CryptStream: Cryptographic Access Controls for Streaming Data

Cory Thoma
corythoma@cs.pitt.edu

Adam J. Lee
adamlee@cs.pitt.edu
University of Pittsburgh
Pittsburgh, Pennsylvania
15260

Alexandros Labrinidis
labrinid@cs.pitt.edu

ABSTRACT

With data becoming available in larger quantities and at higher rates, new data processing paradigms have been proposed to handle large and fast data. Data Stream Processing is one such paradigm wherein transient data flows as streams through sets of continuous queries, only returning results when data is of interest to the querier, allowing uninteresting data to be ignored. To process these data streams, users are employing third party computational platforms or large private platforms to reduce the individual cost for querying and computing over data streams. Utilizing third parties for outsourcing computation means data being processed is available to the third party as well, which could violate the data provider's privacy. There has been research done into access control for streaming data, and these works provide a good first step, but fall short of a complete system. In this paper, we introduce CryptStream for cryptographically enforcing access controls over streaming data. CryptStream combines data providers access control policies with ones prescribed by the data consumer and the server as well. We show that CryptStream improves over earlier work in the same design space while providing smaller overheads and more flexibility.

1. INTRODUCTION

The proliferation of small powerful computing devises has led to an increase in the availability of data. Whether it be a smart phone, smart watch, sensor, or large computing clusters; data has become more available at faster speeds and greater volumes. Traditional data management paradigms lack the ability to maintain accurate and quick response to queries being executed over faster data. To combat this problem, the data streaming paradigm has been proposed as a way to efficiently and effectively manage large amounts of quick data, or data that is time sensitive. DSMSs separate the data *source* from the data *consumer* so that control no longer resides with either. As a result, control over what entities have access to data resides with the server employed to handle queries. To bring control back to the data source, we propose *CryptStream*, a method for cryptographically enforcing access control over streaming systems. CryptStream allows a data source to describe an access control policy over their streaming data while also allowing data consumers to maximize the use of third party computing platforms through the use of special encryption schemes.

Work in the same design space as CryptStream provides a good first step. The closest work, StreamForce [1], falls short of providing a complete system. First, it limits the querying power of the data consumer by limiting it to predescribed views. It also incurs large overheads due to the use of Outsourced Attribute Based Encryption for every data tuple. In this paper, we introduce CryptStream as a system that cryptographically enforces access controls over streaming data. CryptStream aims at overcoming limitations of previous works while also providing new techniques for policy enforcement and query processing. Specifically, CryptStream makes the following contributions:

- Unlike previous work, CryptStream allows users to cryptographically enforce access control over streaming data and change their policies in real time. In CryptStream, access control is based upon the data consumer's attributes. To enforce changes in policy, CryptStream utilizes modifications to the concept of Security Punctuations [2] to enforce Attribute Based Access Control (ABAC) polices.
- CryptStream provides a built-in scheme for distributing and managing cryptographic keys based on user attributes. Using Security Punctuations, keys can be distributed via CP-ABE enforcement of ABAC policies so that only the proper data consumers can access data.
- CryptStream can combine access control policies from the data provider, data consumer, and the server while not requiring the access control framework to be the same.
- Finally, CryptStream is a system which allows data consumers to submit almost any type of query. Through the use of special encryption techniques, data consumers can execute almost any type of query so long as they were given permission by the data provider.

CODASPY'15, March 2–4, 2015, San Antonio, Texas, USA.
ACM 978-1-4503-3191-3/15/03.
http://dx.doi.org/10.1145/2699026.2699134

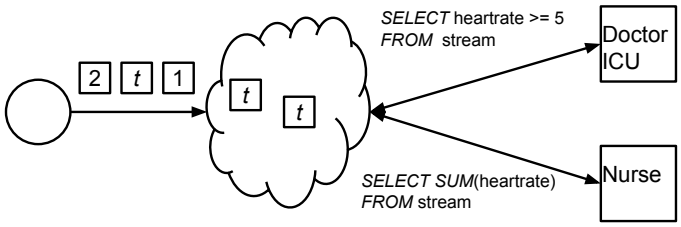

Figure 1: An example of one data provider and two clients accessing that data providers stream.

dsp	$heartrate$	$(doctor \wedge ICU)$	$+$	tsp	d
qsp	$\{heartrate, pid, name\}$	$nurse$	$+$	tsp	i

Figure 2: Two security punctuations illustrating a data provider and a querier punctuation.

