# FT-RC4: A Robust Security Mechanism for Data Stream Systems

Mohamed Ali, Mohamed ElTabakh, and Cristina Nita-Rotaru

Department of Computer Science, Purdue University,
West Lafayette, IN 47906, USA
{mhali, meltabak, crisn@cs.purdue.edu}

**Abstract.** Stream database systems are designed to support the fast on-line processing that characterizes many new emerging applications such as sensor-based environments, on-line business processing and network monitoring. Data stream processing is a highly demanding environment where streams are usually infinite, bursty, and running at high arrival rates. Due to limited buffer storage or real-time constraints, data items may be dropped out of the system and lost for ever. In many applications, sensitive stream data needs to be secured against malicious attacks. Various security mechanisms have been well studied in literature. However, these mechanisms are not tuned to work in the lossy streaming environment. Stream security mechanisms are required to provide security services and to be fault-tolerant as well.
In this paper we identify the security requirements for data stream systems, focusing on *Nile*, a prototype query processing engine for data streams developed at *Purdue University*. We first propose a security architecture for data stream systems, then focus on a particular service: data integrity and confidentiality. We present a new mechanism, FT-RC4, that provides data integrity and confidentiality. We demonstrate its practicality by implementing it inside our prototype data stream system and evaluating its performance.

## 1 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. Such applications are fundamentally different in the way they produce data and perform queries [1]. They continuously generate 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 the data makes storing and processing the whole data infeasible. Data items may be dropped from the buffers and lost for ever if they are not processed in a timely fashion. Queries that are applied on such data streams are not only snapshot queries, but also continuous queries in which the same query is incrementally evaluated each time new input arrives. Processing is performed on a sliding window over the stream to limit the attention to the most recent data items and overcome the infiniteness of the stream.

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 databases that is generated by a large number of sensors is presented in [6]. Below we present the main security services that any data stream system that is 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 data streams or perform queries on streams.
- *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 that is intended by the originator and is not corrupted intentionally or unintentionally.
- *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*: ensures that data generated contains meaningful and correct (non-misleading) information in the generated data streams.
- *Survivability*: provides system recovery from either an attack or a 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 stream systems. However, most of them are more difficult to provide for data streams where standard solutions cannot be applied directly because of the high-demand characteristics of the environment. Additional research is needed to overcome the challenges posed of the new paradigm. One challenge is reconciling application-specific requirements with security services in a high-demand environment. For example, many applications such as medical applications [7], require privacy of data, but also audit capability. Solutions proposed to address this problem, provide the desired audit capabilities and preserve privacy, but have a high associated cost that makes them prohibitive to real-time data stream systems.

Another challenge originates from the conflict between security and real-time processing. For example, providing fine-granularity access control and authorization can have a negative effect on the real-time processing because of the additional processing overhead. Another example is providing data confidentiality for data streams. Although stream ciphers [8] seem to provide the desired performance for data streams, they fail to operate correctly when there is a de-synchronization between the keystream and the encrypted data. Such a de-synchronization is very likely to occur in an environment where data can be dropped or lost either at the *communication level* because of the high transmission rate, or at the *application level* because of the limited storage capability

and processing power. The real-time processing does not allow for trying to recover the data via retransmissions.

Several systems were designed to cope with the demanding performance of data streams. They include: STREAM [2], Aurora [3], and Nile [4]. However, none of them focused on providing security services.

The work presented in this paper is a first step in addressing security concerns for data stream systems, focusing on data integrity and confidentiality. The research in this paper is conducted in the context of the Nile [4] prototype data stream system. Our main contributions are:

- We identify security services for data stream systems and propose a secure architecture for a prototype data stream system.
- We focus on data integrity and confidentiality and propose a mechanism appropriate to data streams. Our scheme, FT-RC4, is able to withstand data loss and recover from it by re-synchronizing the encrypted data with the corresponding keystream.
- We demonstrate the applicability of FT-RC4 to data streams by implementing it in the discussed prototype data stream system and by evaluating it over realistic data streams.

The remainder of the paper is organized as follows. We overview related work in Section 2. We describe how security services can be accommodated in the architecture of an existing data stream system in Section 3. Section 4 focuses on data integrity and confidentiality and presents the design of our scheme, FT-RC4. Section 5 evaluates the performance of the proposed mechanism. Finally, we conclude this work in Section 6.

