View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Purdue E-Pubs

Purdue University

Purdue e-Pubs

Department of Computer Science Technical Reports

Department of Computer Science

2004

Robust Security Mechnisms for Data Streams Systems

Mohamed Ali

Mohamed ElTabakh

Cristina Nita-Rotaru
Purdue University, crisn@cs.purdue.edu

Report Number:

04-019

Ali, Mohamed; ElTabakh, Mohamed; and Nita-Rotaru, Cristina, "Robust Security Mechnisms for Data Streams Systems" (2004). *Department of Computer Science Technical Reports.* Paper 1602.
https://docs.lib.purdue.edu/cstech/1602

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

ROBUST SECURITY MECHANISMS FOR DATA STREAMS SYSTEMS

Mohamed Ali
Mohamed ElTabakh
Cristina Nita-Rotaru

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-019
June 2004

Robust Security Mechanisms for Data Streams Systems

Mohamed Ali, Mohamed ElTabakh, and Cristina Nita-Rotaru
{mhali, meltabak, crisn}@cs.purdue.edu
Department of Computer Science
Purdue University

Abstract—**Stream database systems are designed to support the fast on-line processing that characterizes many new emerging applications such as pervasive computing, sensor-based environments, on-line business processing and network monitoring. The sensitive nature of the data and the high-demands environment where data can be lost or dropped because of limited buffer storage or real-time constraints, require robust security mechanisms, i.e. mechanisms that not only provide security services, but are also fault-tolerant.**

In this paper we identify the security requirements for data stream systems, focusing on Nile a data stream management system. We present a new method, FT-RC4, that provides efficient and fault-tolerant data confidentiality. We demonstrate its applicability to data streams by using it as building block in the design of a security architecture for Nile and by presenting results for a stream based application.

I. Introduction

The Internet revolution, and more recently the wide-spread use of wireless and sensor networks, created a paradigm shift in the way information is accessed and processed, generating new applications such as real-time network monitoring, surveillance, tracking, plant maintenance, telecommunications, data management and environmental monitoring, to name just a few. Such applications are fundamentally different in the way they output data and perform queries [1]. Thus, they continuously produce large volumes of data (streams) obtained from the environment they operate in. Data streams can be obtained from multiple sources at high-arrival (possibly unpredictable) rates. They are continuous and unbounded. The *transitive* characteristic of data makes the complete storage and processing impossible. In turn, data may be summarized and stored only temporarily for processing.

The queries applied on such data streams are also different from traditional database queries. They are not snapshot queries, but rather continuous queries in which the same query is repeatedly evaluated each time new input arrives. Several queries can be registered in the system and different levels of priority can be defined for each query. As resources are a concern, highest priority queries' requirements are served first, while low priority queries may receive answers that are an approximation of the correct results. To overcome the infinite nature of data streams, the processing is performed on windows of data. Queries can specify the size of the windows and the frequency of the result.

Several systems were designed to cope with the requirements of data stream databases. Examples include: STREAM [2], Aurora [3] and Aurora* [4], and Nile [5], [6].

A. Security Requirements for Data Stream Systems

Many of the data stream applications operate over Internet and/or wireless communication networks and are thus exposed to numerous threats such as:

- *Attacks on data integrity*: data can be injected or modified and it is not in the original form as intended by the sender, or originally stored. Data corruption can be due to faults as well as to malicious actions.
- *Attacks on data confidentiality and privacy*: by eavesdropping of communications channels, or bypassing the access control and authorization mechanisms, or by inferring information from data they have legitimate access to [7], attackers can obtain either access to, or learn private information.
- *Attacks on data validity*: malicious clients can inject or update corrupted streams that can potentially compromise the accuracy of query answers on a stream or set of streams. Such attacks are extremely difficult to defend against and potential solutions require corroborate information from multiple independent sources and often depend on application semantics.
- *Denial of service*: attackers can exhaust either the available bandwidth or the database server resources, preventing legitimate clients from obtaining service. At the extreme, such attacks can render the system unavailable.

As data stream applications process sensitive data that is often classified (military applications) or private (financial, health applications, etc) there is an obvious need for providing security services not only for the applications but for the data stream systems themselves. A comprehensive survey of security and privacy requirements and open issues for a particular type of stream database (sensor databases) is presented in [8]. Below we present the main security services that any stream data system concerned with security should consider:

- *Authentication*: authenticates a client when it requests access to the system.
- *Access control and authorization*: checks if a given client is authorized to register/update data streams or perform queries on streams. Different streams can have different access control and authorization mechanisms. Authenticated clients can have different access control and authorization credentials.

