STYX: Stream Processing with Trustworthy Cloud-based Execution

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

October 6, 2016

Purdue University

Table of contents

Overview

Introduction

Compute clouds

- Data analytics platforms
- Cost-efficiency, 'on-demand' compute, low infrastructure setup cost

IoT

- 26 billion smart devices connected to a network by 2020
- Fine-grained user behavior tracking to capture, personalize and/or monetize user experience

Stream processing

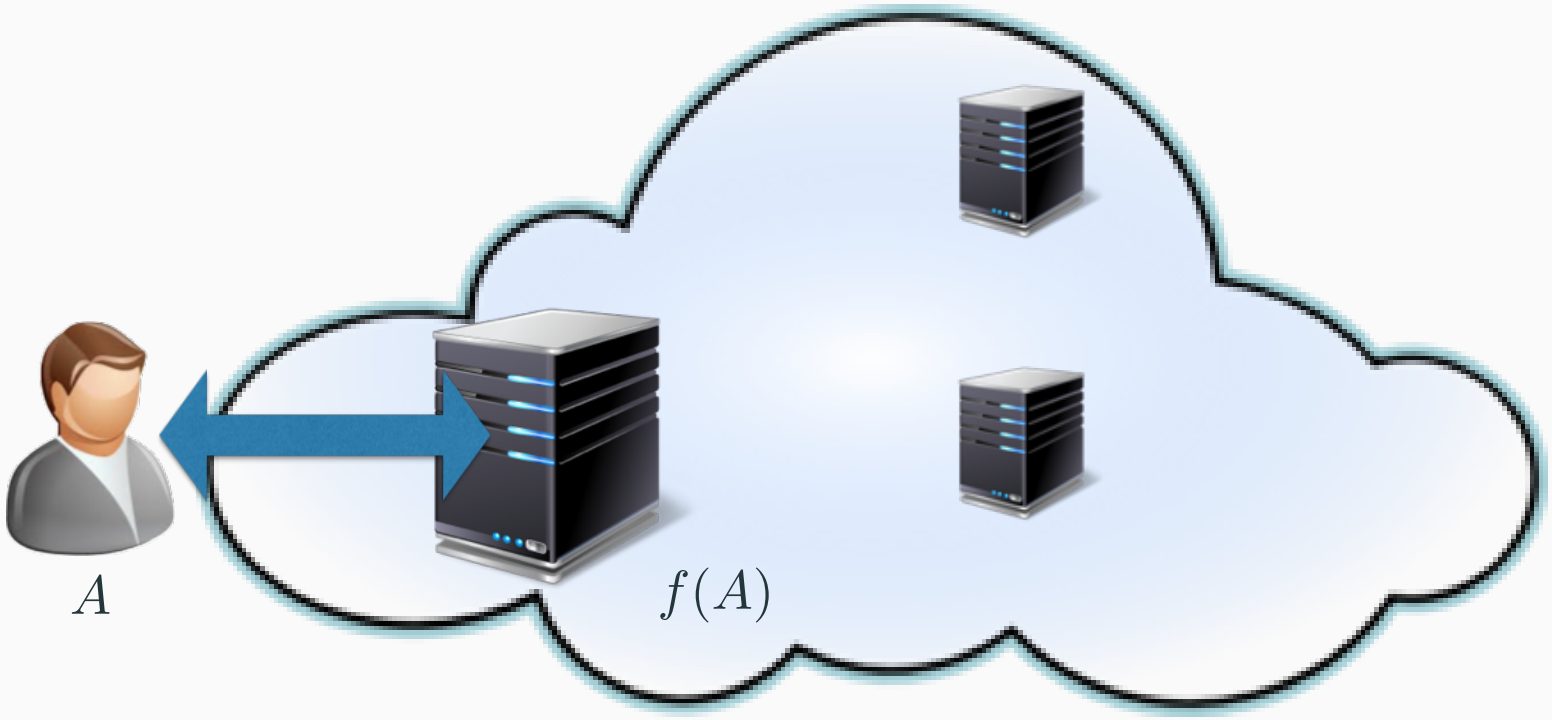
- Analytics on real-time streaming data (continuous queries)
- Many systems over the last few years - Apache Storm, Apache Spark, Apache Flink, Apache Samza, Amazon Kinesis