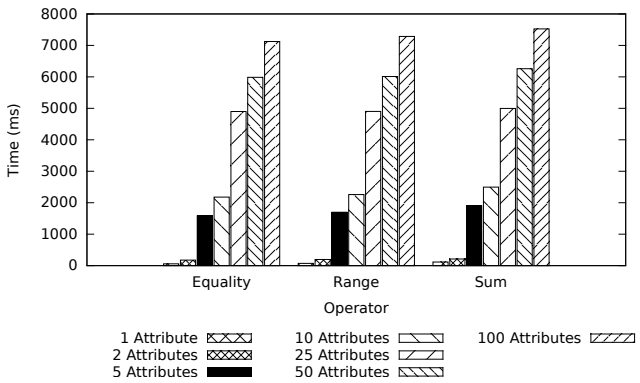

Figure 3: Outsourced ABE decryption for different operators with different numbers of attributes.

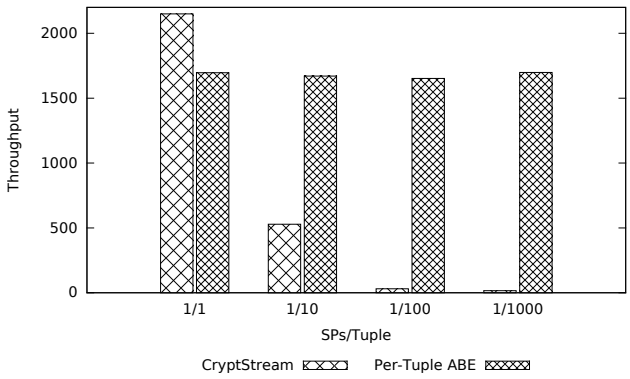

Figure 4: The frequency of Security Punctuation affects the throughput for CryptStream.

2. CRYPTSTREAM

CryptStream has three main components: *data providers*, *clients*, and *compute & route nodes*. Data providers initialize and populate streams of data which they have authored access control policies over. Clients submit queries on data streams and are the subjects for which a data provider will enforce access. The last component, the compute & route node, handles the execution of queries on data streams. The compute & route node is assumed to be untrusted with data provider data in that it should not have access to streams in plain text. Unless a data provider authors an access control policy which grants a compute & route node or a client access, they have no access to a data stream in plain-text. Figure 1 depicts the three components together. The data provider is represented by a circle, the compute & route node is represented by a cloud, and the squares with an ICU doctor and a nurse represent clients. The other components in Figure 1 will be discussed in further sections.

2.1 Background

Each client is associated with a set of attributes which are used by data providers to create *Attribute Based Access Control* (ABAC) policies for enforcing access to their streams. ABAC systems combine attributes through logical "and" and "or" statements and will only grant access if the statement is true based on a clients attributes. In order to transmit data so that the plain-text is not exposed, Attribute Based Encryption (ABE) is used. ABE is a prohibitively expensive operation, so rather than use it directly for each data tuple transmitted, ABE is only used to exchange keys for faster more functional encryption schemes as described below. This reduces the necessity of ABE to use only when creating or updating policies. A new or updated policy is disseminated via *Security Punctuations* (SPs) which are simply a special tuple injected into a data stream by the data provider containing six fields: the type of SP, the data being accessed, the policy, whether the policy is denying or granting access, a timestamp, and either immediate or deferred enforcement. For CryptStream, the type is either a data security punctuation (dsp) if it comes from a data provider, or a querier security punctuation (qsp) if it comes from a client. SPs are represented as numbers squares in Figure 1. Note that SPs are just part of the normal data stream. All data provider policies are ABAC policies enforced via ABE.

The overhead of ABE depends on the number of attributes. Figure 3 shows query types if ABE encryption is used for operating on data tuples. CryptStream avoids these overheads since ABE is only used to transmit policies. The frequency of these transmissions can also affect performance. Figure 4 shows how the frequency of security punctuations affects the throughput of CryptStream. Note that after a 2:1 ratio of data tuples to punctuations, CryptStream outperforms ABE encryption for each data tuple, and improves with less frequent policy updates.