- *Data confidentiality*: guarantees that only intended parties can understand the content of the stream, the query, or the result.
- *Data integrity*: ensures that data is in the form as intended by originator and was not corrupted unintentionally or intentionally.
- *Data non-repudiation*: ensures that a party that performed an operation can not deny that he did it. This service is useful for audit purposes.
- *Data privacy*: defines what is the minimum information that should be disclosed and provides ways of protecting (personal) information even after it was disclosed to other parties.
- *Data validity*: by this we mean that the data stream generated provides meaningful information. This service can be provided under a non-malicious model (like in [9]), or under a malicious model.
- *Survivability*: provides system recovery from either an attack or failure and ensures that a service is available.
- *Security policy*: all the above security mechanisms must be governed by a security policy.

Most of the security requirements listed above are not necessarily specific to data streams systems. However, several of them are more difficult to provide for data streams and standard solutions can not be directly applied, they require additional research.

One challenge is reconciling application specific requirements with security services, in a high-demand environment. For example, many applications require privacy of data, but also audit capability (for example medical applications [10]). Some solutions proposed for this problem, relying on public key encryption [11], [12], provide audit capabilities while preserving privacy, but the associated cost makes them prohibitive to real-time data stream systems.

Another challenge originates from the conflict between security and real-time processing that can impact several services. For access control and authorization the fine-granularity can have a negative effect on the real-time processing. Another example is providing data confidentiality. For example, good candidates to provide confidentiality are stream ciphers [13] because they are highly efficient. However, data can be dropped or lost either at the *communication level* because of the high-rate and data can not be recovered, or at the *application level* because of limited storage capability and processing power. For stream ciphers, the impact will be the de-synchronization between the key-stream and the encrypted data and will result in incorrect decryption of the whole stream, wasting bandwidth and processing power. We would like to point out that block ciphers are not immune to this problem either. They are recommended to be used in encryption modes that also require data reliability.

B. Our Focus

In this paper we investigate the relation between security and fault-tolerance in the context on data streams, focusing on data confidentiality. Our new contributions are:

- We identify security services for data streams systems and propose a secure architecture for a data stream system, Nile [5], [6].
- We focus on a particular service, data confidentiality. We show why current mechanisms fail to address the requirements of real-time data streams and design a mechanism, called FT-RC4, based on the RC4 stream cipher. We evaluate its overhead and show how it performs in a lossy environment.
- We discuss implementation issues of FT-RC4 in Nile and show its performance over queries with different requirements.
- We discuss applicability of FT-RC4 to other security services, such as providing privacy through processing of encrypted data.

The remainder of the paper is organized as follows. We overview related work in Section II. We describe how the security services presented in Section I-A can be accommodated in Nile, a stream database management system. We present the design of FT-RC4 in Section IV. Section V shows how our mechanisms performs within Nile. Finally, we conclude this work and discuss several future work directions in VI.

II. Related Work

In this section we overview related work in several areas related to security for data streams in particular and database in general.

a) Security for Stream Databases: To the best of our knowledge there is very little work that focuses on the security requirements and services for data streams. A significant work in this direction is the work in [8] that overviews the main research directions and challenges in security for database sensor networks. The paper points out among other issues the need for robust security mechanisms, i.e. mechanisms that not only provide security services, but are also fault-tolerant.

b) Access Control for Database Systems: Significant work was done in the area of providing access control to database systems [14]. Some of the work focused on investigating how several access control models can be applied to databases (for example RBAC [15]). Another topic in this area focuses on providing access control [16], protection and administration to XML data sources [17]. More recent results analyze what are the requirements and mechanisms that need to be provided in query processing, in order to provide very fine-grained access control (at the level of individual tuples) [18].

c) Searching and Querying Encrypted Data: Another topic of interest is privacy preservation. In this category work was conducted in a model where the server is not trusted to see the original data, in other words data is stored by servers in encrypted form. In the case servers are not trusted also to process the data, there is a need for algorithms able to process encrypted data. Some of the security issues that are raised when querying encrypted data are discussed in [19]. Methods to execute queries are proposed in [20] and [21]. The first shows how SQL queries can be performed over encrypted data

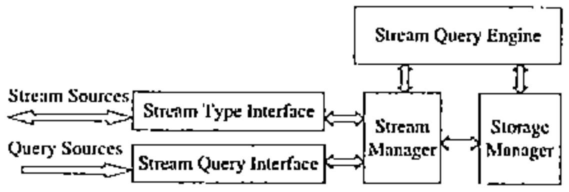


Fig. 1. Nile system architecture

where the query processing is partitioned such that most of the processing happens at the server site, while the latter relies on indexing information attached to the encrypted database to balance the trade off between efficiency and protection requirements. More recent work proposes encryption algorithms that preserve order for numeric data [22].

We would like to point out other significant work addressing the general problem of searching on encrypted data. Some solutions proposed for this problem, rely on public key encryption [11], [12], while others rely on symmetric encryption to achieve similar goals [23]. The work in [23] although efficient because it relies on symmetric cryptography, has the drawback that the search is linear with the size of the document. The work in [24] improves over [23] by using tree structures to avoid the linear scalability.

d) Digital Rights Management for Databases: Another security service that was addressed in the context of databases is rights protection. Work in this direction focused in providing rights protection for relational database systems [25] and more recently on designing resilient schemes that achieve rights protection for sensor streams [26].