Vulnerabilities - 1

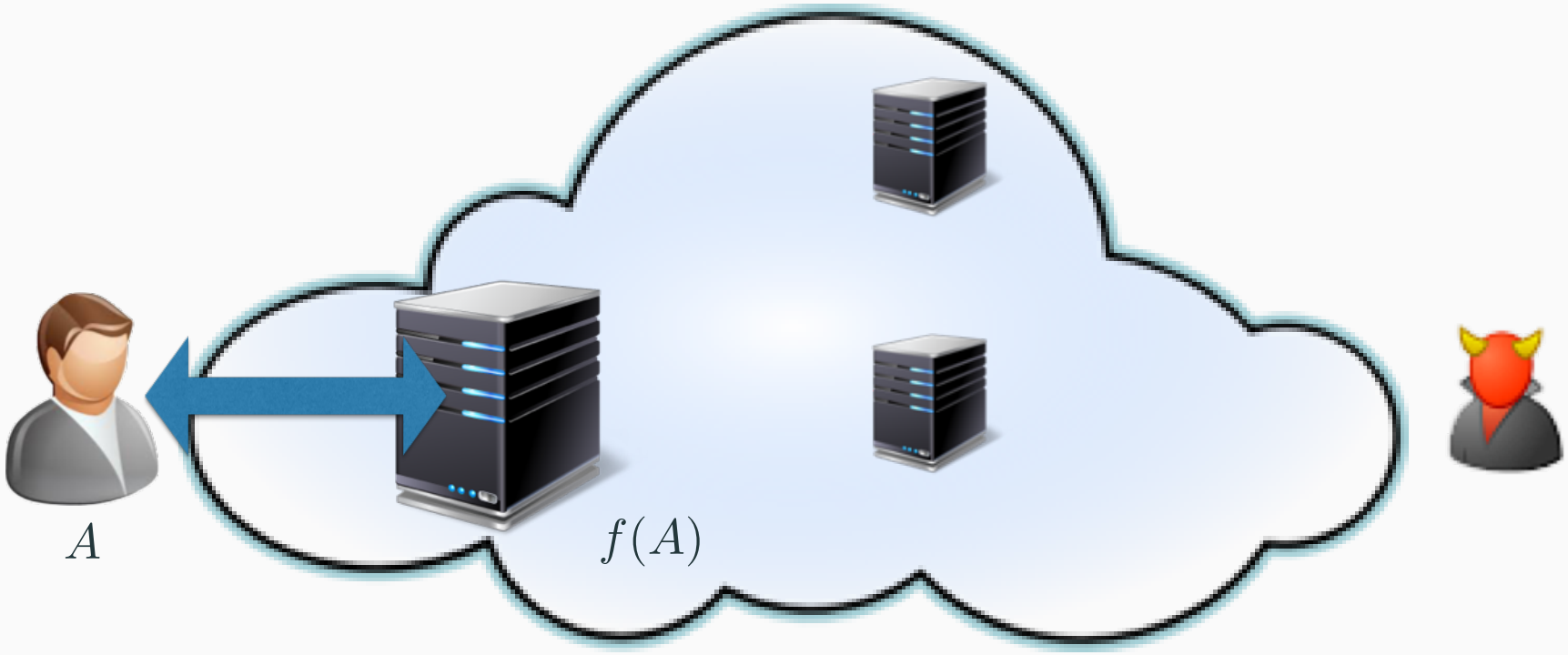
Vulnerabilities - 1

Vulnerabilities - 1

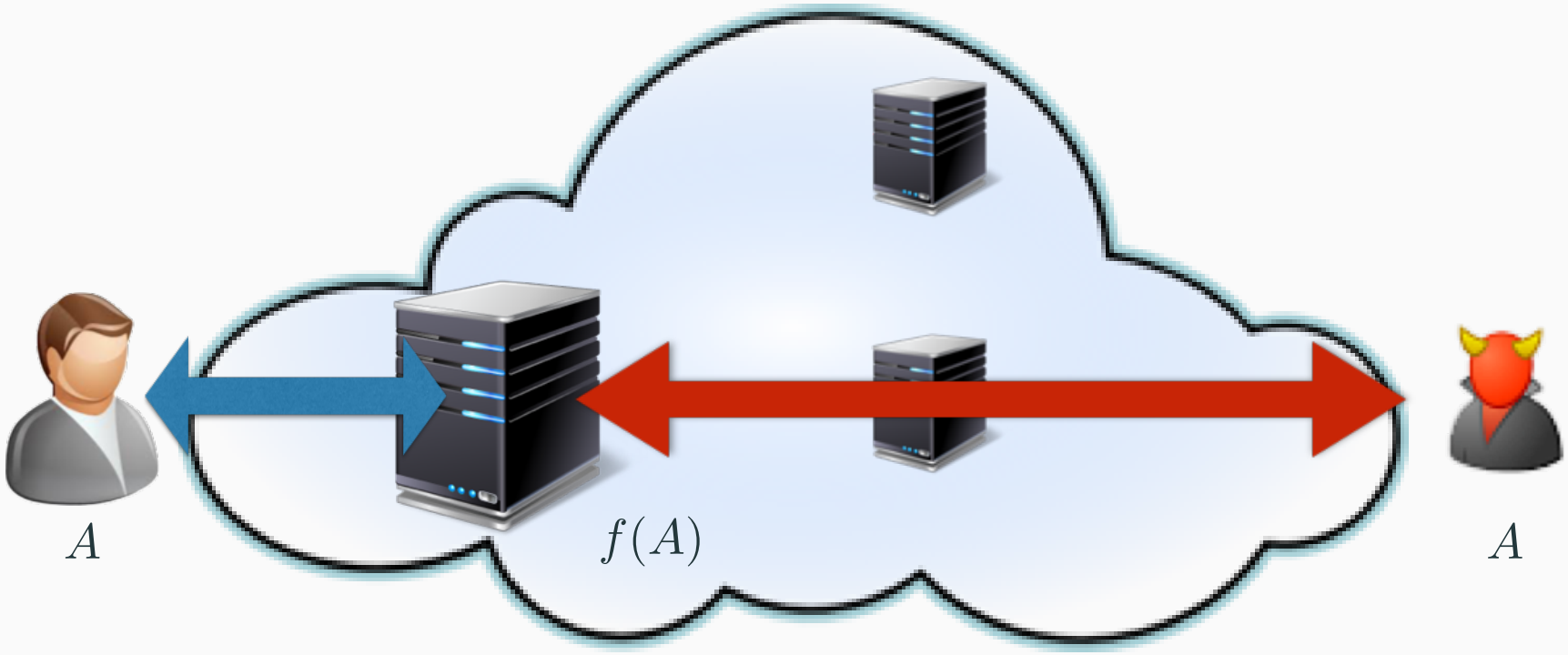
Vulnerabilities - 1



Vulnerabilities - 1



Vulnerabilities - 1



Vulnerabilities - 2

Real problems

```
                Xen Security Advisory CVE-2014-7188 / XSA-108

IMPACT
======

A buggy or malicious HVM guest can crash the host or read data
relating to other guests or the hypervisor itself.


Public release.
```

Ensuring confidentiality in the cloud

Confidentiality in the cloud

Fully homomorphic encryption (FHE)

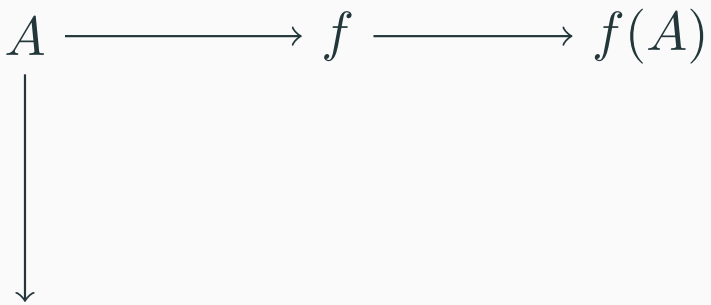
- Allows arbitrary computation on encrypted data

$$A \longrightarrow f \longrightarrow f(A)$$

Confidentiality in the cloud

Fully homomorphic encryption (FHE)

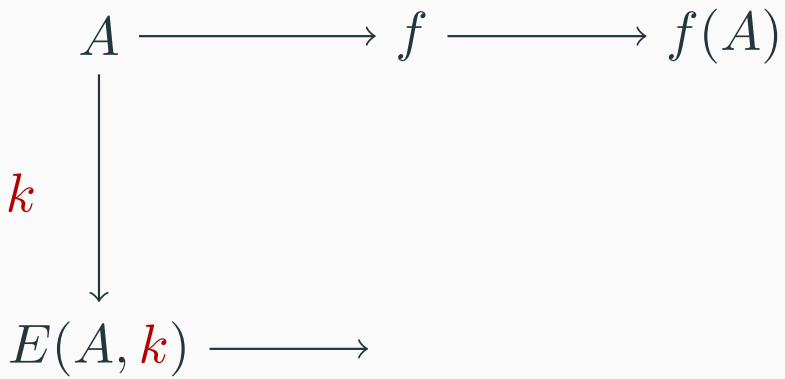
- Allows arbitrary computation on encrypted data



Confidentiality in the cloud

Fully homomorphic encryption (FHE)

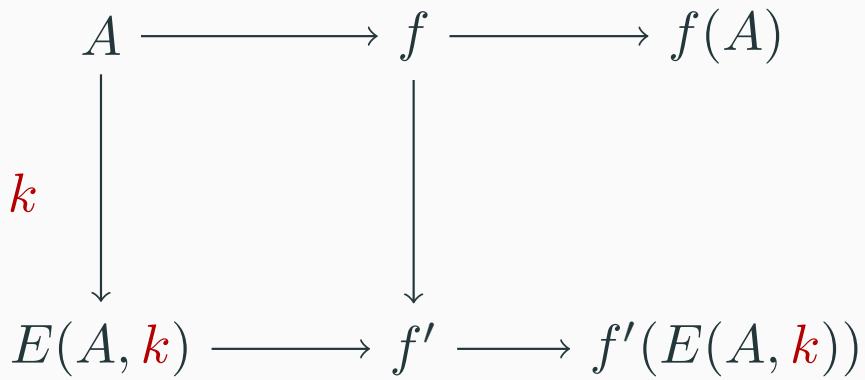
- Allows arbitrary computation on encrypted data