Consider the punctuations in Figure 2. The first punctuation shows the a data security punctuation where the data provider is granting access (the "+") to the field "heartrate" for a doctor who is in the ICU. They are deferring enforcement, meaning the change in policy only affects tuples arriving after the timestamp. The second punctuation shows a querier security punctuation which only grants the role of "nurse" access to the fields "heartrate", "name" and the "pid" patient ID and is immediately enforced, meaning that it also affects a tuples currently in system or operator buffers. Notice that a qsp allows Role Based Access Controls to be used. CryptStream supports arbitrary access control policies for clients so long as the compute & route node supports it. When a data provider generates a new policy, the policy is given to CryptStream which parses it and generates the

Scheme	Type of Queries	Supported operators	Information Gained by Adversary
RND	None	None	Nothing
DET	Equality	Select, Project, Equi-Join, Count, Group By, Order by	Equality of attributes
OPE	Range	Select, Join, Count	A partial to full order of tuples
SUM	Summations	Aggregates over summations	Nothing

Table 1: Summary of what types of queries and operators are supported by each encryption scheme, as well as what each scheme could reveal to a potential adversary.

punctuations. When a data provider generates a data tuple, it is given to CryptStream which handles the encryption of fields.

2.2 Other Components

In CryptStream, compute & route nodes are unchanged since CryptStream sits on the data provider and client acting as an interface into the DSMS. On the client, this interface is tasked with decryption of data provider streams, handling the translation of queries, and gathering keys and accesses from data provider punctuations. When a punctuation is sent to compute & route nodes, any client wanting to access that stream is given the tuple. The CryptStream interface will use the ABE keys provided by a trusted third party to try decrypt the key given in the SP. If successful, the key is stored along with the attributes it is used to encrypt and the punctuation is forgotten.

Another task of the client is to translate queries. When a client wishes to access a stream, it gets the schema and a deterministic key from the data provider via a special tuple. Whenever the client wants to access a field in the tuple, the CryptStream interface will intercept the query and encrypt any field with the deterministic key so that fields can be accessed without the plain-text field name being exposed to the compute & route node. The query is then submitted to be executed. When results are returned to the client, they pass through the CryptStream interface where they are decrypted and then given to the client.

Recall that SPs are only used to transmit keys for more efficient encryption types. In order to allow computation to be done on the server, CryptStream employs four encryption types; deterministic (DET), order preserving (OPE), Homomorphic (HOM), and Random (RND). Each encryption type comes with some guarantee on what the server learns as well as a different level of functionality. These trade-offs are described in table 1. When a data provider issues a new SP, part of the description of data being accessed includes the level of access, which allows the CryptStream interface to determine which queries can run on the compute & route nodes and which need to be processed locally. Figure 5 shows the cost for each operation compared to plain-text operations and a straw-man approach. This straw-man approach simply sends all tuples to the client encrypted encrypted with the RND encryption scheme, meaning that the server gains no information about the tuple, but no functionality is provided.

Figure 1 shows two different queries. The top query requires OPE encryption in order to execute on the compute & route nodes. This means that SP 1 would have given access using the OPE encryption technique and passing the proper

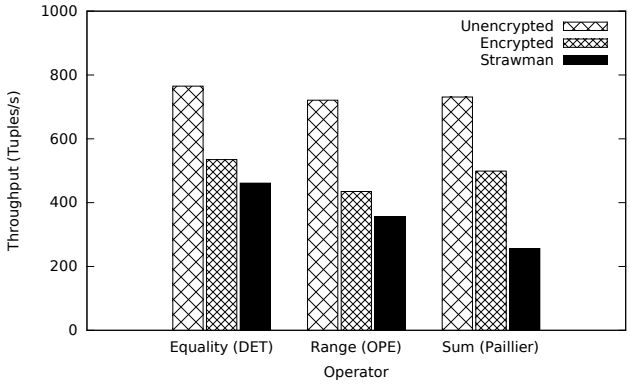

Figure 5: Throughput for each of the different operations supported for both unencrypted and encrypted streams given each encryption type.

key to decrypt them. The second query requires using addition, and therefore requires HOM levels of access. The data provider would have to generate an SP to give HOM permission to the client, but since none was provided, tuples are sent back the client for processing.

3. CONCLUSION

In this paper, we overview CryptStream for cryptographically enforced access controls on streaming data. CryptStream enables a data provider to author Attribute Based Access Control policies on their data streams. Using interfaces on both the data provider and client, CryptStream allows multiple policies to be enforced at the same time from any number of clients, data providers, and compute & route nodes. Through the use of different encryption techniques, CryptStream allows computation to be done on the compute & route node while maintaining some guarantee of confidentiality. Finally, CryptStream does not rely on third parties or off-line key distribution. Using Security Punctuations, CryptStream distributes and manages cryptographic keys online.

4. REFERENCES

[1] D. T. T. Anh and A. Datta. Streamforce: outsourcing access control enforcement for stream data to the clouds. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pages 13–24. ACM, 2014.

[2] R. Nehme, E. A. Rundensteiner, and E. Bertino. A security punctuation framework for enforcing access control on streaming data. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 406–415. IEEE, 2008.