III. A Security Architecture for Nile DBMS

In this section we discuss how security services presented in Section I-A can be accommodated in Nile, a stream database management system. We first provide an overview of Nile, then present the proposed security architecture.

A. Nile Architecture

Nile is a stream database management system designed and developed at Purdue University. It is built over a relational database management system called PREDATOR [27] and provides support for processing of continuous and snap-shot queries over data streams.

Nile is a centralized system using a client-server architecture. Several clients can communicate with the system; each client can send multiple input streams and receive one or more output streams as a response to queries. Each output stream from the server corresponds to a query requested by a client. The same client can send input streams and receive output streams, or one client is only sending data and another client is querying the data and receiving the output stream.

Figure 1 shows the main architectural components of Nile. The *Stream Type Interface* component is the interface between the streams generators (i.e. sensors, retail stores, etc) and the system. The definition and configuration of streams is done through this interface. The *Stream Manager* component handles multiple incoming streams and acts as a buffer between the streams sources and the *Stream Query Engine*. The main function of the *Stream Manager* is to register new stream-access requests (queries), retrieve data from the registered streams into local stream buffers, and supply data to the query engine.

The *Stream Query Interface* component is used to register new snap-shot or continuous queries. Snap-shot queries are queries that are executed once over the current data, whereas the continuous queries are queries that reside in the system and are continuously re-evaluated to produce stream of results. In some situations the *Stream Source Interface* and the *Stream Query Interface* can be the same.

The *Storage Manager* is responsible for building and maintaining summaries over data streams, allowing the system to answer queries related to past data. Summaries are maintained at different granularities such that most recent data will have summaries built at a finer level whereas the old data will have summaries built at a coarser level.

The *Stream Query Engine* component is empowered with certain capabilities and features that allow fast and efficient processing of the stream. For example, new access methods are defined such as StreamScan (SScan) to allow efficient non-blocking pipeline execution. The engine also supports new SQL operators such as Window operator (W-Exp) that allow the user to limit his/her interest of the data to a specific period of time. The W-Exp operator is the only operator that is aware of the time and it keeps track of the new items that enter the interesting window and of the expired items that leave the current window. More details about the query processing mechanisms of Nile can be found in [5].

B. Security Architecture

The current architecture of Nile does not provide any security service. Taking into account the architecture and functionality of Nile we reason about which of the security services discussed in Section I-A are relevant and needed for Nile and how can they be provided.

In Figure 2 we propose a generic security architecture for Nile. Two new modules are added. The role of the first module is to handle authentication, encryption, integrity and non-repudiation services. The reason we grouped them together is because sometimes well-known standards or protocols provide all of them or a subset. This module is responsible for authenticating clients, performing key management, integrity and encryption/decryption operations.

The second module added is the *Access Control and Authorization Manager* that is responsible for making sure that input streams, queries or results are performed by authorized clients.

One important aspect is how a decision is made with respect to the security policy. Both server and client can define their own policy in which case a trust negotiation must be performed.

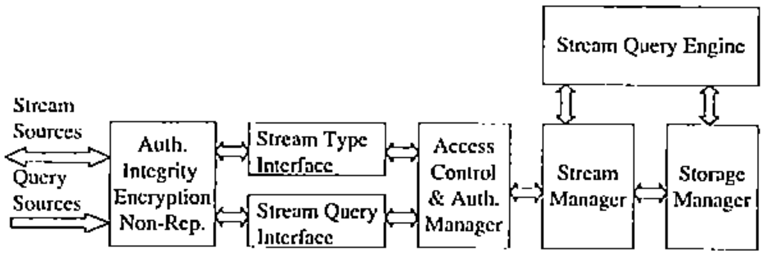


Fig. 2. A security architecture for Nile

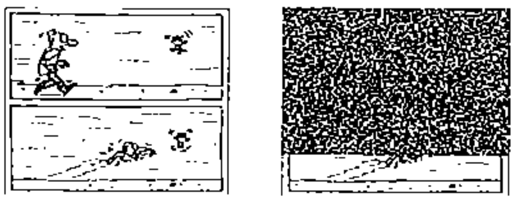


Fig. 3. Effect of loss on RC4

In this work we chose to demonstrate how encryption can be provided for systems operating in a lossy or high-rate environment in which data is lost or dropped because of limited buffer capability, or it can not be recovered because the real-time constraints. We show how a well-known stream cipher can be adapted to operate in such an environment.

IV. FT-RC4 Design

First we will give a description for RC4 algorithm, and demonstrate with an example what happens in case data is lost. We then describe our modification to RC4 to make it more resilient to loss.