Confidentiality in the cloud

Fully homomorphic encryption (FHE)

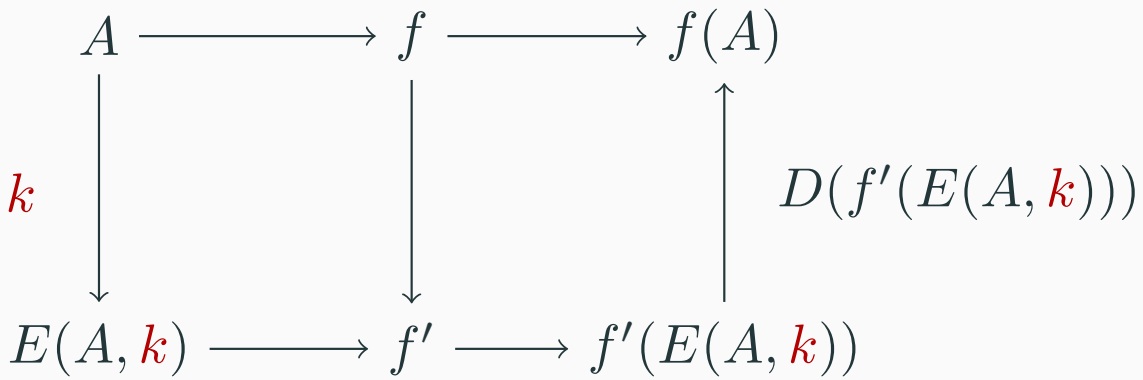
- Allows arbitrary computation on encrypted data



Confidentiality in the cloud

Fully homomorphic encryption (FHE)

- Allows arbitrary computation on encrypted data



Confidentiality in the cloud

Fully homomorphic encryption (FHE)

- Allows arbitrary computation on encrypted data
- Prohibitive overhead

Confidentiality in the cloud

Fully homomorphic encryption (FHE)

- Allows arbitrary computation on encrypted data
- Prohibitive overhead

Partially homomorphic encryption (PHE)

- Allows certain operations to be performed over encrypted text
- AHE: $D(E(x1)\psi E(x2)) = x1 + x2$
- AHE, MHE, OPE, DET

Conjecture

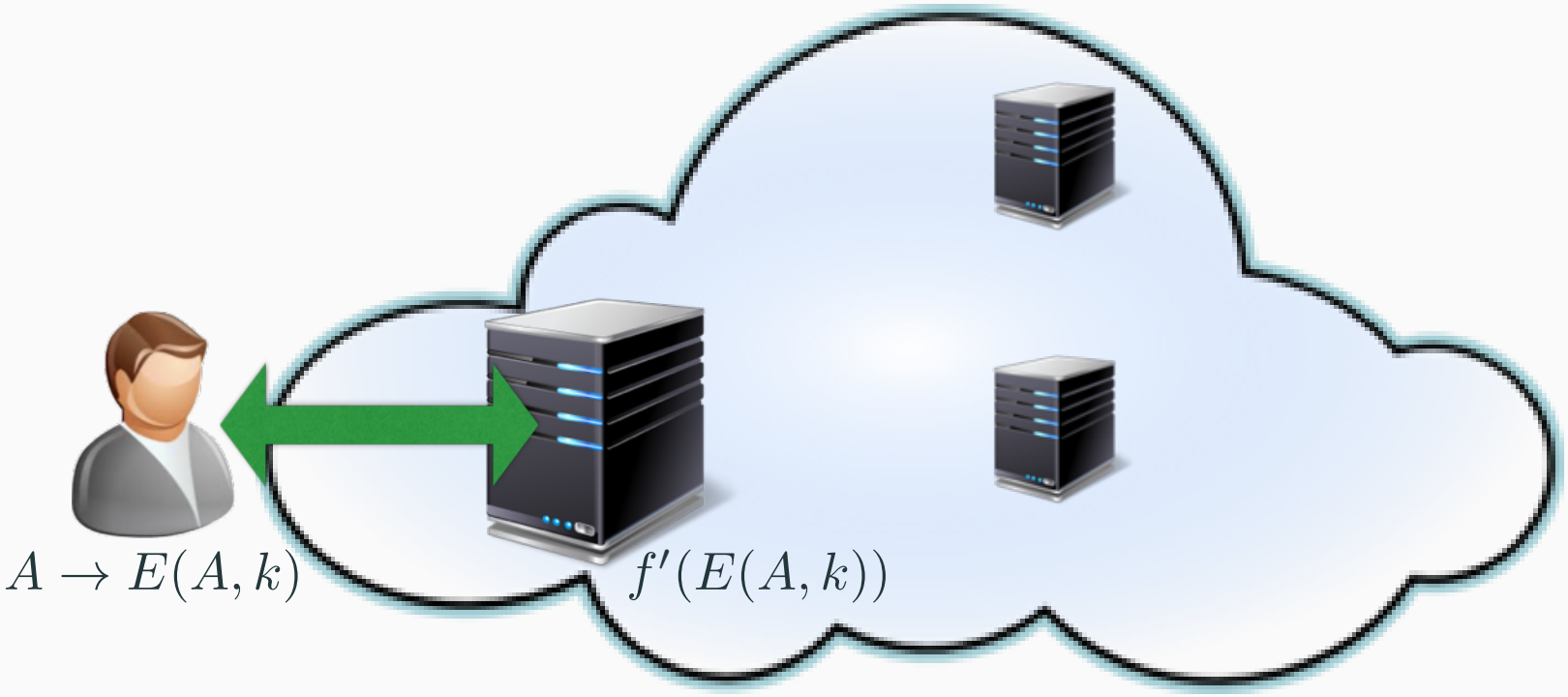
Many data analytics jobs can be performed securely using a combination of partially homomorphic encryption schemes

