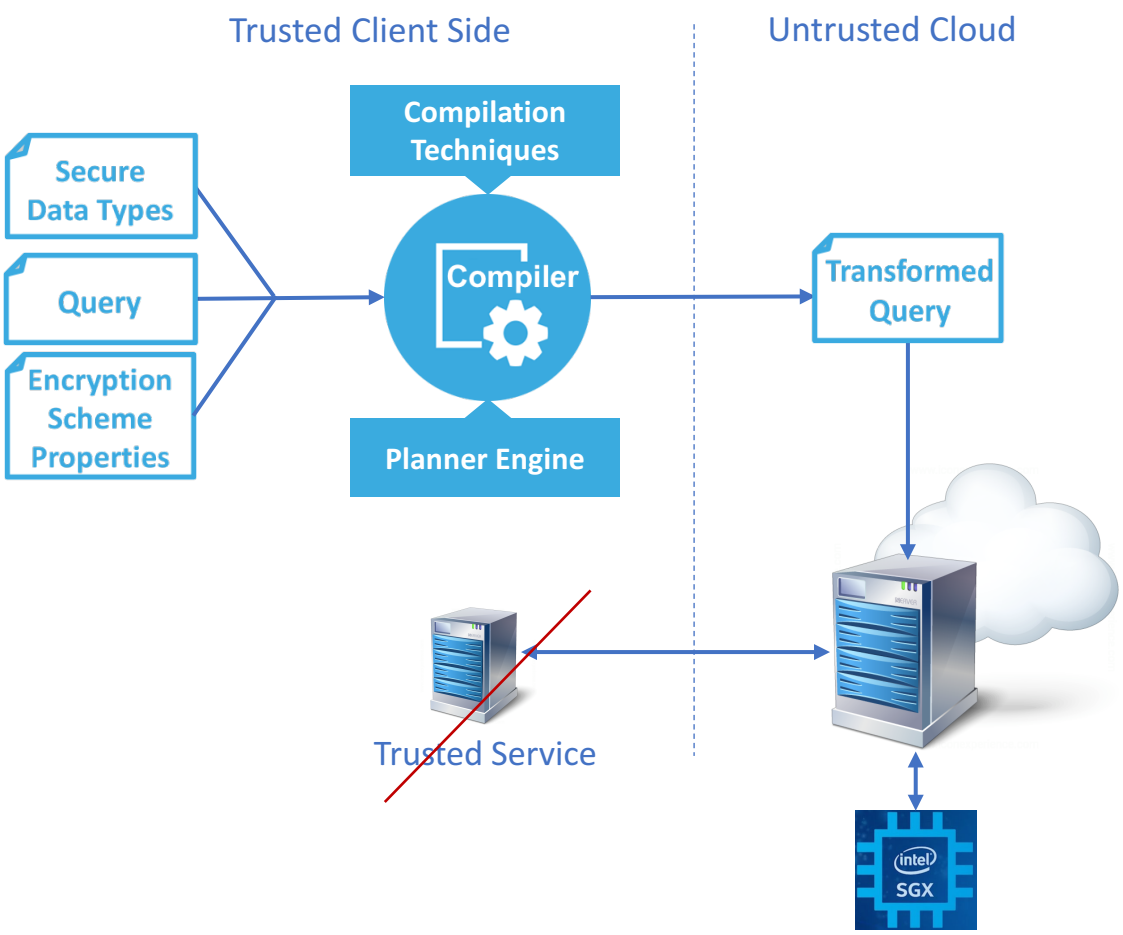
Drawbacks

1. Compilation transparent to data constraints
2. Compilation largely ignores encryption scheme properties

	ASHE [OSDI'16]	Paillier [EUROCRT'99]
E(x) + E(y)	✓	✓
E(x) + y		✓
E(x) × y		✓
Performance	symmetric	asymmetric
Security	high	high

3. No/Limited use of trusted service
   a) Give up (CryptDB [SOSP'11])
   b) Split execution (Monomi [VLDB'13])
   c) Re-encryption (Crypsis [ASE'14])