## 2 Related Work

In this section, we overview the related work in several directions related to the security of data streams in particular and databases in general, These directions can be summarized as follows:

**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 systems. A significant work in this direction is the work in [6] that overviews the main research directions and challenges in security for sensor network databases. 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.

**Access Control for Database Systems.** Significant work has been conducted in the area of providing access control to traditional database systems [9]. Some of the work focused on investigating how several access control models can be applied to databases (for example RBAC [10]). Another topic in this area focuses on providing access control [11], protection and administration to XML data sources [12]. 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) [13].

**Data Confidentiality and Integrity.** Block ciphers,e.g., DES [14] and AES [15], has been proposed to provide data confidentiality. Although they are widely used, their

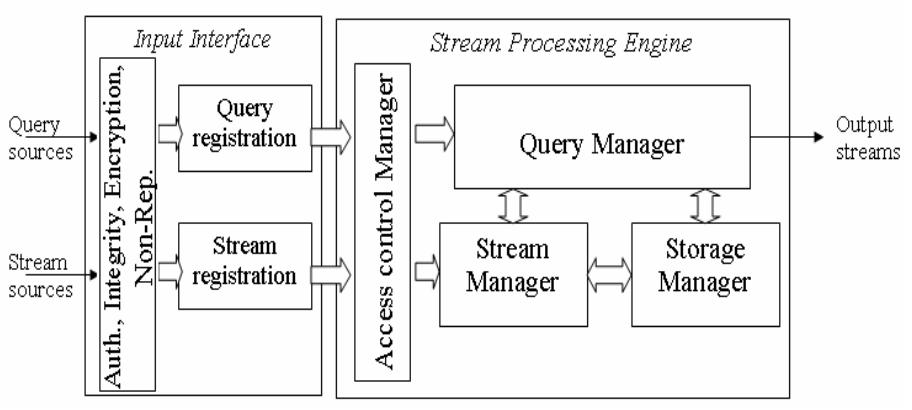


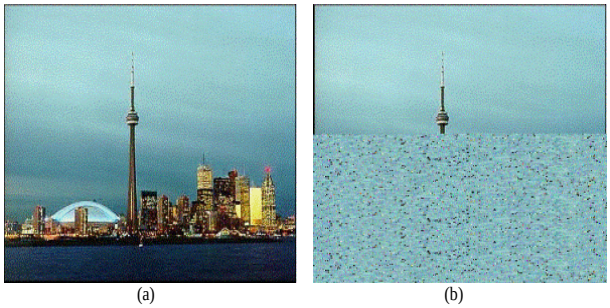

**Fig. 1.** Nile system architecture

performance makes them prohibitive to data streams. Another type of ciphers, stream ciphers, (such as RC4 [16]) have performance that makes them appropriate for streams. Several techniques have been proposed to provide data integrity. They were based on hashing techniques to calculate authentication bits that can be viewed as a digest of the messages. MD5 [17], SHA1 [18] and HMAC [19] are examples of these techniques. Our work builds on existing work in providing data integrity and confidentiality.

## 3 A Security Architecture for Data Stream Systems

In this section, we discuss how some of the security services presented in Section 1, can be achieved in data stream systems. We extend the architecture of a data stream systems to accommodate components that are specially designed to provide security services.

Figure 1 shows the main architectural components of our prototype data stream system. The *Stream Registration* component is the interface between the stream generators and the system. Its main function is to register new streams into the system. Queries need to be registered by the *Query Registration* component in order to get access to the streams inside the system. The *Stream Manager* component handles multiple incoming streams and acts as a buffer between the streams sources and the *Query Engine* where query processing takes place. The *Storage Manager* component is responsible for building and maintaining summaries over data streams, allowing the system to answer queries efficiently. The above described system is extended with two basic components. The first component handles authentication, encryption, integrity and non-repudiation services. It is responsible for authenticating clients, performing key management, integrity and encryption and decryption operations. These services are end-to-end services, thus the component is placed at the outer level of the system. The second component handles access control policies. It is responsible for making sure that queries are performed by authorized clients who have access to the targeted streams. This component is placed into the system after the query and stream registration phase to keep track of which queries are allowed to access which streams.



(a) (b)

**Fig. 2.** The effect of lost data items on RC4 (a) original image (b) received image

## 4 FT-RC4 Design

A basic security service that is necessary for data streams is data confidentiality and integrity. In this section we focus on this service.