Vulnerabilities - 3

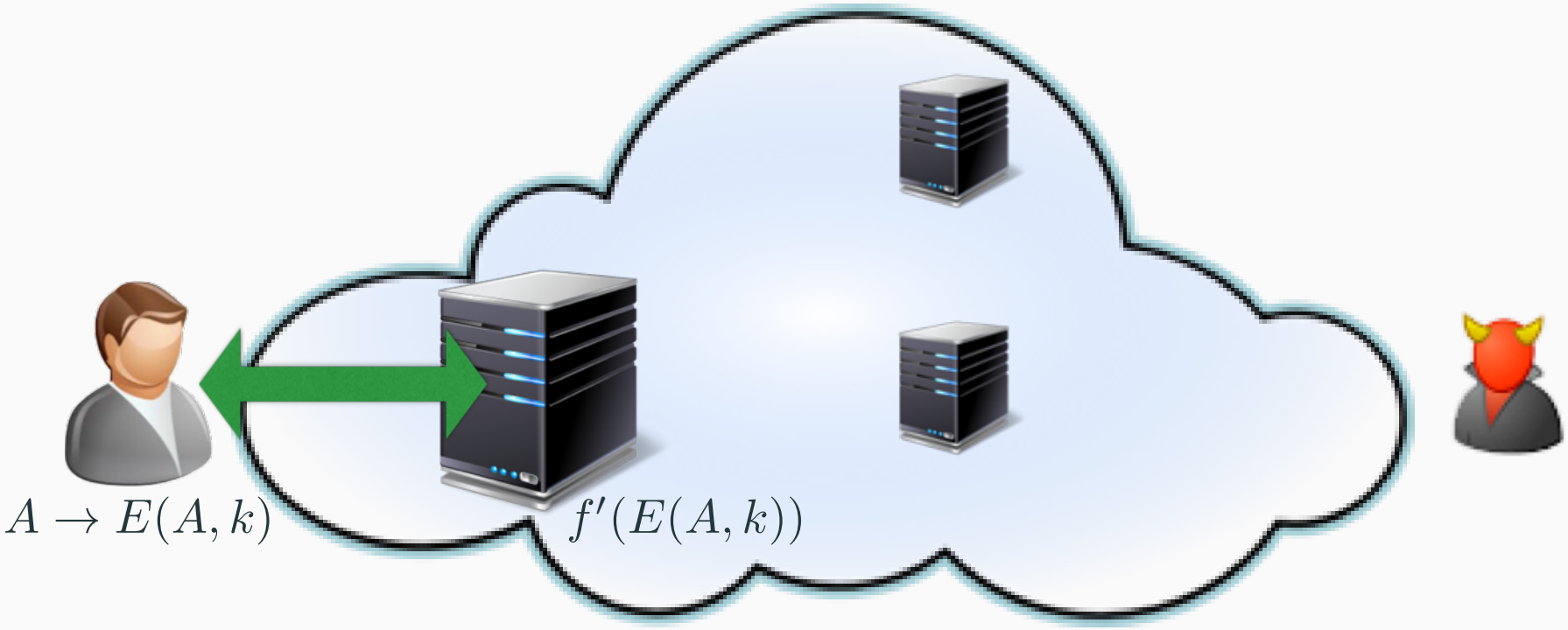
Vulnerabilities - 3

$A \rightarrow E(A, k)$

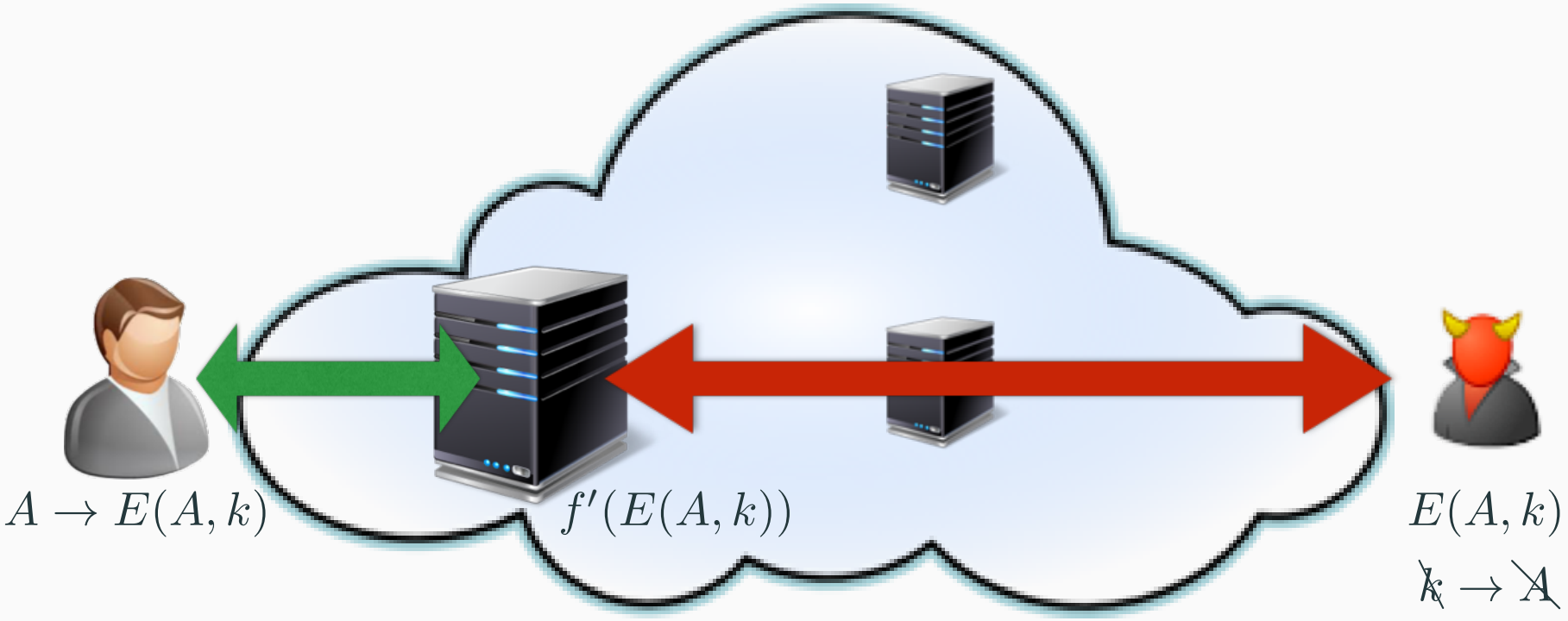
Vulnerabilities - 3



Vulnerabilities - 3



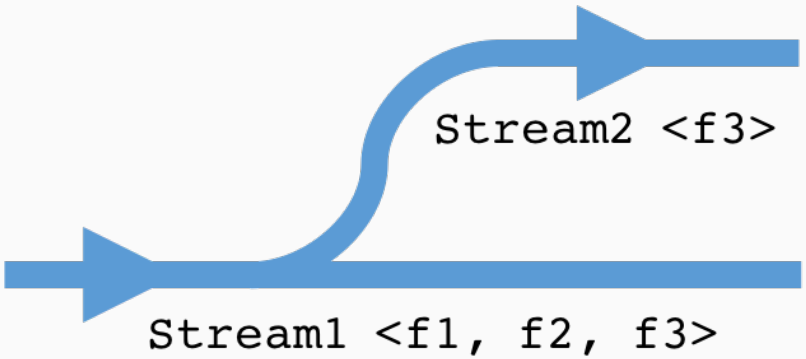
Vulnerabilities - 3



Challenges in encrypted stream processing

Challenges in encrypted stream processing

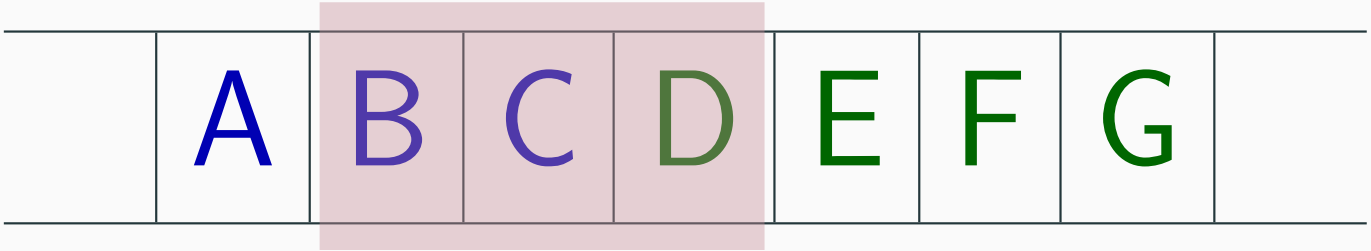
- Programmer effort
  - Need to identify encryption scheme for each input data stream, perform cryptographic equivalent of required operation



