Secure Distributed Data and Event Processing at Scale
Where are we now?
Roadmap

• Motivation: distributed data and event processing
• Problem: untrusted third-party infrastructure
• Solution fragments: security mechanisms
• Requirements: what do we need
• Tradeoffs: what comes first
• HYDRA: a carefully balanced approach
• Conclusions: lessons learned and open challenges
Motivation
Distributed data and event processing

• Rise of “remote execution”
  • Advent of cloud computing, edge
  • Massive resources available on demand, wherever incl. close-by
  • “Distribution by commodity”
• Further digitization of society
  • IoT, wearable computing, …
  • “Distribution by necessity”
Problem

Untrusted third-party infrastructure

• So you want to do distributed processing on large data sets?
  • Do not have your own cluster?
  • Just use cloud or edge data centers, or both!
  • And say goodbye to your data…

• Causes
  • Multitenancy, internal threats, big government
  • Use of “foreign” software, tool sprawl
  • Almost half of data breaches happen in cloud, with average cost of $4.35M!!! [IBM’22]
Data Analytics and Confidential Computing
Relevance for society and economy

• 2022 revenue of DataBricks, a leader in data analytics, estimated at $1.24B up from $800M in 2021, 90% from platform based on Spark [Sabra’23].

• DataBricks/Spark lacks support for security required by many scenarios.

• Gartner predicts size of confidential computing market to be $54B by 2026 [Gartner’23].
Solution Fragments

- encryption
- homomorphic
- secure
- trusted
- circuit
- extensions
- sme
- fhe
- phe
- partially
- attribute-based
- functional
- computing
- memory
- multi-party
- software
- partially
- enclave
- garbled
- software
- multi-party
- attribute-based
- trusted
- execution
- memory
- software
Mechanisms
Software (SW)

- Cryptographic primitives, e.g.,
  - Fully homomorphic encryption [Gentry’10]
  - Partially homomorphic encryption (PHE)
  - Functional encryption [Boneh et al.’11]
  - Attributed-based encryption [Shamir’84]
  - Garbled circuits [Yao’86]
  - …

Mechanisms
Hardware (HW)

- Trusted execution environments, e.g.,
  - Intel Secure Guard eXtensions (SGX)
  - AMD Secure Memory Encryption (SME)
  - ARM Trustzone
  - Amazon Nitro
  - ...
Mechanisms

Hardware (HW)

• Trusted execution environments, e.g.,
  • Intel Secure Guard eXtensions (SGX)
  • AMD Secure Memory Encryption (SME)
  • ARM Trustzone
  • Amazon Nitro
  • ...

Which one(s) to use? How?
Requirements?
More/higher/stronger is better

- Transparency
  - Very few data analysts are experts in security

- Efficiency
  - Many potential users deterred by high overhead w.r.t. security-agnostic execution
  - 10x slowdown critical

- Portability
  - Lock-in with cloud or platform providers is never good, may support different (HW) mechanisms
  - New mechanisms keep being proposed
Requirements (cont’d)?
Higher/stronger is better

- Interoperability
  - Important especially if combining cloud and edge in data pipelines, notably continuous processing
  - Interoperability ≠ portability: if components can interact across platforms doesn’t mean they can move
  - For simplicity we consider interoperability as platform-independence including portability
- Security
  - What is the attacker/threat model and guarantees given in the face of those? If little security is offered …
  - How strongly are they substantiated? Informal, “formal" on paper, mechanically verified, …?
- Expressivity
  - Supporting simpler computations is always easier
Tradeoffs

- Transparency vs interoperability: user-facing language?
  - SecureScala [Hauck et al.’16]
- Transparency vs efficiency: transparency for whom?
  - Cuttlefish [Savvides et al.’17]
- Security vs efficiency: what say you (user)?
  - Symmetria [Savvides et al.’20]
- Expressivity vs efficiency: continuous is always harder
  - STYX [Stephen et al.’16], C3PO [Savvides et al.’22]


Tradeoffs

- Transparency vs interoperability: user-facing language?
  - SecureScala [Hauck et al.’16]
- Transparency vs efficiency: transparency for
  - Cuttlefish [Savvides et al.’17]
- Security vs efficiency: what say you (user)?
  - Symmetria [Savvides et al.’20]
- Expressivity vs efficiency: continuous is always harder
  - STYX [Stephen et al.’16], C3PO [Savvides et al.’22]
Tradeoffs

- Transparency vs interoperability: user-facing language?
  - SecureScala [Hauck et al.’16]
- Transparency vs efficiency: transparency for whom?
  - Cuttlefish [Savvides et al.’17]
- Security vs efficiency: what say you (user)?
  - Symmetria [Savvides et al.’20]
- Expressivity vs efficiency: continuous is always harder
  - STYX [Stephen et al.’16], C3PO [Savvides et al.’22]


HYDRA

Hybrid approach to confidentiality-preserving data analytics

• Transparency: data analysts write queries *agnostically to security constraints*.

