Formal programming techniques for secure data processing

Patrick Eugster, USI Lugano

w/ P. Chuprikov, S. Mangipudi, A. MohammadPur-Fard, G. Prendi, S. Savvides, M. Viering

Confidential Computer Consortium Sep 19 2024

Confidential computing

- Rise of the trusted execution environment (TEE)
- Natural and necessary evolution
- Large number of data breaches in the cloud [IBM'23]
- Standard encryption insufficient for data in use

[IBM'23] https://www.ibm.com/reports/data-breach

Many different offerings

- Intel SGX
- AMD SEV
- ARM CCA
- AWS Nitro
- Intel TDX
- …
- Nvidia H100
- …

Software-based security mechanisms

- E.g. (partial/full) homomorphic encryption (P/F)HE
- Need software anyway
- Easy to inspect
- Easy to deploy
- Efficient*

Differences

Differences

- Features/Functionalities

Differences

- Features/Functionalities

[Zhao et al.'22] Towards A Secure Joint Cloud With Confidential Computing. X. Zhao, M. Li, E. Feng, Y. Xia. IEEE JCC 2022.

	SGXv1	**Scalable SGXv2**	**SEV-SNP**	**TDX**	**TrustZone**	**Realm**	**Nitro**	**Penglai**	**Keystone**	**H100**
Architecture	x86-64	x86-64	x86-64	x86-64	Arm	Arm	x86-64	RISC-V	RISC-V	GPU
Abstraction	enclave	enclave	VM	VM	PM	VM	VM	enclave	PM	vGPU
Instances	unlimited	unlimited	509	unlimited	1	unlimited	unlimited	unlimited	16	7
Encryption	●	●	●	●	○	●	○	●	○	●
Integrity	●	◐	◐	◐	○	◐	○	●	○	●
Freshness	●	○	○	○	○	○	○	●	○	○
Attestation	●	●	●	●	○	●	●	◐	◐	●

Differences

- Features/Functionalities
- Guarantees

[Zhao et al.'22] Towards A Secure Joint Cloud With Confidential Computing. X. Zhao, M. Li, E. Feng, Y. Xia. IEEE JCC 2022.

	SGXv1	**Scalable SGXv2**	**SEV-SNP**	**TDX**	**TrustZone**	**Realm**	**Nitro**	**Penglai**	**Keystone**	**H100**
Architecture	x86-64	x86-64	x86-64	x86-64	Arm	Arm	x86-64	RISC-V	RISC-V	GPU
Abstraction	enclave	enclave	VM	VM	PM	VM	VM	enclave	PM	vGPU
Instances	unlimited	unlimited	509	unlimited	1	unlimited	unlimited	unlimited	16	7
Encryption	●	●	●	●	○	●	○	●	○	●
Integrity	●	◐	◐	◐	○	◐	○	●	○	●
Freshness	●	○	○	○	○	○	○	●	○	○
Attestation	●	●	●	●	○	●	●	◐	◐	●