A. *Overview of RC4*

RC4 is a stream cipher that is designed to encrypt and decrypt stream of bits, so it processes the message as a stream of bits. Stream ciphers are fast and have as central mechanism the generation of a key-stream (based on a shared secret key) that is then XOR-ed with the plaintext. The decryption operation is similar with the encryption operation.

RC4 uses an internal array S of size 256, and it stores values in range 0..255 with some swapping between the values. The encryption continues shuffling the array S values and finally sums two entries to get the desired key.

One of the disadvantages of stream ciphers when used in lossy environments is that they are prone to de-synchronization between the key-stream and the ciphertext. If such a de-synchronization occurs, decryption of the whole stream fails and bandwidth and processing power is wasted. To demonstrate this behavior we run the following experiment. We create a loss of 3 bytes (randomly) in a BMP file and try to decipher the received data. Figure 3 presents the original picture on the left, and the decrypted picture on the right. As can be seen the effect is devastating. Because of only 3 bytes lost, almost the whole picture is lost. We performed the same test over a JPG file (which is more compressed and less resistant to byte changes) and we noticed that the decrypted file was so significantly compromised that the picture could not be displayed. Finally, we also performed several tests using ASCII text. With only 1 byte lost, the result was that 80 % of the text was meaningless.

B. *FT-RC4 Description*

Fault tolerant RC4 (FT-RC4) is based on RC4 design, and it uses the same stream key generation technique. RC4 can not handle losing any data between the source and the destination, and if some of the data is dropped then the whole stream (after the first loss) will not be decrypted correctly due to the shift in the key generation. Also the current RC4 can not even detect that there is lost data, but relies on underlying communication protocols to achieve this. Although appropriate for other applications, the assumption is not correct for streams for several reasons: data is gone and can not be recovered (or is not relevant anymore), or data is dropped at the receiver end because of limited buffer and processing capabilities.

The main idea of FT-RC4 is to synchronize the bytes in the message by adding synchronization bits before the encryption of the message. The decryption algorithm will then check on these synchronization bits to detect any data loss and try to recover from the loss (re-synchronize the keystream).

The description of FT-RC4 is presented in Algorithm 1. Let us assume that our original message that we need to encrypt is M, and the length of M is L. FT-RC4 first expands the message by padding a specific number of synchronization bits n after every specific number of bits from the original message M, so all transmitted units in M now carry synchronization bits. The synchronization bits simply form a counter which starts from 0 and reset whenever it reaches its maximum ($2^n - 1$), where n is the number of synchronization bits.

The expanded message will be M' and its length will be L', where $L' > L$. FT-RC4 then encrypts M' using the same RC4 encryption technique, and sends the resulted cipher text C to the destination. The destination then will decrypt the message C to get M', but after decrypting every transmitted

Algorithm 1 FT-RC4 description

```
Key Schedule:
    for i = 0 to 255 do
        S[i] = i
    j = 0
    for i = 0 to 255 do
        j = (j + S[i] + k[i mod L])(mod 256)
        swap (S[i], S[j])
Encryption:
    i = j = 0
    for each byte mi in message M
        i = (i + 1) (mod 256)
        j = (j + S[i]) (mod 256)
        swap(S[i], S[j])
        t = (S[i] + S[j]) (mod 256)
        Ci = mi XOR S[t]
Decryption:
    i = j = 0
    for each byte ci in message C
        i = (i + 1) (mod 256)
        j = (j + S[i]) (mod 256)
        swap(S[i], S[j])
        t = (S[i] + S[j]) (mod 256)
        mi = ci XOR S[t]
        -check mi to see if it is the expected one?
        if YES
            remove synch. bits from this byte
        else
            figure out how many bits are lost, shift
            the stream key, and inject zero-bits.
```

unit it must check whether that unit is the expected one, or that unit arrived out of order and there was data loss. This information can be obtained from the synchronization bits. If there is no data loss and that was the expected unit, the algorithm removes the synchronization bits, and continues the decryption. If the decryption algorithm detects data loss, then it has to detect how many bits are lost and start shifting the key stream by the same value to resynchronize the message bits with the stream key. In case the resynchronization does not take place then the whole stream after the first loss will be garbage.

We note that it is not enough to detect the loss and shift the key stream; the algorithm also needs to recover from the loss. The algorithm must also readjust bytes boundaries of the original message M because the lost data may not be multiple of bytes. In this case although the decryption is correct. most applications reading the data in bytes or words will not be able to read the data (i.e. text editors, audio and video applications, images, etc). FT-RC4 handles this issue by injecting bits (set to zero) instead of the lost ones, this way readjusting bytes boundaries such that applications at the destination can read the message normally.