• Efficiency: *over 10x faster* than state of the art [Mangipudi et al.’23].

• Interoperability: *combining* software mechanisms (homomorphic encryption schemes, e.g., Paillier [Paillier’99]) and hardware mechanisms (trusted execution environments, e.g., Intel SGX) from extensible sets.

• Security: *secure end-to-end with formally verified guarantees*.

• (Expressivity: focus on distributed data analytics)

---


Transparency

• Full transparency is unrealistic, undesired

• Can define/split different roles (SecureScala, Cuttlefish)
  1. *Data analysts*: know queries; but not computer scientists, not expert programmers
     • Should be able to write queries agnostically to security
  2. *Security experts*: know security mechanisms, attacks, cloud platforms, etc.
     • Define (static) security constraints and policies
  3. *Data managers*: know data semantics, understand security basics
     • Define (static) security requirements for data together with 2.
Efficiency

- SW can outperform HW (Cuttlefish)
  - E.g., PHE > SGX sometimes
- Want SW and HW
- Differences between SW (Symmetria)
- Researchers frantically working on new SW solutions
- Researchers and engineers frantically working on new HW solutions
- Want to be able to integrate new solutions without complete redesign

<table>
<thead>
<tr>
<th>Operation</th>
<th>PHE scheme</th>
<th>Plaintext</th>
<th>PHE</th>
<th>SGX</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter (_=_)</td>
<td>AES-ECB [Daemen and Rijmen 2002]</td>
<td>5.4 s</td>
<td>5.8 s</td>
<td>8.0 s</td>
</tr>
<tr>
<td>filter range</td>
<td>OPE [Boldyreva et al. 2009]</td>
<td>5.7 s</td>
<td>8.7 s</td>
<td>10.7 s</td>
</tr>
<tr>
<td>filter match</td>
<td>SWP [Song et al. 2000]</td>
<td>5.4 s</td>
<td>8.8 s</td>
<td>9.4 s</td>
</tr>
<tr>
<td>groupby</td>
<td>AES-ECB [Daemen and Rijmen 2002]</td>
<td>6.2 s</td>
<td>7.2 s</td>
<td>57.7 s</td>
</tr>
<tr>
<td>sort</td>
<td>OPE [Boldyreva et al. 2009]</td>
<td>7.2 s</td>
<td>13.0 s</td>
<td>41.0 s</td>
</tr>
<tr>
<td>select (_+_)</td>
<td>Paillier [Paillier 1999]</td>
<td>5.2 s</td>
<td>19.4 s</td>
<td>7.8 s</td>
</tr>
<tr>
<td>select (_\times_)</td>
<td>ElGamal [ElGamal 1985]</td>
<td>5.1 s</td>
<td>22.4 s</td>
<td>7.8 s</td>
</tr>
</tbody>
</table>

Efficiency

- Generalized Policy-Based Noninterference for Efficient Confidentiality-Preservation

Appendix A

Empirical measurements

We designed HydraHYBRID based on an empirical assessment of individual operations' execution times when run using different security mechanisms (see Table 4). For the assessment we used Spark in SGX-enabled single-node setup and a synthetic relation with 1 million rows, encrypted as follows:

- For plaintext execution data is unencrypted.
- For PHE it is encrypted under a PHE scheme supporting the specific operation (see “PHE scheme” column).
- For SGX we encrypt using AES-GCM.

We observe that PHE is faster than SGX for some operations and slower for others. In particular, for operations using AES-ECB [Daemen and Rijmen 2002], OPE [Boldyreva et al. 2009], and SWP [Song et al. 2000], PHE performs better than SGX. PHE overhead of these operations is relatively low, as the difference with plaintext computation is mostly due to a slightly larger size of operands. In contrast, SGX's overheads due to crossing the enclave boundary are substantial.

The situation is the opposite for operations using Paillier and ElGamal: ciphertext size increases substantially and costly multi-precision arithmetic operations are required.