Differences

- Features/Functionalities
- Guarantees

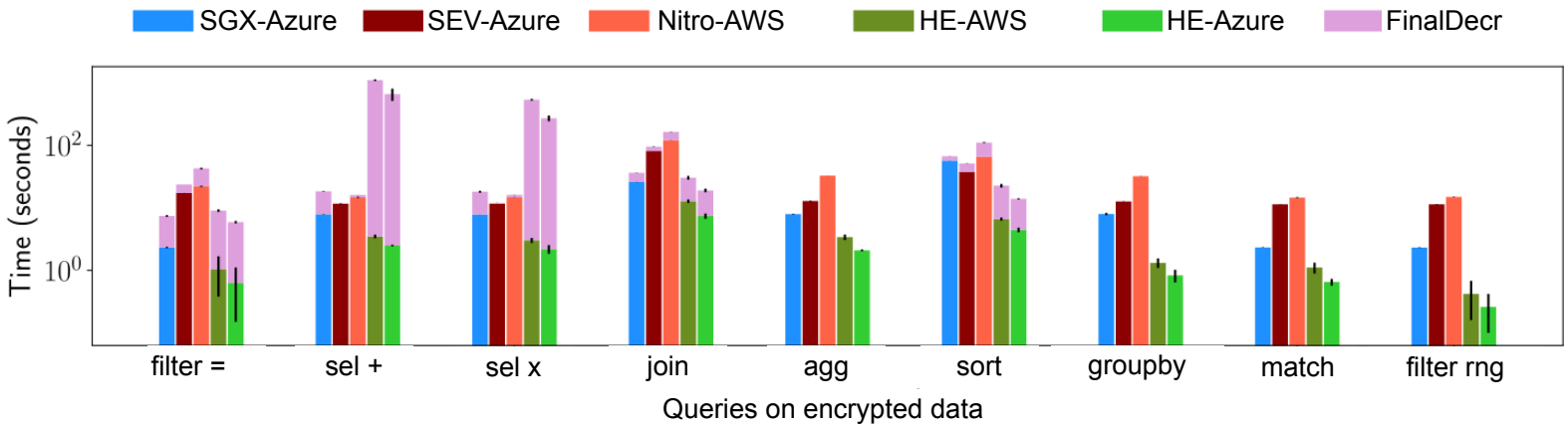
Differences

- Features/Functionalities
- Guarantees
- Attacks/threats

[Van Bulck et al.'17] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx. Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution. USENIX Security 2017.
[Kocher et al.'19] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom: Spectre Attacks: Exploiting Speculative Execution. IEEE S&P 2019.
[Zhang et al.'18] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou: TruSense: Information Leakage from TrustZone. IEEE INFOCOM 2018.
[Li et al.'19] M. Li, Y. Zhang, Z. Lin, and Y. Solihin: Exploiting Unprotected I/O Operations in AMD's Secure Encrypted Virtualization. USENIX Security 2019.
[Li et al.'21] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng: CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channels. USENIX Security 2021.
…

Differences

- Features/Functionalities
- Guarantees
- Attacks/threats
- Performance



Differences

- Features/Functionalities
- Guarantees
- Attacks/threats
- Performance

Differences

- Features/Functionalities
- Guarantees
- Attacks/threats
- Performance
- Deployment

Problem

- How to choose?
- How to reconcile?
- Software solutions, other accelerators
- Step back: requirements?

Confidential analytics

[Savvides et al.'22] S. Savvides, S. Kumar, J. J. Stephen, P. Eugster: C3PO. Cloud-based Confidentiality-preserving Continuous Query Processing. ACM Transactions on Privacy and Security 25(1) 2022.

[Savvides et al.'20] S. Savvides, D. Khandelwal, P. Eugster. Efficient Confidentiality-Preserving Data Analytics over Symmetrically Encrypted Datasets. VLDB 2020.

[Eugster et al.'19] P. Eugster, S. Kumar, S. Savvides, J. J. Stephen. Ensuring Confidentiality in the Cloud of Things. IEEE Pervasive Computing 18(1) 2019.

[Eugster et al.'18] P. Eugster, G. A. Marson, B. Poettering. A Cryptographic Look at Multi-party Channels. IEEE CSF 2018.

[Savvides et al.'17] S. Savvides, J. J. Stephen, M. Saeida Ardekani, V. Sundaram, P. Eugster. Secure Data Types: A Simple Abstraction for Confidentiality-Preserving Data Analytics. ACM SoCC 2017.

[Hauck et al.'16] M. Hauck, S. Savvides, P. Eugster, M. Mezini, G. Salvaneschi. SecureScala: Scala Embedding of Secure Computations. ACM SCALA 2016.

[Stephen et al.'16] J. J. Stephen, S. Savvides, V. Sundaram, M. Said Ardekani, P. Eugster. STYX: Stream Processing with Trustworthy Cloud-based Execution. ACM SoCC 2016.

[Hsu et al.'16] T. C.-H. Hsu, K. Hoffman, P. Eugster, M. Payer. Enforcing Least Privilege Memory Views for Multithreaded Applications. ACM CCS 2016.

[Stephen et al.'14a] J. J. Stephen, S. Savvides, R. Seidel, P. Eugster. Practical Confidentiality Preserving Big Data Analysis. USENIX HotCloud 2014.

[Stephen et al.'14b] J. J. Stephen, S. Savvides, R. Seidel, P. Eugster. Program Analysis for Secure Big Data Processing. ACM/IEEE ASE 2014.

[Stephen et al.'13] J. J. Stephen and P. Eugster. Assured Cloud-Based Data Analysis with ClusterBFT. ACM/IFIP Middleware 2013.