Cuttlefish



Secure data types (SDTs)

- Capture constraints and structure of data

Encryption scheme properties

- Capture supported operations, performance and security guarantees of encryption schemes

→ Compilation techniques

- More optimized queries

→ Planner engine

- More efficient deployment
- Can utilize trusted hardware

Secure Data Types

- Sensitivity levels
  - `high`, `low`, `public`
  - Accounts for different security guarantees offered by cryptosystems
- Data range
  - `+`/`-` numbers
  - Fixed `range`s, e.g., 100-200
- Composite types
  - Values containing multiple parts, e.g., dates, addresses, phones
  - E.g., `composite[(4:int[+])-(2:int[range(1-12)])-(2:int[range(1-31)])]`
- Also: decimal accuracy, uniqueness, tokenization, enumerated types, etc.

Compilation Techniques

- Expression rewriting
  - Simplify expressions involving composite types
  - E.g., `d ≥ 2010-01-01 AND d < 2011-01-01`

  → `y ≥ 2010 OR (y == 2010 AND m ≥ 01) ...`

  → `y == 2010`

- Condition expansion
  - Expand conditions to aggressively filter rows, based on **`range`** information
  - E.g., `x + y > c`

  → `y > (c − max(x)) AND x + y > c`

  Short-circuit

  - Similarly for [+, −, ×, /] and [==, >, ≥, <, ≤]

Compilation Techniques (cont.)

- Selective encryption
  - Choose encryption scheme that does not require use of trusted service
  - E.g., `(x + y) × z` where `z` is **`public`**