Table 4. Execution times of individual operations using plaintext, PHE, and SGX. “PHE scheme” denotes the scheme data is encrypted under to enable the operation in PHE.
Interoperability

• **Support extensible** sets of HW and SW mechanisms
  • Domains \( \mathcal{D} \): capture HW mechanisms and infrastructure providers
    • Can treat same HW mechanism differently in different infrastructures
  • Crypto(graphic) schemes \( \mathcal{S} \): capture SW mechanisms
• Why not just 1 dimension/set of mechanisms?
  • Isn’t SGX just AES(-NI) in HW?
    • Different characteristics and use
  • Benefits of combining HW with SW mechanisms for *same* data
Security

- Extensible sets $\mathcal{D}$ and $\mathcal{S}$
- How to integrate mechanisms with different guarantees?
  - E.g., different threat models, known attacks [Fei et al.’21]
- Basis: honest-but-curious (HbC) attacker model
  - Focus on confidentiality
- Let security experts define what they trust and where (Cuttlefish)
- Leverage formal techniques to verify guarantees end-to-end (SecureScala)

Security Policy \(\mathcal{S}\)

- Custom finite set of labels/levels \(\mathcal{L}\) with lowest element \(\bot\) [Denning’76]
- Arranged according to lattice \(\langle \mathcal{L}, \leq, \sqcup, \sqcap \rangle\)
  - Partial order \(\leq\) reflexive antisymmetric binary relation on \(\mathcal{L}\)
    - \(l_1 \leq l_2\): confidentiality requirements imposed by \(l_2\) are at least as strict as \(l_1\); always safe to replace \(l_1\) with \(l_2\) when labelling data
- Least upper bound operator \(\sqcup\): \(\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}\)
- Greatest lower bound operator \(\sqcap\): \(\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}\)
- Security policy \(\mathcal{S}\) assigning levels to combinations of SW and HW mechanisms \(\mathcal{S}: \mathcal{L} \rightarrow 2^{\mathcal{D} \times \mathcal{S} \cup \emptyset}\)

Example Security Policy Setup
All steps performed once only

1. Define secrecy levels

<table>
<thead>
<tr>
<th>High</th>
<th>Low</th>
<th>Public</th>
</tr>
</thead>
</table>

Cf. US DoD

2. Assign mechanisms to levels

<table>
<thead>
<tr>
<th>Label</th>
<th>Domain</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>CLNT, SGX</td>
<td>Ø</td>
</tr>
<tr>
<td>High</td>
<td>CLD</td>
<td>AES-GCM</td>
</tr>
<tr>
<td>Low</td>
<td>CLD</td>
<td>SWP, AES-ECB, Paillier, ElGamal, OPE</td>
</tr>
</tbody>
</table>

E.g., data at level High can be handled only on the client side (CLNT), by Intel SGX (SGX), or in the cloud (no SGX) only when using AES-GCM encryption

3. Label data with levels

<table>
<thead>
<tr>
<th>Relation</th>
<th>Field</th>
<th>Type</th>
<th>Label</th>
<th>Security type (inferred)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers</td>
<td>custId</td>
<td>Str</td>
<td>Low</td>
<td>(Str^{AES-ECB},Low)</td>
</tr>
<tr>
<td></td>
<td>bal</td>
<td>Dbl</td>
<td>High</td>
<td>(Dbl^{AES-GCM},High)</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orders</td>
<td>orderId</td>
<td>Str</td>
<td>High</td>
<td>(Str^{AES-GCM},High)</td>
</tr>
<tr>
<td></td>
<td>custKey</td>
<td>Str</td>
<td>Low</td>
<td>(Str^{AES-ECB},Low)</td>
</tr>
<tr>
<td></td>
<td>price</td>
<td>Str</td>
<td>High</td>
<td>(Dbl^{AES-GCM},High)</td>
</tr>
<tr>
<td></td>
<td>date</td>
<td>Int</td>
<td>Low</td>
<td>(Int^{OPE},Low)</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Security types automatically inferred at query transformation, e.g. AES-GCM encryption for price tags in Orders
Complete Workflow

Security policy
Annotated data schema
Heuristic

Query → Transformer → Annotated query → Compiler & optimizer → Final execution plan → Result

Data manager, security expert
Security expert, performance engineer

Data analyst

Heuristic

PHE
SGX
...
Core Language

Lambda with relations and relational operators

Type $\kappa ::= \bar{\kappa} \to_\Delta \kappa \mid \{ f : (p^S \ I) \} \mid T\{ f : (p^S \ I) \} \mid (p^S \ I)$

Prim type $p ::= \text{Int} \mid \text{Dbl} \mid \text{Str} \mid \text{Bool} \mid ...$