SITE requirements

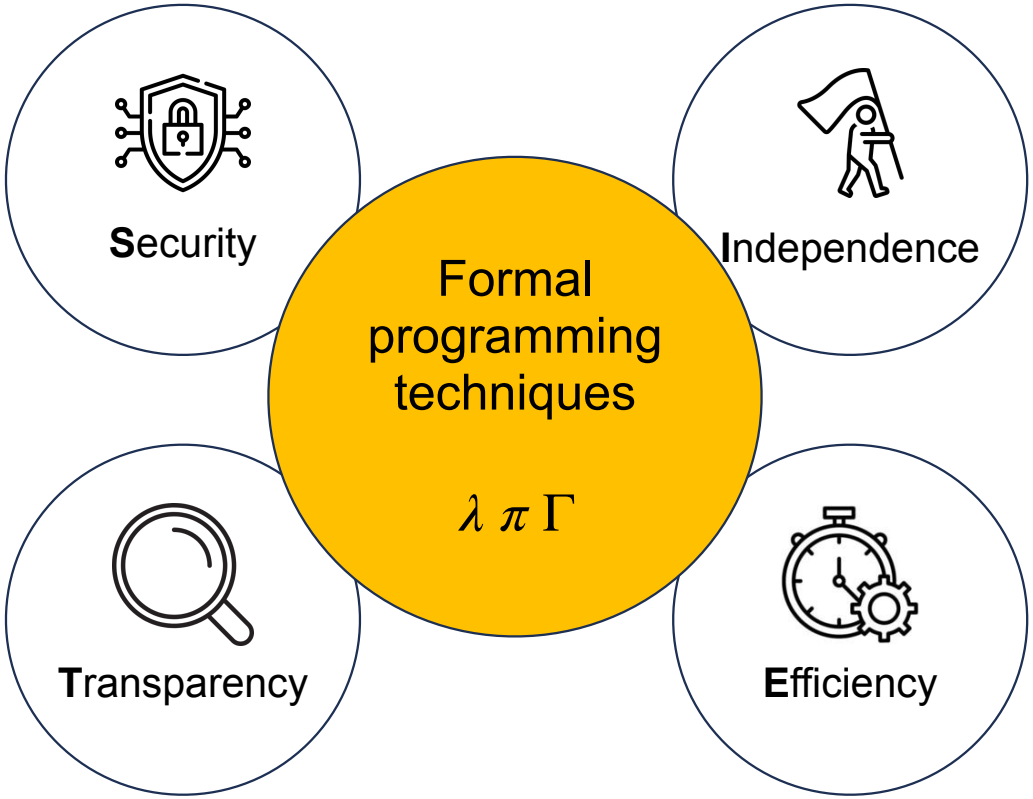
SITE requirements

SITE requirements

SITE requirements

SITE requirements

SITE requirements



Hydra [Mangipudi et al.'23]

Confidentiality guarantee based on *noninterference* [Goguen&Meseguer'82]
Enforced by static information flow *type system*

Extended multi-level security policy *mapping mechanisms to levels*
Formal query language with lambdas, relations, security abstractions

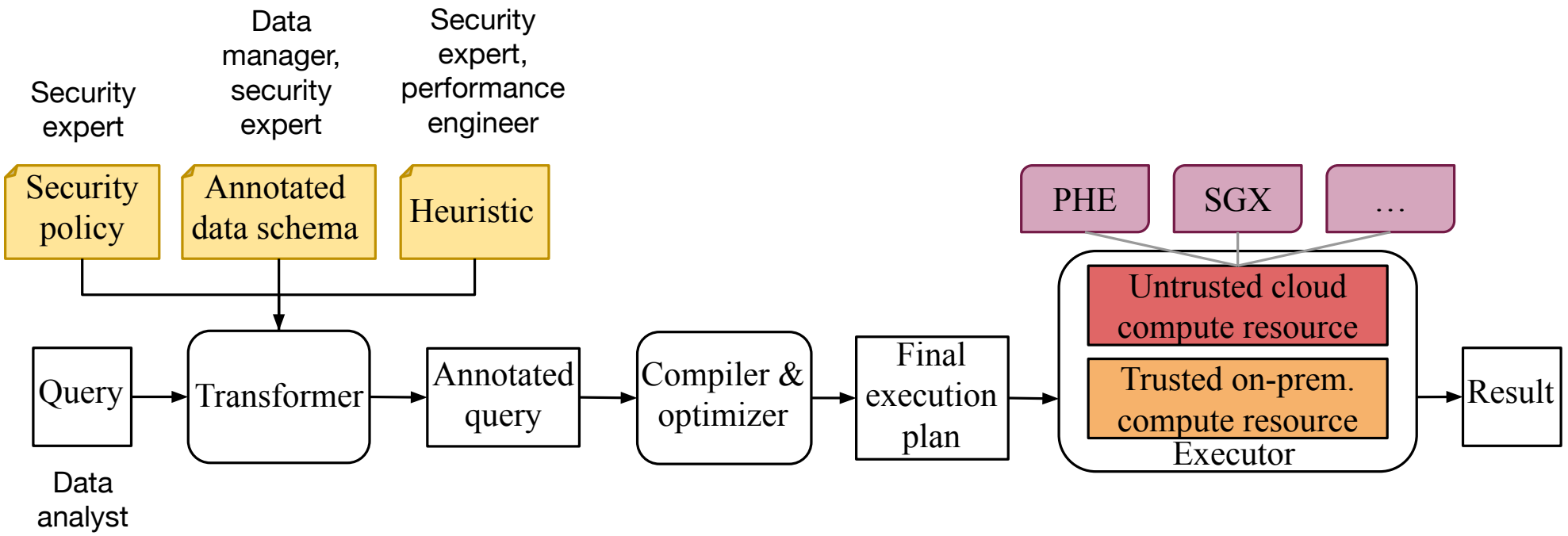
Automated transformation of queries to use security mechanisms
Semantics preservation proven