Stream ciphers, such as RC4 [16] provide the performance required by the high-demand environment of data streams. The main mechanism of such ciphers is to generate a keystream based on a shared secret key. The generated keystream is then XOR-ed with the original stream to obtain the encrypted stream. The decryption operation is performed in a similar manner as the encryption: the keystream is recreated at the receiver side, then XOR-ed with the encrypted stream to obtain the original data. Stream ciphers, including RC4 are vulnerable to de-synchronization between the keystream and the ciphertext, in case of data loss. Figure 2 shows that RC4 [16] fails to decrypt the data after it encounters the first error due to the de-synchronization between the keystream and the incoming data stream.

To overcome this problem, we propose FT-RC4, that builds on the design of RC4. FT-RC4 extends RC4 to work in the lossy streaming environment, where recovering the data through retransmissions is not an option. Figure 3 presents the design of FT-RC4. FT-RC4 consists of three steps taken at the sender side and other corresponding three steps at the receiver side. The sender divides the stream into cycles of fixed length. After encrypting each cycle, the sender appends synchronization and integrity bits immediately after each cycle. The receiver uses these extra bits to check the integrity of data. If the cycle is believed to be corrupted, it is thrown away or replaced by zeros. Otherwise, the synchronization bits are used to adjust the keystream and decrypt the cycle. For each cycle, the following steps are performed at the sender side:

1. The input stream is XOR-ed with the keystream to obtain an encrypted stream.
2. The encrypted stream is passed through a position registrar to punctuate the stream with the current position of the stream by appending *stream position locating bits*. (Notice that position bits may or may not be encrypted depending on whether we are interested in hiding stream contents or contents plus position as well. This encryption is done through a separate encryption step and may use a different key and technique.)

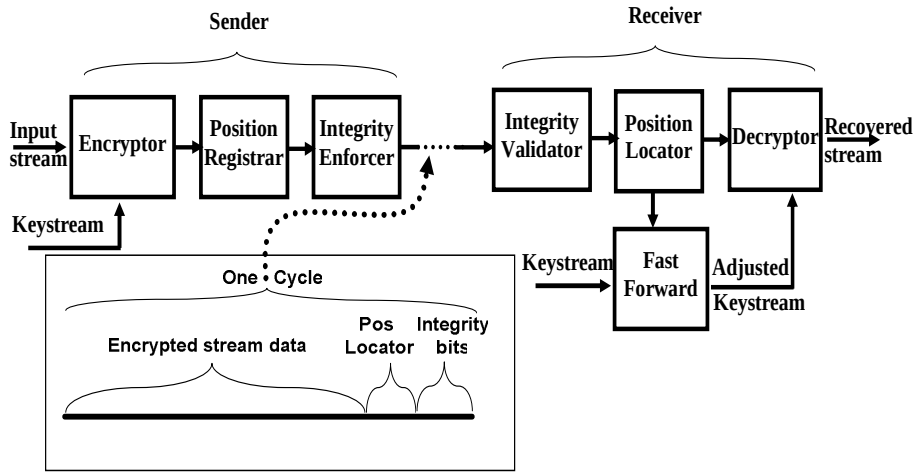


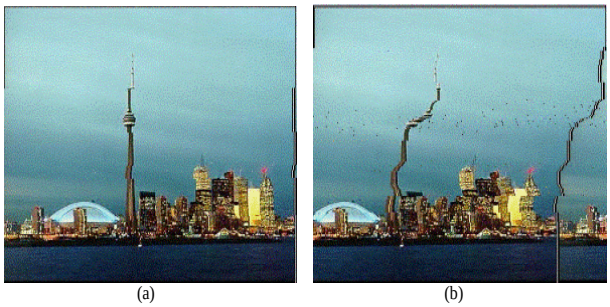

**Fig. 3.** Basic components of FT-RC4

3. The encrypted stream and position punctuation is passed through an integrity enforcer to hash the current cycle and obtain *integrity bits* that are considered as the cycle digest that verifies the stream integrity.