Scheme $s ::= \emptyset \mid \text{AES-GCM} \mid \text{ElGamal} \mid \text{Paillier} \mid ...$ (in one-to-one correspondence with $S_\emptyset$)

Domain $d ::= \text{CLNT} \mid \text{SGX} \mid \text{SEV} \mid \text{CLD} \mid ...$ (in one-to-one correspondence with $D$)

Value $v ::= T\{ f : \bar{v} \} \mid \{ f : v \} \mid c^S \mid \lambda[d](x: \kappa). e \mid f$

Expression $e ::= v \mid x \mid e(\bar{e}) \mid \oplus(\bar{e}) \mid \{ f : e \} \mid e.f \mid \text{table(name)} \mid \theta(\bar{e}) \mid \text{encr}(e,s) \mid \text{decr}(e) \mid [e]_d$

Prim ops $\oplus ::= + \mid - \mid ... \mid \wedge \mid \lor \mid ...$

Query ops $\theta ::= \text{filter} \mid \text{proj} \mid \text{cross} \mid \text{agg} \mid ...$

- **Blue** parts not used by data analysts
- **Green** parts only used during evaluation
Transformation

1. Original security-agnostic query

```java
agg(filter(cross(table(Customers),
    filter(table(Orders),
        λ(rO: /* Orders */). rO.date < 16052002)),
    λ(rCO: /* Customers + Orders */). rCO.custId == rCO.custKey),
    custId, 0,
    λ(rP: {price: Dbl}, acc: Int).
        acc + rP.price)
```
Transformation

2. Automatically transformed query with security annotations

```
agg(filter(cross(table(Customers),
    filter(table(Orders),
        \[SGX\](rO: /* Orders */). rO.date < 0x..OPE)),
        \[SGX\](rCO: /* Customers + Orders */). rCO.custId == rCO.custKey),
    custId, 0x..AES-GCM,
    \[SGX\](rP: {price: (Dbl^{AES-GCM}, High)}, acc: (Dbl^{AES-GCM}, High)).
    encr(decr(acc) + decr(rP.price), AES-GCM))
```
Transformation Heuristics and Framework

- Formal framework denoting well-formed transformations
  - (Partially) enforced through APIs and libraries for writing heuristics
- Information flow type system verifies transformed queries
  - Also queries written in full language
Guarantees

Informal

- Based on noninterference [Goguen&Meseguer’82]
  - Generalized to our security policy, thus $S$-noninterference
- Partially ordered set of adversaries $\langle A, \subseteq \rangle$
  - $A \in A$ downward-closet set $A \subseteq D \times S\emptyset$ it can break
  - If $S(l) \cap A = \emptyset$ then $A$ is unable to see any inputs at level $l$
- Adversary who can’t break anything at level $l$ can’t see difference in outputs of computation if inputs at level $l$ or higher are changed

Guarantees

Formal

$(\Omega$ table store, $\rho$ plaintext schema, $\kappa$ type w/ annotations)

Definition 4 (Level-l $\mathcal{S}$-noninterference $\mathcal{S}$-NI$(e)_{\rho,d,l}$). Expression $e$ has $\mathcal{S}$-NI$(e)_{\rho,d,l}$ property dubbed level-l $\mathcal{S}$-noninterference if and only if for any two stores $\Omega_1$ and $\Omega_2$ satisfying $\rho$, $\Omega_1 \sim^l_\rho \Omega_2$, and any two values $v_1$ and $v_2$, $e \xrightarrow{\Omega_1} * v_1$ and $e \xrightarrow{\Omega_2} * v_2$, it holds that $v_1 \sim^{d,l}_{\rho} v_2$.

Definition 5 ($\mathcal{S}$-noninterference $\mathcal{S}$-NI$(e)_{\rho,d}$). Expression $e$ has $\mathcal{S}$-noninterference property $\mathcal{S}$-NI$(e)_{\rho,d}$ if and only if it has level l $\mathcal{S}$-noninterference property $\mathcal{S}$-NI$(e)_{\rho,d,l}$ for every l in $\mathcal{L}$.

Theorem 1 (Soundness). If there exists non-function $\kappa$, s.t., $\rho \vdash_d e : \kappa$ w.r.t. $\mathcal{S}$ then $\mathcal{S}$-NI$(e)_{\rho,d}$.

Theorem 2 (Subject Reduction). Let $\vdash_d \Omega : \rho$, $\rho \vdash \Gamma \vdash_d e : \kappa$ and $e \xRightarrow{\Omega} e'$ then $\rho \vdash \Gamma \vdash_d e' : \kappa$. 
Guarantees

Transformation

Definition 1 (query transformation). Function $\tau[\cdot, \cdot]$ is a query transformation iff $\forall \rho, e, \text{ and } (\rho', e') = \tau[\rho, e]$, we have $e \sim e'$ and $\rho \sim_S \rho'$.