To summarize, the encryption in FT-RC4 is modified over the one in RC4 by adding the expansion phase and padding the synchronization bits before the encryption, and the decryption in FT-RC4 is modified over the one in RC4 by adding the checking and recovering phase. FT-RC4 uses the same technique for the stream key schedule, which is efficient due to its simplicity.

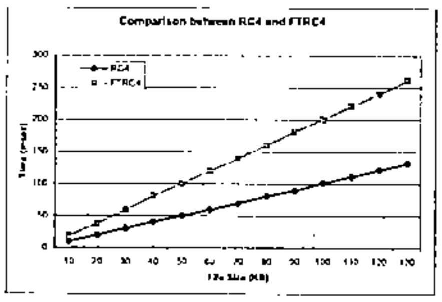


Fig. 5. FT-RC4 Overhead

C. FT-RC4 Evaluation

We evaluate how resilient is FT-RC4 by comparing its performance in a lossy environment with the standard RC4, and measure how efficiently can FT-RC4 recover from the loss of bytes between the source and the destination. The evaluation is done over text and images.

Text: We compared both techniques over text data streams. We set the loss rate to be a percent of the input data size (i.e. 1% and 5%) and the lost bytes will be selected uniformly form the input file. For a text of about 700 words, with a 1% loss, RC4 fails to recover the text correctly after 1 byte is lost. while FT-RC4 recovers very quickly.

Image: We set the loss rate to be a percent of the input data size (i.e. 1% and 5%) and the lost bytes were uniformly selected from the input file. We protected the header part untouched, because the header part of a file specifies its type, so it can be opened by the correct application. If any loss or damage occurs in the header part, the entire file will be unreadable even if the remaining part is correct. Also if we assume that the data loss occurs in the applications layer, it is practical that these application handle the header parts carefully since they are critical.

We compared the algorithms over JPEG files, which are more compressed and less resistant to byte changes. Figure 4 presents an original picture compared with the results of its encrypted/decrypted with RC4 and FT-RC4 for. As it can be seen with only 1% loss, RC4 fails to recover the image, while FT-RC4 is still able to show the image when loss is 5 %. At a 1% loss where RC4 fails. FT-RC4 recovers the picture almost in its original quality. With a lost of higher than 1% RC4 fails.

FT-RC4 Overhead: We compared the performance of the standard RC4 and our FT-RC4 with respect to the time required by the algorithm to perform the encryption and decryption operations. Figure 5 shows the performance results. The figure shows that both techniques linearly increase with the file size, but FT-RC4 has a higher slop. The measurements

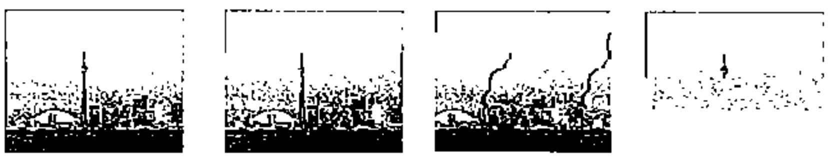


Fig. 4. FT-RC4 Resilience

```
SELECT SUM(R1.txn.Price() * R1.txn.Quantity())
FROM Retail1 R1
WHERE R1.txn.ItemID = 15
WINDOW 00.00.05.00:
```

Fig. 6. Query Q1 description

show that FT-RC4 almost takes double the time taken by RC4. The reason is that FT-RC4 performs two more operations over the stream which are the expansion phase to augment the synchronization bits in the encryption, and the compaction phase to remove the synchronization bits in the decryption.

V. Integration of FT-RC4 in Nile

In this section we present performance results and discuss several aspects of the integration of FT-RC4 in Nile.

A. Experiments in Lossy Environments

We implemented RC4 and FT-RC4 protocols inside the Nile system by adding a security layer (encryption and decryption) between the clients and the server, such that both entities either use the standard RC4 or use the FT-RC4. We evaluated the performance of the system in a lossy environment by demonstrating the performance of RC4 and FT-RC4 using a retail store application [6] that sends a stream of transactions generated from 5 retail stores, each transaction consisting of $< StoreID, ItemID, Price, Quantity, TimeStamp >$.

First, FT-RC4 is evaluated over a simple query presented in Figure 6. The meaning of the query is to select the sum of the product of the price and the quantity from the stream Retail1 where the item $itemID$ equals to 15 and the window of interest is of size 5 seconds. The window operator has syntax Window hh,mm,ss,uu; where h means hours, m means minutes, s means seconds, and u means microseconds.

We executed the query under loss rates of 1%, 5% and 10%. Figure 7 shows the results generated from the system for query Q1, with Figures (a), (b) and (c) corresponding to the three different loss rates, 1%, 5% and 10%, respectively. Figure 7 shows that the query result changes over time as new items arrive inside the window of interest and old items expire from the window. As it can be noticed, the RC4 protocol simply fails to process the stream and after the first loss, it crashes the system as it starts producing garbage. This is the reason why the line representing the RC4 protocol in Figure 7, (a), (b) and (c) stops after several readings. On the other hand, it can be noticed that FT-RC4 is very resistant to losses, and although it does not produce the optimal results due to replacing the lost values with 0, it still preserves the stream's behavior and produces acceptable results.