Formal language supports *reasoning about performance*
Domain-specific language for pluggable heuristics

[Mangipudi et al.'23] S. Mangipudi, P. Chuprikov, P. Eugster, M. Viering, and S. Savvides. Generalized Policy-Based Noninterference for Efficient Confidentiality-Preservation. ACM PLDI'23.

[Goguen&Meseguer'82] A. Goguen and J. Meseguer: Security Policies and Security Models. IEEE S&P 1982.

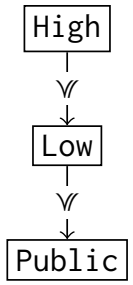
Hydra workflow



Example security policy

All steps performed once only

1. Define secrecy levels/labels



2. Assign mechanisms to levels

Label $\mathcal{L}$	Domain $\mathcal{D}$	Scheme $\mathcal{S}_\varnothing$
High	CLNT, SGX	$\varnothing$
High	CLD	AES-GCM
Low	CLD	SWP, AES-ECB, Paillier, ElGamal, OPE

E.g., data at level `High` can be handled only on client (CLNT), by Intel SGX (SGX), or in plain cloud only encrypted with AES-GCM

3. Label data with levels

Relation	Field	Type	Label	Security type (inferred)
Customers	custId	Str	Low	$(\text{Str}^{\text{AES-ECB}}, \text{Low})$
	bal	Dbl	High	$(\text{Dbl}^{\text{AES-GCM}}, \text{High})$
	…			
Orders	orderId	Str	High	$(\text{Str}^{\text{AES-GCM}}, \text{High})$
	custKey	Str	Low	$(\text{Str}^{\text{AES-ECB}}, \text{Low})$
	price	Str	High	$(\text{Dbl}^{\text{AES-GCM}}, \text{High})$
	date	Int	Low	$(\text{Int}^{\text{OPE}}, \text{Low})$
	…			

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

Transformation

1. Original security-agnostic query

```
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)
```

2. Automatically transformed query with security annotations — imagine data analysts having to write this

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

APIs (Scylla ≈ Hydra)

Integrating mechanisms

```
trait Provider { val name: String }
object Inet extends Provider { .. = "Inet" }
object Client extends Provider { .. = "Client" }
sealed trait Mechanism { val name: String }
sealed trait TEE extends Mechanism
trait NativeTEE extends TEE
  def initialize(..)
  def setKeyInTEE(..)
  def changeEncryptionScheme(..)
  def project(..)
  ..
trait VirtualTEE extends TEE
trait Scheme extends Mechanism
  def encrypt(input: Long): Bytes
  def decrypt(input: Bytes): Long
sealed trait PHEOp
trait AddPHEOp extends PHEOp
  def add(x: Bytes, y: Bytes): Bytes
trait AddPtxtPHEOp extends PHEOp
  def addPtxt(x: Bytes, y: Long): Bytes
```

Writing heuristics

```
trait Heuristic
  // p ≈ LogicalPlan: has only Free Vars
  def apply(p: ScyllaPlan): ScyllaPlan
type ScyllaStep =
  PartialFunction[ScyllaPlan, ScyllaPlan]
type ScyllaRule = ScyllaPlan => ScyllaPlan
class RuleHeuristic(rs: ScyllaRule*)
  extends Heuristic {..}
trait EqScyllaStep extends ScyllaStep
  def safeApply(p: ScyllaPlan): ScyllaPlan
  override def isDefinedAt(p: ..): Boolean
  final def apply(p: ScyllaPlan) =
    {/* checks p ⇝ safeApply(p) [65, Fig. 7] */}
```