- `if (Stream1.f1 < 100) return; else ...`
- `sum = sum + Stream2.f3;`

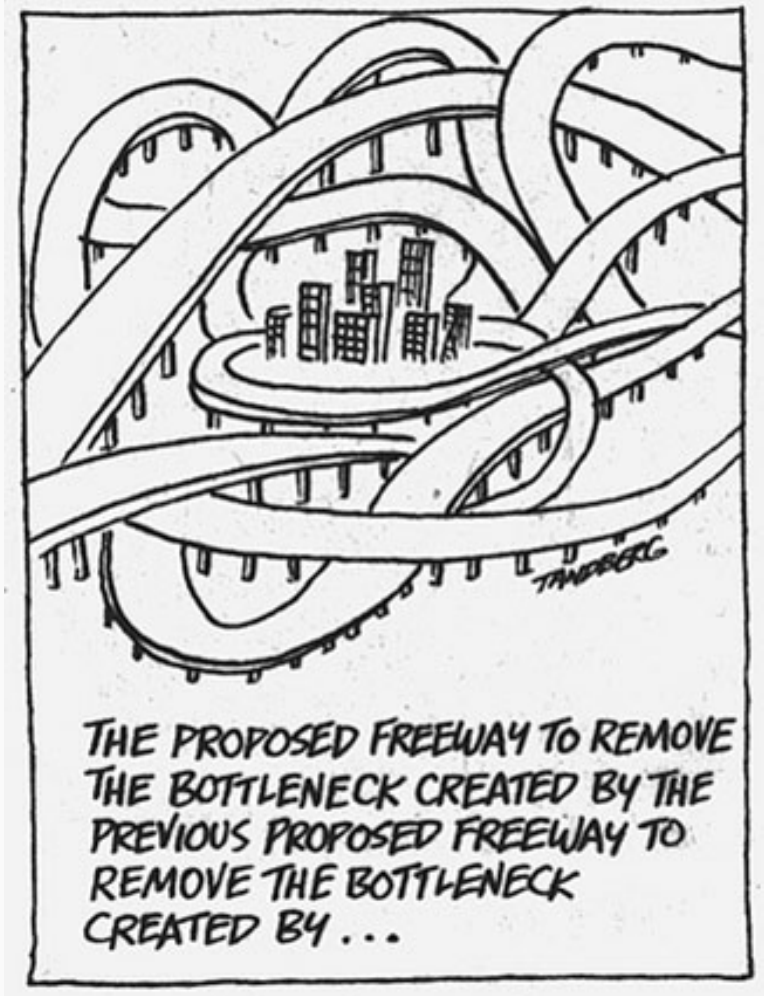
Challenges in encrypted stream processing

- Key change
  - PHE requires all tuples in an aggregate function to be encrypted with same key



Challenges in encrypted stream processing

- Deployment optimizations
  - Deployment parameters specified for plaintext data may not be optimal when computation happens on encrypted data



Challenges in encrypted stream processing

- Programmer effort
  - Need to identify encryption scheme for each input data stream, perform cryptographic equivalent of required operation
- Key change
  - PHE requires all tuples in an aggregate function to be encrypted with same key
- Deployment optimizations
  - Deployment parameters specified for plaintext data may not be optimal when computation happens on encrypted data

Challenges in encrypted stream processing

- Programmer effort
  - Need to identify encryption scheme for each input data stream, perform cryptographic equivalent of required operation
- Key change
  - PHE requires all tuples in an aggregate function to be encrypted with same key
- Deployment optimizations
  - Deployment parameters specified for plaintext data may not be optimal when computation happens on encrypted data
- Limitations of PHE
  - PHE may not support a sequence of operations requiring trusted nodes to perform remaining computation

Challenges in encrypted stream processing

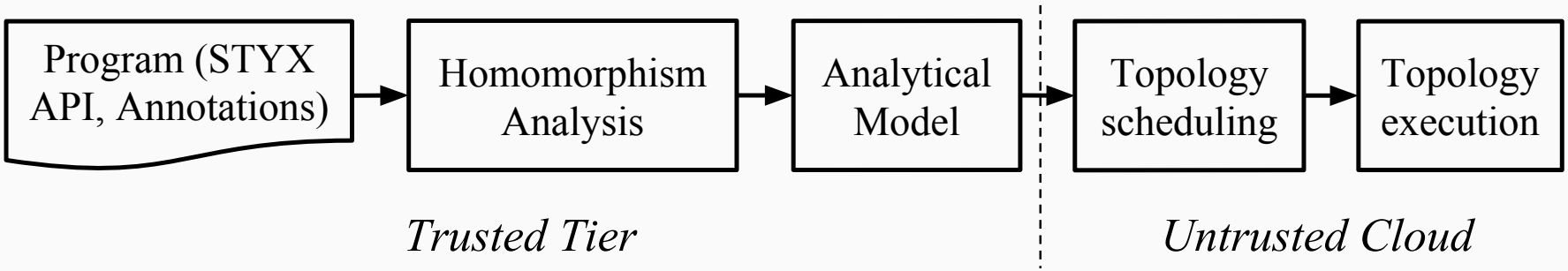
- Programmer effort
  - Need to identify encryption scheme for each input data stream, perform cryptographic equivalent of required operation
- Key change
  - PHE requires all tuples in an aggregate function to be encrypted with same key
- Deployment optimizations
  - Deployment parameters specified for plaintext data may not be optimal when computation happens on encrypted data
- Limitations of PHE
  - PHE may not support a sequence of operations requiring trusted nodes to perform remaining computation
- Constants and initialization
  - Variables must be initialized using the encrypted value of the initialization constant

Challenges in encrypted stream processing

- Programmer effort
  - Need to identify encryption scheme for each input data stream, perform cryptographic equivalent of required operation
- Key change
  - PHE requires all tuples in an aggregate function to be encrypted with same key
- Deployment optimizations
  - Deployment parameters specified for plaintext data may not be optimal when computation happens on encrypted data
- Limitations of PHE
  - PHE may not support a sequence of operations requiring trusted nodes to perform remaining computation
- Constants and initialization
  - Variables must be initialized using the encrypted value of the initialization constant

Architecture

STYX architecture



Execution flow

- User submits program written using system (STYX) API
- Homomorphism analysis identifies crypto systems required to execute the graph
- Analytical model identifies deployment profile
- Scheduler assigns tasks to nodes
- Runtime executes tasks

STYX abstractions and key update

STYX abstraction

Group sum in a sliding window

```
1  /** Track sum of values per group per time slot */
2  public class SlotBasedSum<T>  {
3      ...
4      public void updateSum(T group, int slot,
                             SecField val) {
5          SecField[] sums = objGroupSum.get(group);
6          if (sums == null) {
7              sums = new SecField[this.numSlots];
8              init(sums, val);
9              objGroupSum.put(obj, sums);
10         }
11         sums[slot] = SecureOper
12             .add(sums[slot], val);
13     }
14 }
```

STYX abstraction

Group sum in a sliding window

```
1  /** Track sum of values per group per time slot */
2  public class SlotBasedSum<T>  {
3      ...
4      public void updateSum(T group, int slot,
                             SecField val) {
5          SecField[] sums = objGroupSum.get(group);
6          if (sums == null) {
7              sums = new SecField[this.numSlots];
8              init(sums, val);
9              objGroupSum.put(obj, sums);
10         }
11         sums[slot] = SecureOper
12             .add(sums[slot], val);
13     }
14 }
```