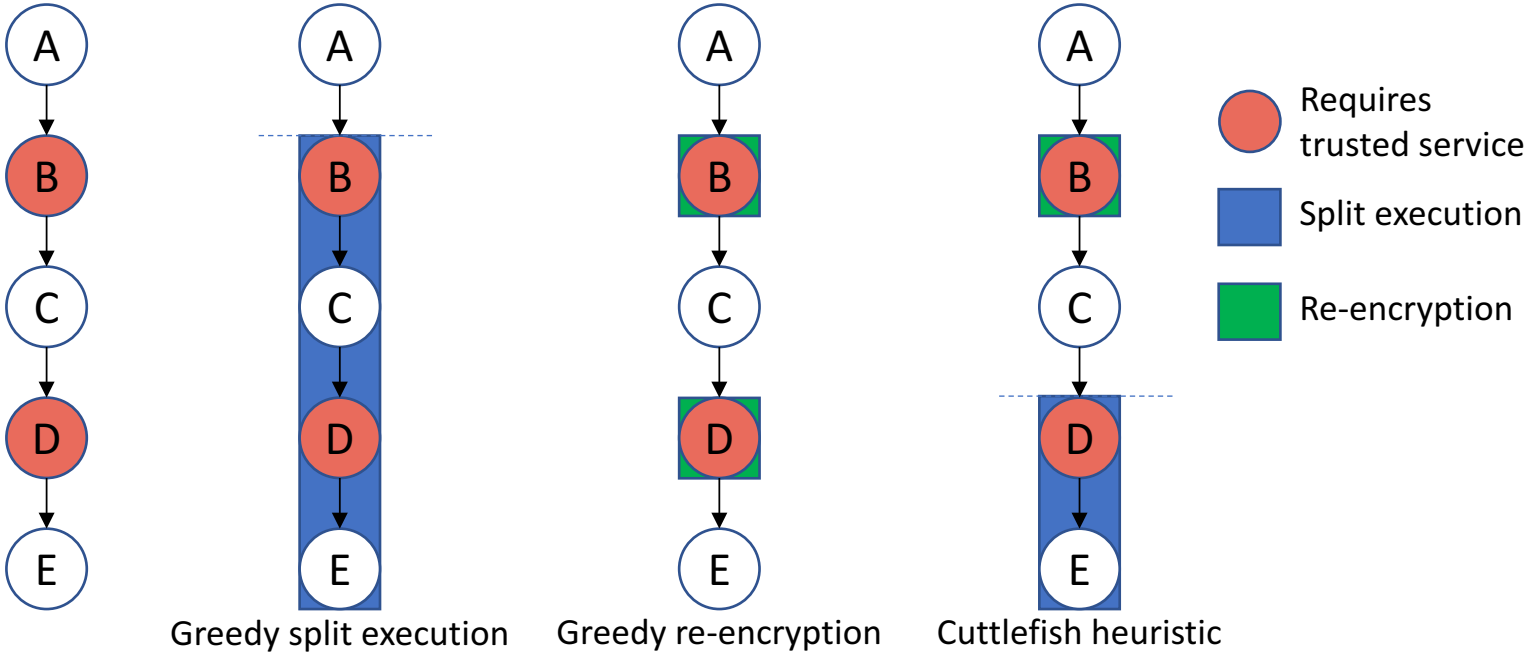
→ `(ashe(x) + ashe(y)) × z`

→ `(paillier(x) + paillier(y)) × z`

	ASHE [OSDI'16]	Paillier [EUROCRT'99]
E(x) + E(y)	✓	✓
E(x) + y		✓
E(x) × y		✓
Performance	symmetric	asymmetric

- See paper for more compilation techniques

Planner Engine



- Cuttlefish Heuristic
  - Use a cost model to choose between re-encryption and split execution at each step
- Utilize *trusted hardware*, if available, to deploy an in-cloud re-encryption service

Evaluation

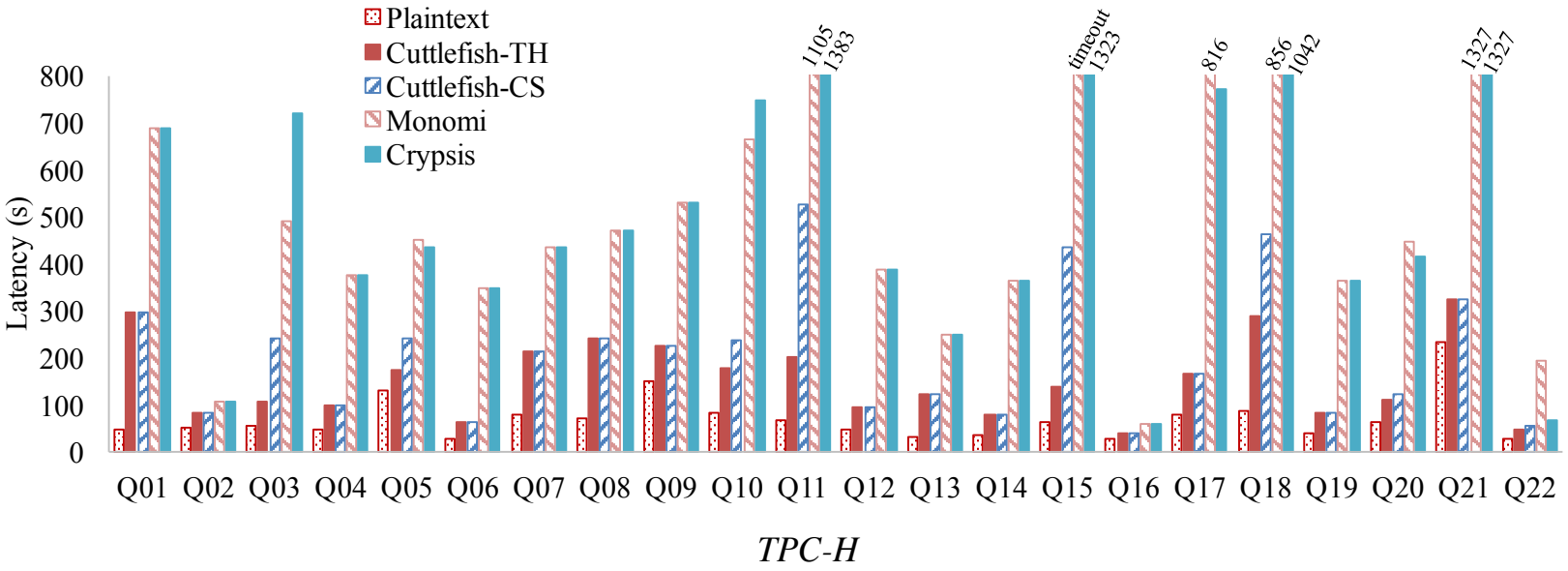
Cuttlefish

- Apache Spark 2.1
- Cuttlefish-TH: trusted service deployed using trusted hardware (Intel SGX)
- Cuttlefish-CS: trusted service deployed using remote client side