Example heuristics

Using rule-based DSL

```
cldWithSGXR: ScyllaRule = _.transformUp(
  check(!_.provider.isFree)
  andThen minCostOf(
    rule(_.setP(AWS).setT(None))
      andThen setSchemesMinCost(),
    rule(_.setP(AWS).setT(SGX))
      andThen setSchemesMinCost()
  )
)
cldWithSGX = RuleHeuristic(cldWithSGXR)
```

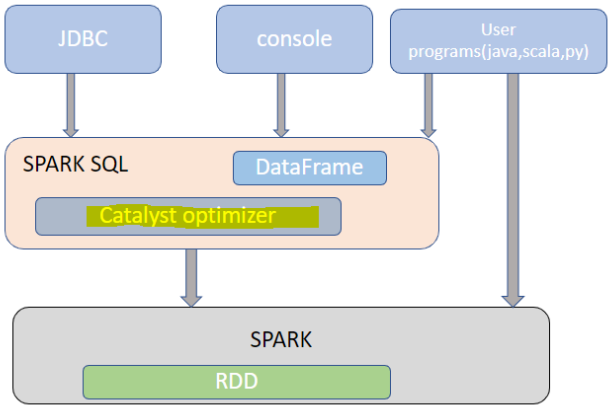
↑ PHE or SGX, whichever is expected to be faster (and possible for PHE)

PHE as long as faster and possible, then SGX [Mangipudi et al.'23] →

```
cldR: ScyllaRule = _.transformUp(
  check(_.provider.isFree
    && _.children.all(_.provider == AWS)
  ) andThen rule(_.setP(AWS).setT(None))
    andThen setSchemesMinCost()
)
restSGXR: ScyllaRule = _.transformUp(
  check(_.provider.isFree)
    andThen rule(_.setP(AWS).setT(SGX))
    andThen setSchemesMinCost()
)
restClientR: ScyllaRule = _.transformUp(
  check(_.provider.isFree)
    andThen rule(_.setP(Client).setT(None))
    andThen setSchemesMinCost()
)
// all three Hydra heuristics
hydraSGX = RuleHeuristic(restSGXR)
hydraPHE = RuleHeuristic(cldR, restClientR)
hydraHybrid = RuleHeuristic(cldR, restSGXR)
```

Implementation

- Built on Apache Spark
  - Used by data analysts through original APIs (SparkSQL)
- Leveraging *Catalyst* extensible query optimizer
  - 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)

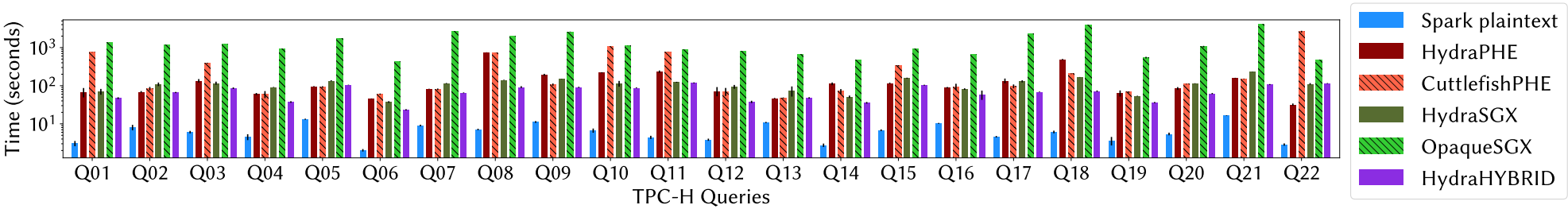


Current status

- Tested and evaluated in 3 major clouds (AWS, Azure, GCP)
- Supports
  - 4 hardware mechanisms (TEEs)
  - 6 software mechanisms (PHE schemes) + 2 own [Savvides et al.’20]

Evaluation

- TPC-H benchmarks, avg 5 runs, labeling from Cuttlefish [Savvides et al.'17]