Without STYX abstractions

Group sum in a sliding window (Storm)

```
1  public class SlotBasedSum<T>  {
2      BigInteger publicKey = readPubKey();
3      public void updateSum(T group, int slot,
                             BigInteger value) {
4          BigInteger[] sums = objGroupSum.get(group);
5          if (sums == null) {
6              sums = new BigInteger[this.numSlots];
7              init(sums, "AHE");
8              objGroupSum.put(group, sums);
9          }
10         sums[slot] = sums[slot].multiply(value)
11             .mod(publicKey.multiply(publicKey));
12     }
13 }
14
```

Without STYX abstractions

Group sum in a sliding window (Storm)

```
1  public class SlotBasedSum<T>  {
2      BigInteger publicKey = readPubKey();
3      public void updateSum(T group, int slot,
                             BigInteger value) {
4          BigInteger[] sums = objGroupSum.get(group);
5          if (sums == null) {
6              sums = new BigInteger[this.numSlots];
7              init(sums, "AHE");
8              objGroupSum.put(group, sums);
9          }
10         sums[slot] = sums[slot].multiply(value)
11             .mod(publicKey.multiply(publicKey));
12     }
13 }
14
```

Without STYX abstractions

Group sum in a sliding window (Storm)

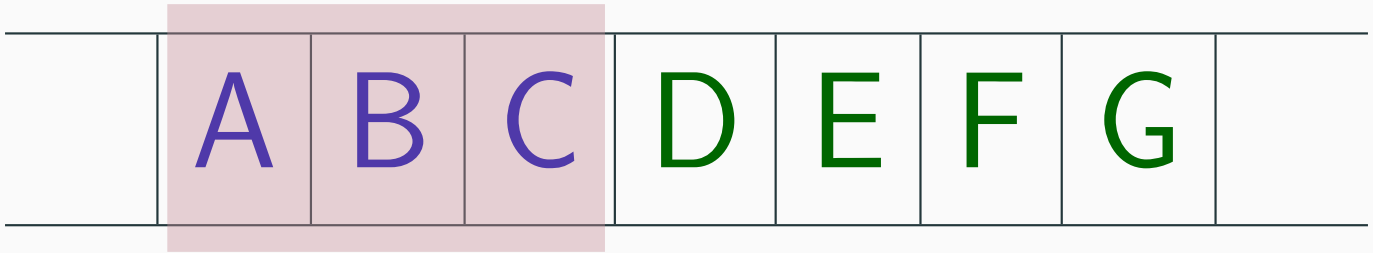
```
1  public class SlotBasedSum<T>  {
2      BigInteger publicKey = readPubKey();
3      public void updateSum(T group, int slot,
                             BigInteger value) {
4          BigInteger[] sums = objGroupSum.get(group);
5          if (sums == null) {
6              sums = new BigInteger[this.numSlots];
7              init(sums, "AHE");
8              objGroupSum.put(group, sums);
9          }
10         sums[slot] = sums[slot].multiply(value)
11             .mod(publicKey.multiply(publicKey));
12     }
13 }
14
```