As mentioned above, the receiver does not have the ability to recover the lost data. Thus, the receiver goals are to detect a loss and then resynchronize the keystream with the incoming encrypted stream. Corrupted cycles are considered lost, while correct cycles are decrypted and if data loss occurred, used to resynchronize the keystream for further decryption. *Integrity bits* are used to check the integrity for each cycle, while *stream locating bits* are used to fast forward the keystream till it is repositioned at the correct position. Due to the bursty nature of streams, errors are usually close-by, destroying one cycle and leaving others uncorrupted. At the receiver side, the following steps are carried over:

1. The receiver slides a window of the same size as the cycle length over the received encrypted stream. It calculates the same hash function and checks the integrity bits. If integrity checks, the cycle is considered correct, otherwise, it is considered to be corrupted and thrown away. (Notice that the hash function should be evaluated incrementally as the window slides [20] to allow the efficient computation of integrity bits as one data item gets into the window and another one leaves it.)
2. For a correct cycle, the position locating bits are used to fast forward the stream to the correct position and to inform the decryptor to report data loss to the client, and perhaps add filling values instead of the corrupted ones. These filling values would be of interest to some applications, e.g., images where image size is important.
3. The adjusted keystream is XOR-ed with the encrypted stream to obtain the original data stream.

FT-RC4 is able to detect corrupted portions and is able to readjust the keystream in order to regain synchronization. In Figure 4 we show the same image from Figure 2,



(a) (b)

**Fig. 4.** The effect of lost data items on FT-RC4: (a) 1% loss rate, (b) 5% loss rate

encrypted with FT-RC4, under two loss rates. Figure 4 shows that FT-RC4 recovers from lost data and continues the decryption process (whereas RC4 generated garbage after the first error it encountered). Although the image gets distorted as the loss rate increases, it is still viewable.

## 5 Experiments

In this section, we evaluate the performance of FT-RC4 inside the Nile data stream system. We implement FT-RC4 inside Nile and perform experiments using a real data set generated from Wal*Mart [1] retail stores. Each store is sending its transactions to a centralized data stream system for online processing. Each transaction comes in the form of <*StoreID, ItemID, Price, Quantity, TimeStamp*>. A sample query "Q", shown in Figure 5, is performed on the system side under different rates of uniform information loss. The query calculates the total revenue by multiplying the price of each item by the associated sold quantity and accumulating the sum. A window operator of 5 minutes is placed to limit the attention to the most recent five minutes. The window format is *window (hour, minutes, seconds, mseconds).*

```
SELECT SUM(R1.Price() * R1.Quantity())
FROM Retail1 R1
WINDOW 00,05,00,00;
```

**Fig. 5.** Query "Q" syntax

Figure 6 shows the result of query "Q" under different loss rates. The figure shows that FT-RC4 is able to produce answers that are very close to the ones in a non-lossy environment. The lower the loss rate is, the more accurate results are obtained. The obtained result is always less than the exact one due to replacing the lost bytes with zeros which in turn decreases the sum value. Figure 7 shows the normalized mean square error (MSE) between the exact and obtained answers. The horizontal segments

[1] Data was supplied by Wal*Mart and NCR Corporations

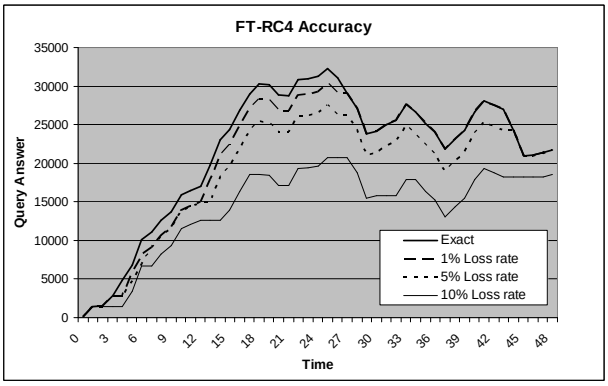



**Fig. 6.** The effect of loss rate on the query answer

of the curves indicate that the newly-arriving tuples as well as the expiring tuples are both correct tuples. Therefore, they do not change the MSE. The positive-slope segments indicate that the newly-arriving tuples are lost and replaced by zeros. Finally, the negative-slope segments indicate that expiring tuples are zero-replaced tuples, and their expiration reduces the MSE. As expected, the MSE increases with the increase of the loss rate. Increasing the loss rate increases the magnitude and number of positive and negative slope segments.