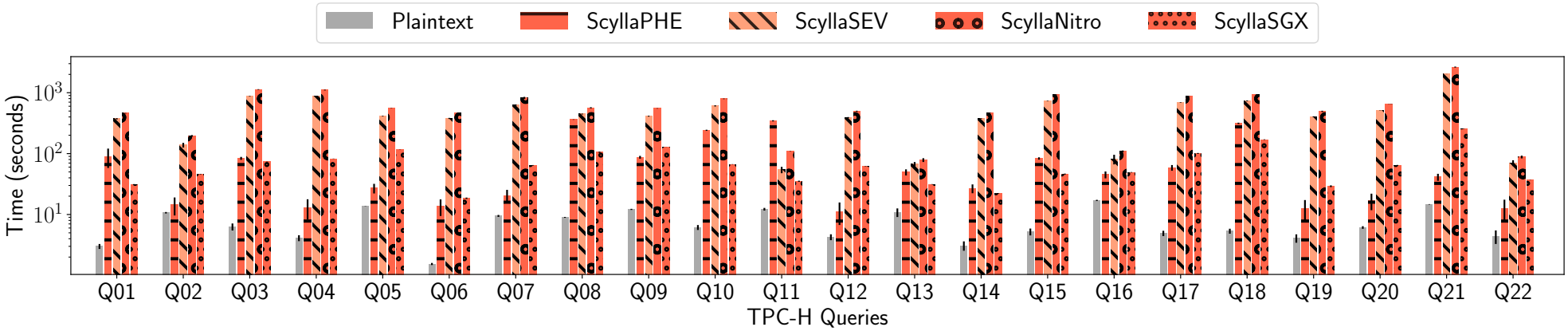


- Hydra (PHE/SGX) 1.6x/11.2x faster than Cuttlefish/Opaque [Zheng et al.'17]
- Hydra (Hybrid) 1.7x/1.6x faster than Hydra PHE/SGX
  - 2.7x/17.9x faster than Cuttlefish/Opaque

[Zheng et al.'17] W. Zheng, A. Dave, J. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica: Opaque: An Oblivious and Encrypted Distributed Analytics Platform. USENIX NSDI'17. (ORAM disabled for comparison.)

Latest numbers

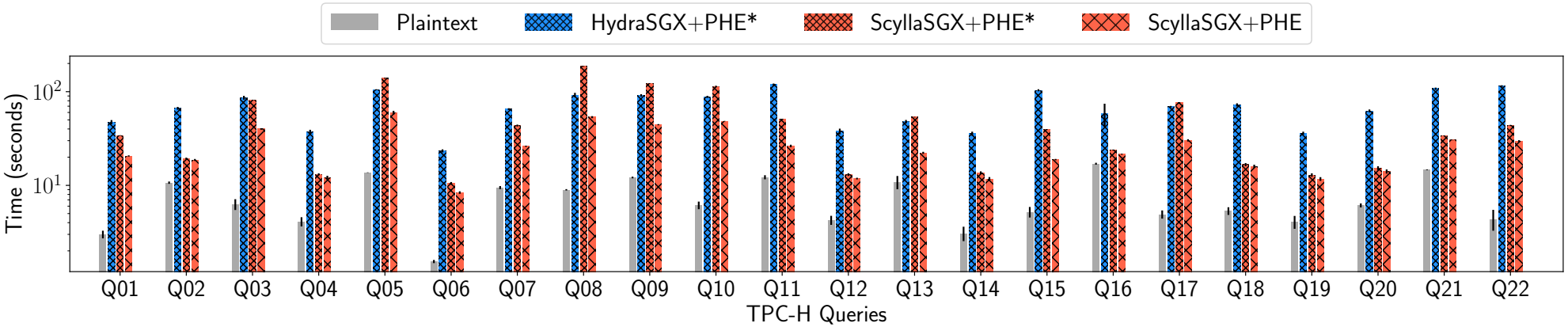
Different TEEs



- ScyllaPHE 1.31× faster than ScyllaSGX, which is 6.09× faster than ScyllaSEV, which in turn is 1.31× faster than ScyllaNitro
- SEV ~ Nitro, but slower than SGX
  - Likely due to encryption/decryption exiting/entering the enclaves happening in JVM in the former vs native

Latest numbers

Different heuristics



- SGX+PHE heuristic vs old simpler SGX+PHE* used in Hydra
- ScyllaSGX+PHE 1.59× faster than ScyllaSGX+PHE*
  - 2.87× faster than HydraSGX+PHE*

Outlook

- Mechanized proving of guarantees
- Support for alternative mechanisms
- Extensions and refinements galore, e.g., integrity, differential privacy
- Application to alternate (data flow) models, e.g., multi-party, streams