Key change

Challenges

- Functions that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Problem

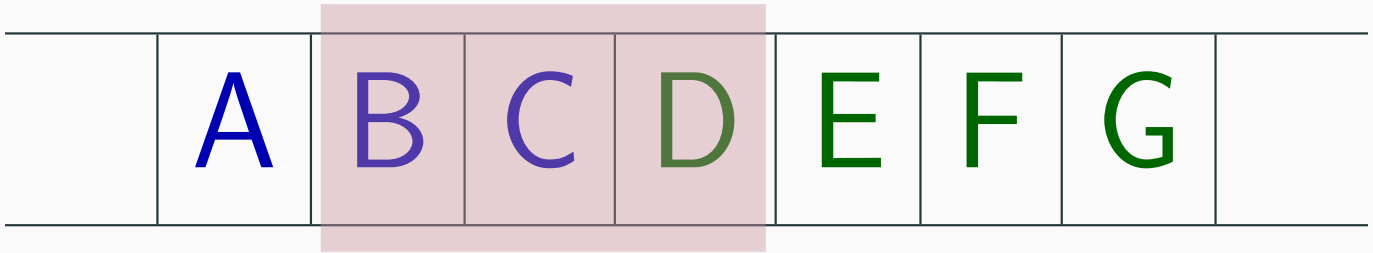


Key change

Challenges

- Functions that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Problem

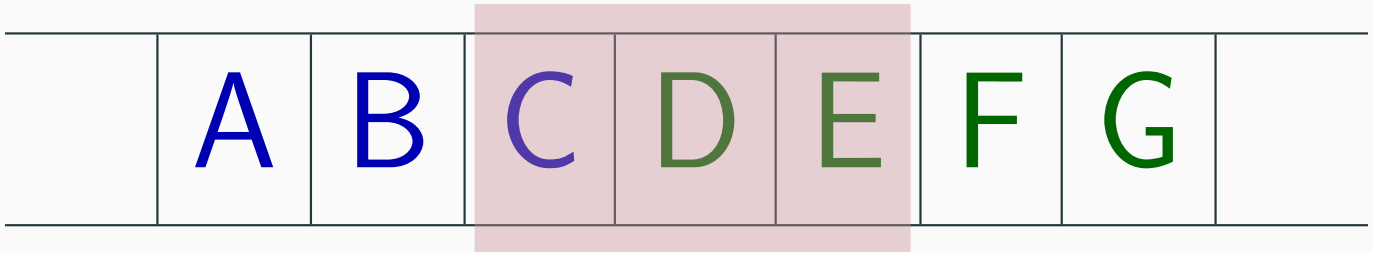


Key change

Challenges

- Functions that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Problem

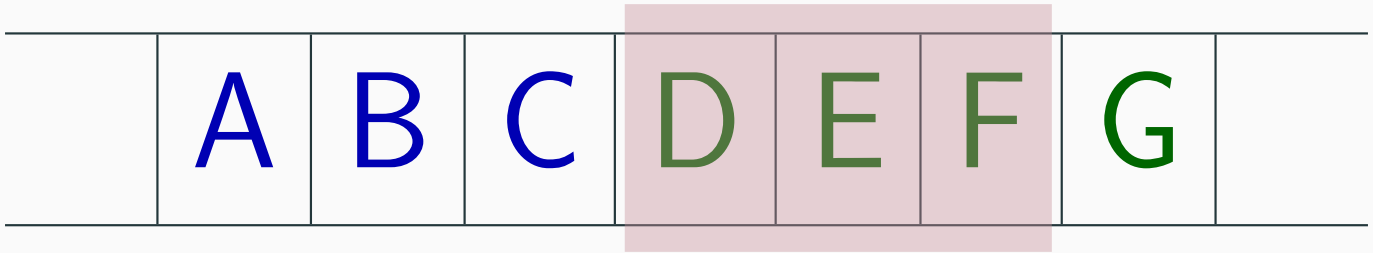


Key change

Challenges

- Functions that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Problem

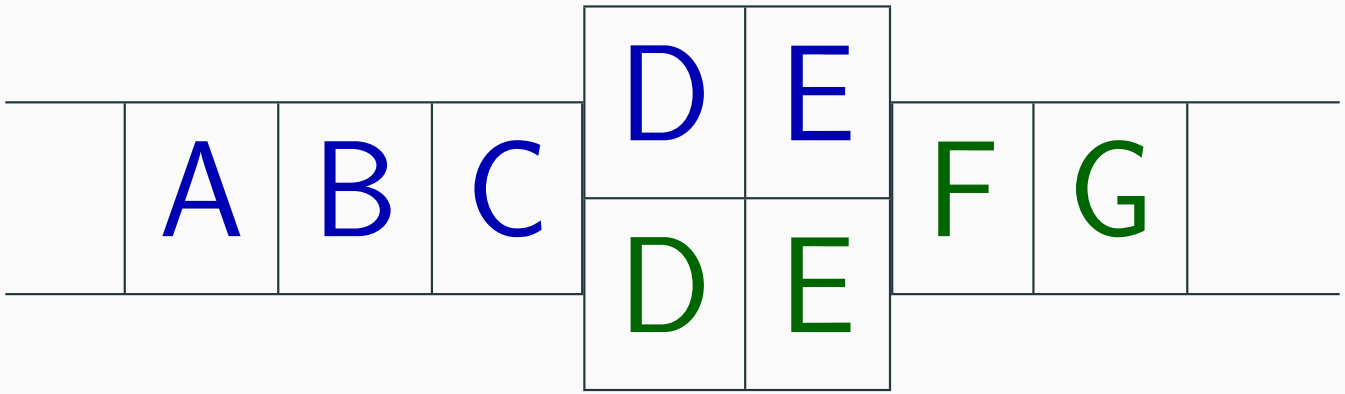


Key change

Challenges

- Continuous queries that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Solution

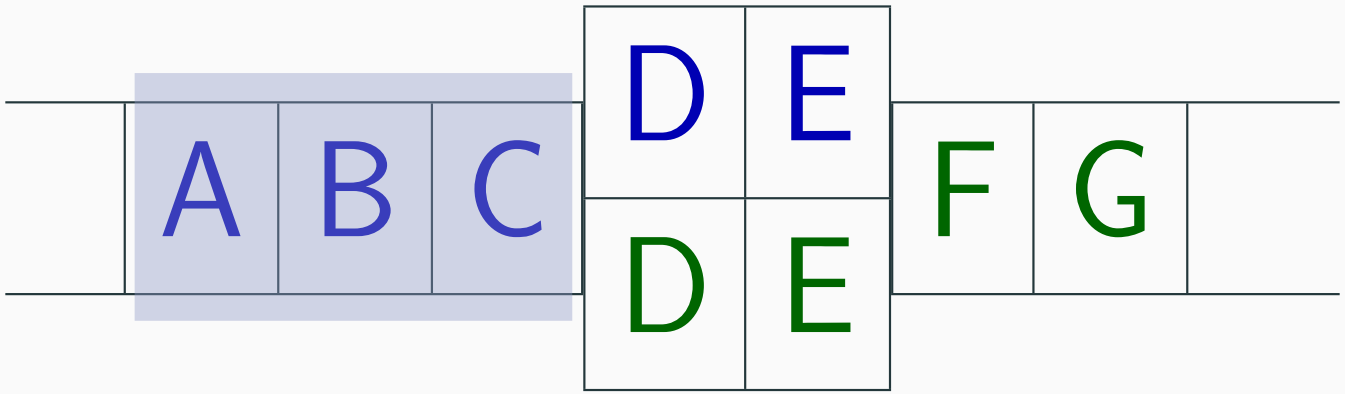


Key change

Challenges

- Continuous queries that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Solution

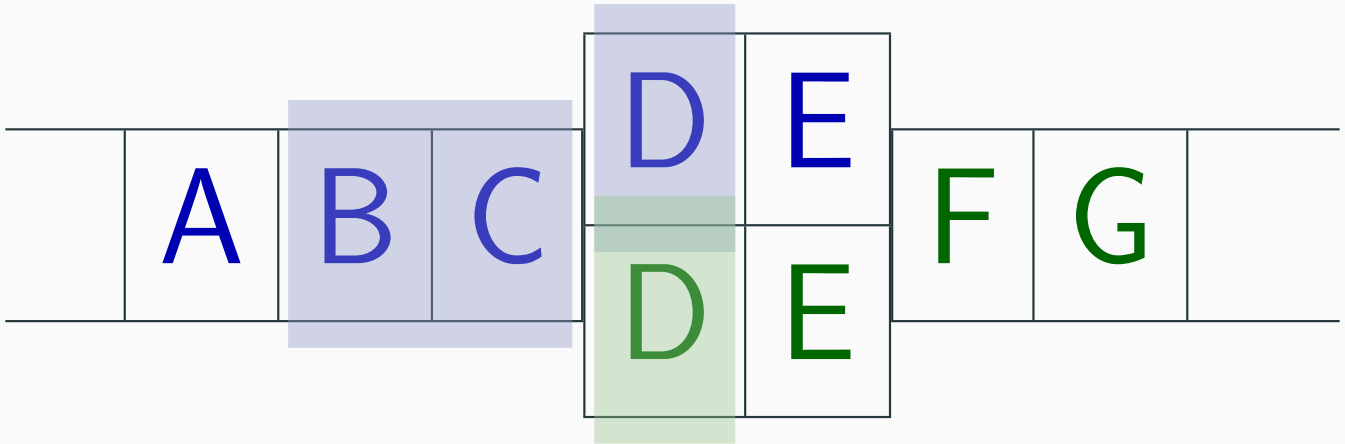


Key change

Challenges

- Continuous queries that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Solution

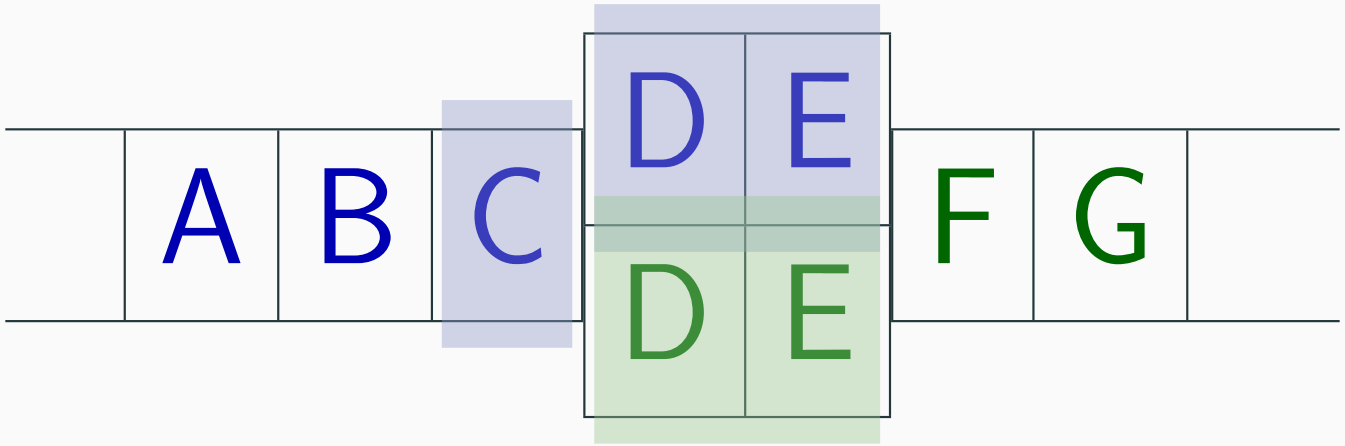


Key change

Challenges

- Continuous queries that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Solution