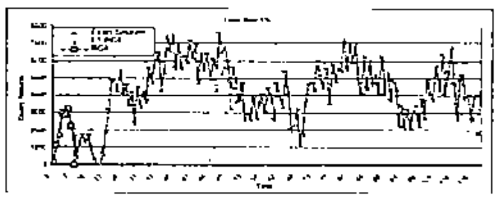
For query Q1, the effect of the loss is limited only to items with $ItemID$ equals to 15, so any loss that occurs to values related to other items will not affect the query result. To make the query more sensitive to losses we modified the query (described in Figure 8), by removing the $WHERE$ clause from Q1. This way, any loss over the selected columns in the query will affect the query results. We executed the query under loss rates of 1%, 5% and 10%. Figure 9 shows the results generated from the system for query Q2, with Figures (a), (b) and (c) corresponding to the three different loss rates, 1%, 5% and 10%, respectively. Figure 9 shows that RC4 has very poor performance even under a low loss rate, while FT-RC4 has a high resistance to losses and it produces meaningful results.

Finally, we evaluated the FT-RC4 protocol over a more complex query involving two data streams. The query Q3 description is presented in Figure 10, while the results are presented in Figure 11. In Figure 11 (a) it is assumed that both streams have the same loss rate which is 10%, while in Figure 11 (b) it is assumed that both streams have different loss rates, 10% and 20% respectively. Figure 11, again shows that RC4 fails immediately, while FT-RC4 behaves well in both cases.

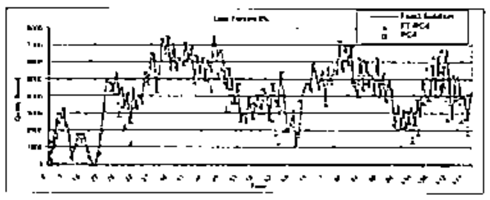
B. Adapting FT-RC4 to Stream Rates

It should be noted that FT-RC4 can also fail to decipher data correctly if the stream looses one complete cycle. The size of a cycle depends on the number of synchronization bits used in the protocol. For example, if the protocol uses n synchronization bits then the cycle size equals to 2^n. In this case, if at any time the server losses one complete cycle of contiguous units, then the FT-RC4 will fail to detect and re-synchronize the keystream correctly.

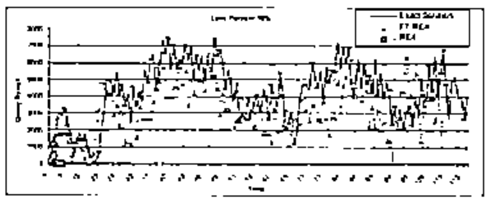
In this section we propose an adaptive scheme by which the FT-RC4 can change the number of synchronization bits such that it minimizes the transmission overhead and at the same time it will be able to cope with peak losses that may occur from time to time over the stream. We will demonstrate the



(a) 1% loss



(b) 5 % loss

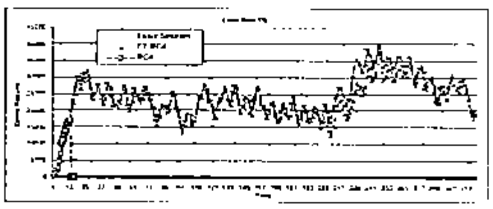


(c) 10 % loss

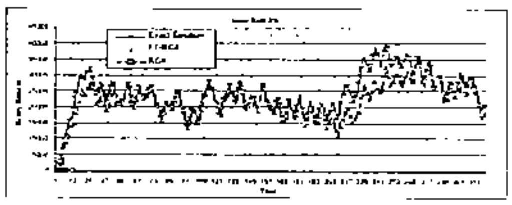
Fig. 7. Results for query Q1

SELECT SUM(R1.txn.Price() * R1.txn.Quantity())
FROM Retail1 R1
WINDOW 00.00.05.00;

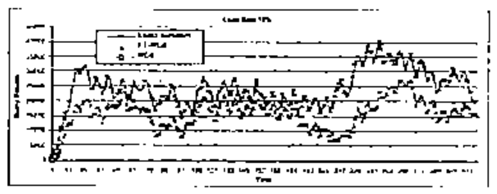
Fig. 8. Query Q2 description



(a) 1% loss



(b) 5 % loss



(c) 10 % loss

Fig. 9. Results for query Q2

SELECT R1.txn.Quantity()+R2.txn.Quantity()
FROM Retail1 R1, Retail2 R2
WHERE R1.txn.ItemID() = R2.txn.ItemID()
WINDOW 00,00,10,00;

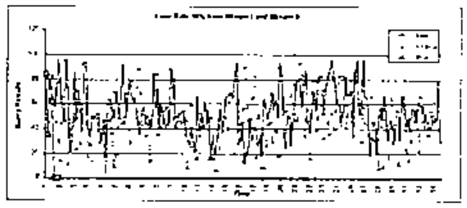
Fig. 10. Query Q3 description

importance of the adaptation schema using a new query Q3, over the same application as in previous section.