Setup

- TPC-H and TPC-DS (subset) at scale 100
- Cloud: 20 AWS m4.xlarge instances (4 CPUs and 16GB memory)
- Client: 1 AWS c4.2xlarge instance (8 CPUs and 15GB memory)

System Performance



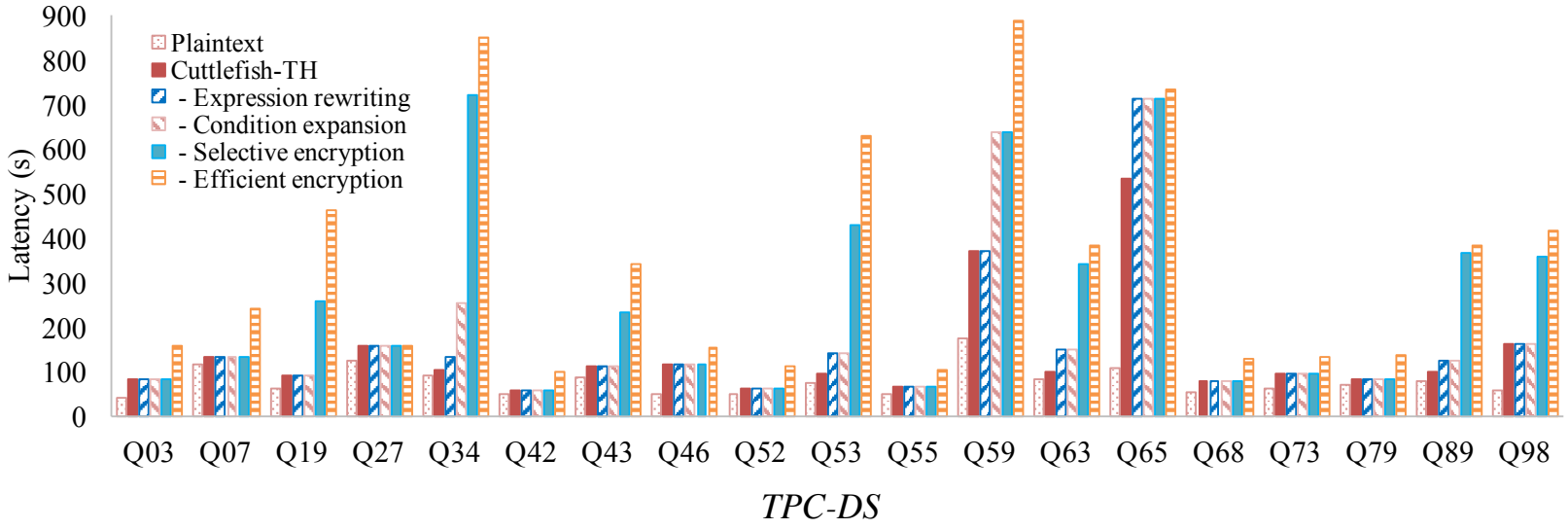
Average overhead compared to plaintext

- Cuttlefish-TH: 2.34 ×
- Cuttlefish-CS: 3.05 ×

Average performance gains

- 3.35× faster than Monomi
- 3.71× faster than Crypsis

Compilation Techniques Performance



Average overhead compared to plaintext

- With Compilation techniques: 1.69 ×
- Without Compilation techniques: 4.23 ×

Conclusion

- Cuttlefish enables efficient data analytics in public clouds

- Secure data types
  - Capture constraints and structure of data
- Compilation techniques
  - Enable more efficient queries
- Planner engine
  - Optimized use of trusted service

Thank you!