Theorem 3 (Transformation correctness). For query transformation $\tau[\cdot, \cdot]$, schemata $\rho$, and expression $e$, let $(\rho', e') = \tau[\rho, e]$. If $\rho' \vdash_d e' : \kappa$ for some domain $d$ and type $\kappa$, and also $e \xrightarrow{\Omega}^* v$ for some table store $\Omega$ satisfying $\rho$, then for any $\Omega' = \text{encrVal}(\Omega, \rho')$, there exists $v'$, s.t., $e' \xrightarrow{\Omega'}^* v'$ and $\text{decrVal}(v') = v$. 
Implementation

- Built on Apache Spark
  - Used by data analysts through original APIs (SparkSQL)
- Leveraging Catalyst extensible query optimizer (query analyzer, optimizer, execution planner)
  - Input data encrypted in analysis phase (3.2 kLoC Scala)
  - Logical optimization in analysis phase (3.4 kLoC Scala)
  - Physical opt. in planning phase (6.2 kLoC Scala and 100 LoC Java)
  - Code generation step (29 kLoC C++, 2.5 kLoC C, 568 LoC Scala)
- SGX used via JNI (mini-interpreter implemented in C)
- Simple HYDRA Hybrid PHE&SGX heuristic: PHE until hitting limit of PHE or operation faster in SGX
Evaluation

Execution time

- TPC-H benchmarks, average of 5 runs, labeling from [Savvides et al.’17]

- HYDRA (PHE) on average 1.6x faster than Cuttlefish

- HYDRA (SGX) on average 11.3x faster than Opaque [Zheng et al.’17] (ORAM disabled)

- HYDRA (Hybrid) 1.7x and 1.6x times faster than HYDRA PHE and HYDRA SGX respectively
  - 2.7x and 17.9x faster than Cuttlefish and Opaque respectively

Evaluation

Transparency

• What if a programmer wanted to use/combine mechanisms manually?

• Proxy for effort: #ops and distribution for HYDRA Hybrid in TPC-H
  • Large number of operators gives idea of all possible combinations!

• 16 of 22 queries use mix of PHE and SGX

• Note: 18 of 22 queries use plaintext operations
Extensibility

1. Adding/changing data relations (schema changes): (re-)label following $\mathcal{L}$ (data manager w/ security expert)

2. Modifying security policy $\mathcal{S}$
   A. To assign mechanisms differently to levels: change only $\mathcal{S}(l)$ (security expert)
   B. To add/remove levels: change $\langle \mathcal{L}, \leq, \sqcup, \sqcap \rangle$ and $\mathcal{S}(l)$ (security expert)
      i. Possibly revisit data labeling (data manager w/ security expert, cf. 1.)

3. Adding whole new mechanism to $\mathcal{D}$ or $\mathcal{S}$
   - Some glue code
   - Augment heuristic (security expert w/ performance engineer)
   - Anybody wanting to use either 2.A. or 2.B.
Ongoing/Future Work and Open Challenges
HYDRA

• Placement

• Heuristic framework
  • More advanced heuristics
  • Improved verification of properties
  • Tracing back compilation/execution errors

• Security
  • Stronger guarantees, e.g., intermediate results, relation size leakage
  • Precision of guarantees, cf. universal composability [Canetti’00]

• Beyond HbC: integrity  
Ongoing/Future Work and Open Challenges
Beyond HYDRA

• Multi-party computing (MPC)

• Differential privacy (DP) [Dwork et al.’17]

• Continuous processing
  • Spark allows incremental computing, micro-batching
  • Main efforts in general still around data stores, e.g., *data clean rooms*

• Vs MPC and/or DP [Pettai&Laud’15][Cao et al.’19]
Corporate and Open Community Initiatives

- Fortanix: specific products and generic platform, PHE & SGX ($130M raised)
- Opaque Systems: data clean rooms based on Opaque ($32M raised)
- Decentrique: data clean rooms, SGX ($26M raised)
- Decision3: chain(?) and SGX & AWS ($100K raised)
- <Your company here>
- Veracruz: backed by Linux Foundation’s Confidential Compute Consortium (C3)
- Apache Teaclace: generic secure computing with SGX & Trustzone
- …
- General lack: continuous processing support, security guarantees that are verified (“securified”)
Thank You