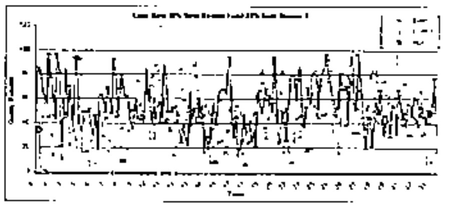
In some applications the transmission of the data is in the form of packets, each packet consists of a payload that contains the actual data and other additional header fields to hold certain information necessary for the transmission. In such applications adding one byte in each packet to be used (at the application level) as synchronization bits in the FT-RC4 is sufficient and the overhead is acceptable. But in many stream applications such as sensor networks the transmission of the data is unstructured, it can be a stream of integers, characters, etc. Therefore for this type of applications it is clear that using one byte or fixed number of bits with each transmitted unit will involve a high and sometimes unacceptable overhead.

The purpose of the adaptation is to achieve two goals: (1) use a minimal number of synchronization bits, (2) avoid loosing one complete cycle of subsequent units. The proposed method is as follows:

1) Initially when client C registers with the server to start sending data, the client informs the server about the expected sending rate R.
2) The server calculates the overall transmission rate in the system at the current moment, and based on the available resources, the server estimates the overall lose rate S.
3) The server divides S over the streams according to their sending rate ratios. Let's assume that client C is going to suffer a loss rate (number of packets per second) equal to L. In this case, the server sends a message to client C



(a) Stream 1 and Stream 2 10% loss



(b) Stream 1 10% loss and Stream 2 20% loss

Fig. 11. Results for query Q3

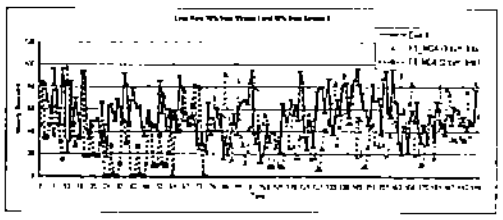


Fig. 12. Effect of adapting FT-RC4 to stream rate

to set the number of synchronization bits to $log(L) + 1$ such that no cycle can occur without the server detecting the loss.

4) Since the stream rate can change over time: burst, normal or low, then the server periodically performs steps (2) and (3) to adapt the number of the synchronization bits to the currect state of the stream.
5) To avoid transit periods, the server will not assume that the client is using the new value for the synchronization bits until the client sends back a message to confirm the change.

To show the importance of the adaptation scheme we performed the following experiment using query Q3 described in Figure 10. In the experiment it is assumed that *Stream I (Retail I)* has a moderate arrival rate and loss rate set to 10%, while *Stream II (Retail II)* has a very high arrival rate and loss rate set to 40%, and we forced *Stream II* to loss from time to time 5 subsequent bytes. Figure 12 shows the result of query Q3 under two scenarios. In the first scenario there is no adaptation, and all clients use fixed number of synchronization bits (set to 2). In the second scenario the adaptation will allow the server to ask the client sending *Stream II* to use more synchronization bits (set to 3) as the server knows that it may loose more than 4 subsequent bytes. The results show that before the loss of any 5 subsequent bytes both scenarios produce exactly the same results. However, after the first loss of 5 subsequent bytes, FT-RC4 fails in the first scenario as it looses the synchronization and starts producing garbage, while the FT-RC4 in the second scenario performs well and is able to cope the loss.

C. Using FT-RC4 for Other Security Services for Streams

There are other security services that we believe can benefit from FT-RC4. One such service is providing data privacy, while maintaining audit capabilities. In such a service the servers are not trusted, so clients will input data in encrypted form. However, there is a need to be able to do search on the encrypted data, and sometimes to be able to delegate certain keyword search capabilities to authorized parties.

Recent results [11], [12] addressing the problem of searching on encrypted data provides rely on public key encryption. Although appropriate for off-line logging and traditional databases, their cost is prohibitive for data streams. More appropriate schemes for data streams are schemes that use symmetric encryption, in particular stream ciphers. In fact a scheme like that was proposed in the past [23], having in focus email as the target application. The scheme proposed in [23] can be adapted for data streams, particularly because of the reduced complexity of both data and operations that can be performed on streams. When used in lossy environments, the scheme will suffer from the same problems as RC4. Therefore, we believe that the same synchronization technique we used for data confidentiality, can be applied to make symmetric-based searching on encrypted streams robust to faults. We would like to explore the topic in the future and apply it to several stream applications with different environments and security requirements.