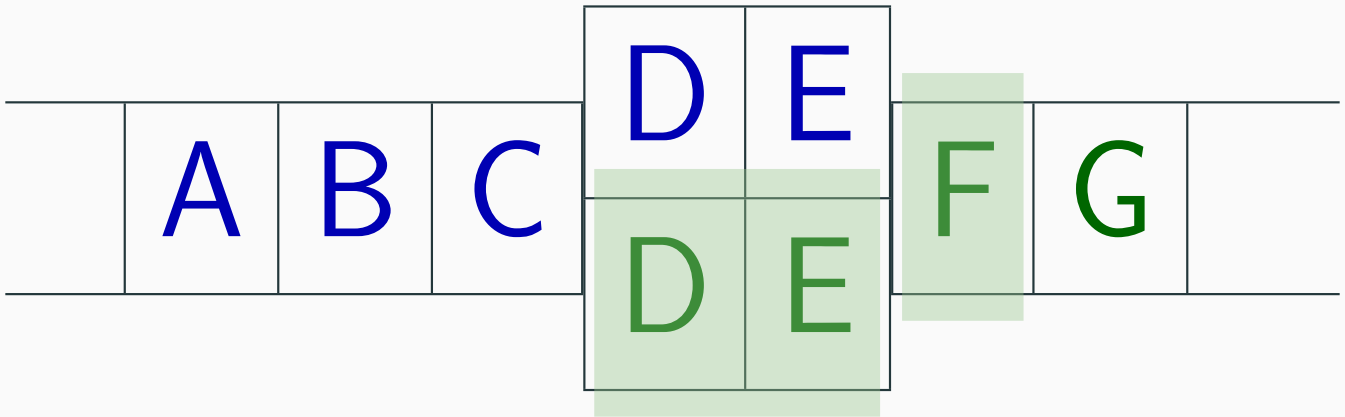


Key change

Challenges

- Continuous queries that aggregates data over a sliding window makes it impossible to change the encryption key without disrupting output

Solution

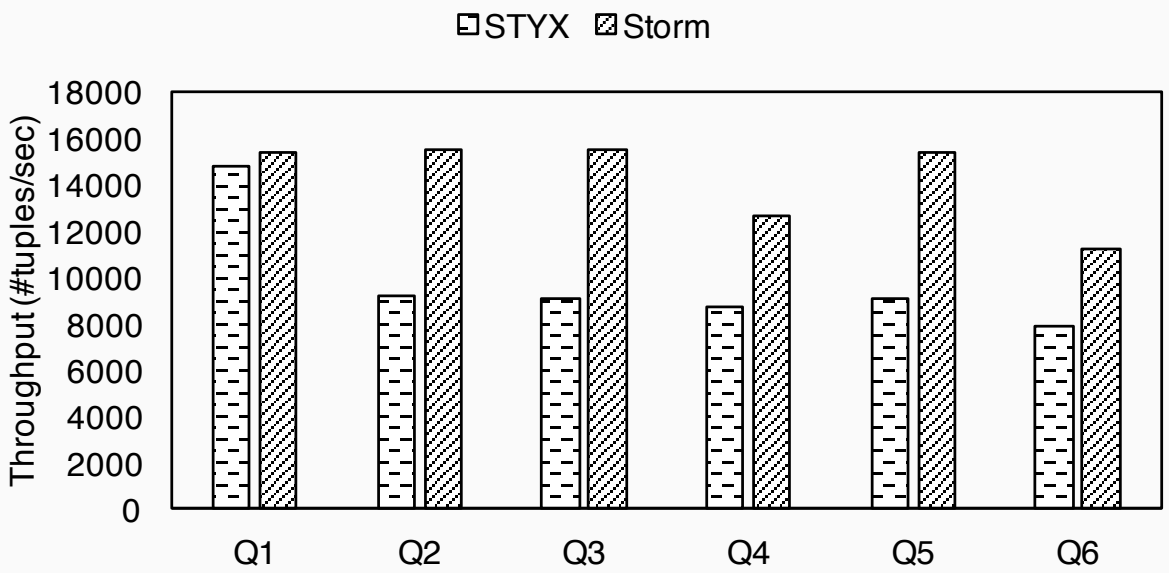


Evaluation

Evaluation - 1

IoT Bench

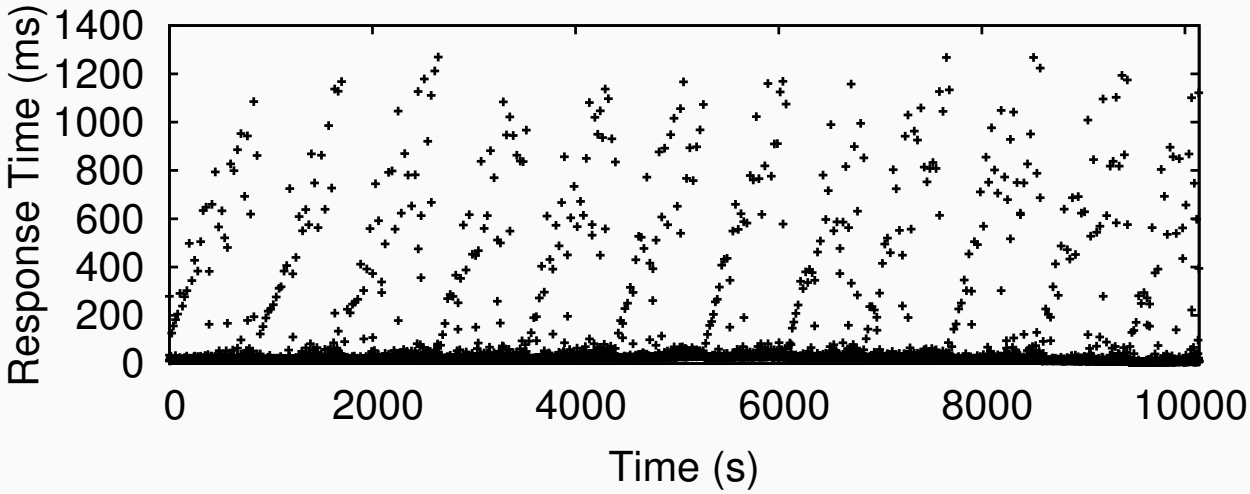
- Smart meter data over a 24 hour time period at the rate of 1 reading per minute from 443 unique homes, totaling 637526 records
- 4 ec2 m3.large node
- Each tuple comprises of timestamp, meter id and meter reading



Evaluation - 2

Performance when keys change

- New york taxi route data (10G)
- Application finds the top 10 most frequent routes during the last 30 minutes of taxi servicing
- 9 ec2 m3.large node



Related work and conclusion

Related work

Prior work on encrypted computaion

- [Popa et al.; SOSP'11];
- [Gentry.; STOC'09];
- [Stephen et al.; VLDB'13];

Other approaches

- Using secure processors (e.g., Intel SGX)

Conclusion

- Confidentiality breaches pose a serious threat to adoption and utilization of cloud resources
- PHE has proven to be effective for various batch workloads
- STYX leverage PHE for stream data analytics in the cloud
- Makes it easier for programmers to use PHE
- Automatically translates the program and optimizes deployment parameters

Questions