VI. Conclusions and Future Work

In this paper we focused on security services for data streams. More precisely, we focused on data confidentiality based on stream ciphers and on the interaction between fault-tolerance and security. We show how current stream schemes fail to decipher correctly when de-synchronization between the ciphertext and the keystream happens because of lossy environments or inability of the application to process the incoming streams. We proposed a modification to a well-known stream cipher RC4, to cope with the problem. We showed how the modified scheme, referred as FT-RC4 addresses the problem, how can be used as a building block for a security architecture for Nile, a data stream database system and presented results for several queries with different requirements and loss rates. Finally we discuss how other security services such as privacy can benefit from FT-RC4.

We also discussed how our scheme can be made adaptive. Although the scheme we proposed is not very complex, the results we presented indicate the benefits that can be obtained. We would like in the future to design more sophisticated adaptive algorithms that are robust to faults while still providing the security service they were designed for. In addition, we would like to explore symmetric-based privacy preserving schemes, in the same context of data streams, and experiment with several stream applications operating under different environment constraints and having different security requirements.

REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *ACM Symp. on Principles of Database Systems (PODS 2002)*, June 2002.

[2] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, resource management, and approximation in a data stream management system," in *First Conference on Innovative Data Systems Research (CIDR)*, January 2003.

[3] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," *VLDB Journal*, vol. 2, pp. 120–139, August 2003.

[4] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable distributed stream processing," in *First Biennial Conference on Innovative Database Systems (CIDR'03)*, January 2003.

[5] M. Hammad, W. Aref, M. Franklin, M. Mokbel, and A. Elmagarmid, "Efficient execution of sliding window queries over data streams," Tech. Rep. CSD TR 03-035, Department of Computer Science, Purdue University, December 2003.

[6] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. Ghanem, R. G. andIhab F. Ilyas, M. Marzouk, and X. Xiong, "Nile: A query processing engine for data streams," in *Proceedings of the 20^{th} IEEE International Conference on Data Engineering, ICDE*, 2004.

[7] C. Farkas and S. Jajodia, "The inference problem: A survey," *SIGKDD Explorations, Special Issue on Privacy and Security*, vol. 4, pp. 6–12, December 2002.

[8] B. Thuraisingham, "Security and privacy for sensor databases," *Sensor Letters*, 2004.

[9] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani, "The price of validity in dynamic networks," in *SIGMOD 2004*, 2004.

[10] C. Farkas, M. Valtorta, and S. Fenner, "Medical privacy versus data mining," in *5^{th} World Multiconference on Systemics, Cybernetics and Informatics*, pp. 194–200, July 2001.

[11] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Searchable public key encryption," in *Eurocrypt 2004*, 2004.

[12] B. R. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an encrypted and searchable audit log," in *Network and Distributed Systems Security Symposium 2004*, 2004.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied CryptographyL Chapter 6*. CRC Press, october 2001.

[14] S. De, C. Eastman, , and C. Farkas, "Secure access control in a multi-user database," in *ESRI User Conference*, 2002.

[15] S. Osborn, "Database security integration using role-based access control," in *IFIP WG11.3 Working Conference on Database Security*, August 2000.

[16] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for XML documents," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, pp. 169–202, May 2002.

[17] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti, "Protection and administration of XML data sources," *Data Knowl. Eng.*, vol. 43, no. 3, pp. 237–260, 2002.

[18] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending query rewriting techniques for fine-grained access control," in *SIGMOD 2004*, 2004.

[19] M. Kantarcioglu and C. Clifton, "Security issues in querying encrypted data," Tech. Rep. CSD TR 04-013, Purdue University, Computer Sciences Department, 2004.

[20] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pp. 216 – 227, 2002.

[21] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, "Balancing confidentiality and efficiency in untrusted relational DBMSs," in *Proceedings of the 10^{th} ACM conference on Computer and communication security*, pp. 93–102, ACM Press, 2003.

[22] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *SIGMOD 2004*, SIGMOD 2004.

[23] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Research in Security and Privacy*, 2000.

[24] R. Brinkman, L. Feng, J. Doumen, P. Hartel, and W. Jonker, "Efficient Tree Search in Encrypted Data," in *Proc. of the 2^{nd} Intl. Workshop on Security in Information Systems*, April 2004.

[25] R. Sion, M. Atallah, and S. Prabhakar, "Protecting rights over relational data using watermarking," *IEEE Journal of Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16, June 2004.

[26] R. Sion, M. Atallah, and S. Prabhakar, "Resilient rights protection for sensor streams," in *Very Large Databases Conference (VLDB)*, 2004. To appear.

[27] P. Seshadri, "Predator: A resource for database research," *SIGMOD Record*, vol. 27, no. 1, pp. 16–20, 1998.