Figure 8 shows the processing time for the FT-RC4 under different cycle sizes. Notice that FT-RC4 requires more processing time than RC4 due to augmenting the stream with synchronization bytes. The figure shows that as the cycle size increases the processing time decreases because the number of the synchronization bytes decreases. Figure 9 shows the effect of the cycle size in the recovery process. As the cycle size increases, more data is lost. This is because in FT-RC4, if an error occurs in a cycle, the whole cycle is lost. Finer granularities of cycles offer a better chance of recovery, at an additional processing overhead.

## 6 Conclusions

In this paper, we outlined the desired security services for data streams and suggested an architecture for data stream systems that provides such services. We discussed the challenges encountered when designing security services for data streams, emphasizing the interaction between fault-tolerance and security.

We exemplified why standard solutions can not directly be applied in a lossy stream environment where data loss recovery through retransmissions is not possible, by focusing on data integrity and confidentiality. We proposed a new scheme FT-RC4 designed specifically to cope with the requirements of data streams and showed its usefulness by implementing it and evaluating it in the context of a prototype data stream system.

This work is just the first step towards a secure data stream system. We plan to perform further evaluations of our scheme under different application scenarios, with

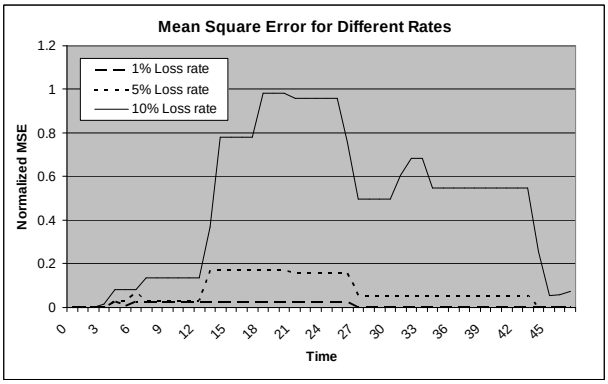



**Fig. 7.** The effect of loss rate on MSE

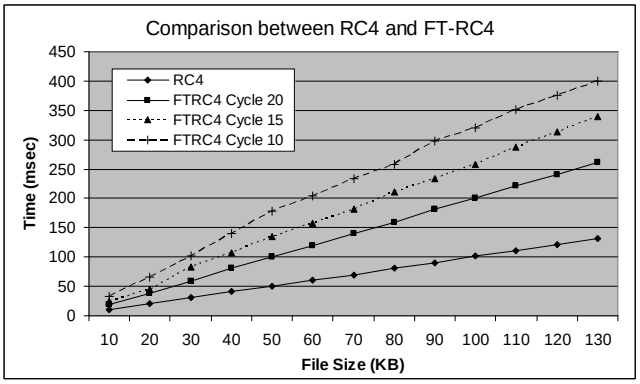



**Fig. 8.** The CPU overhead for both RC4 and FT-RC4

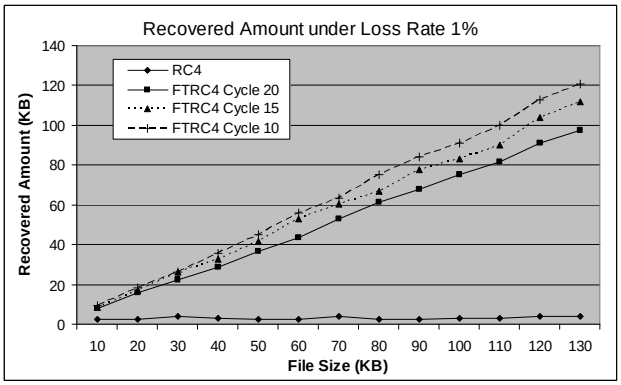



**Fig. 9.** The effect of cycle length on the amount of recovered data

various requirements. In addition, there are several other services we just outlined in this paper and which require further research.

## 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)*, 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. 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. and Ihab 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.
5. 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.
6. B. Thuraisingham, "Security and privacy for sensor databases," *Sensor Letters*, 2004.
7. 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.
8. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied CryptographyL Chapter 6*. CRC Press, October 2001.
9. S. De, C. Eastman, , and C. Farkas, "Secure access control in a multi-user database," in *ESRI User Conference*, 2002.
10. S. Osborn, "Database security integration using role-based access control," in *IFIP WG11.3 Working Conference on Database Security*, August 2000.
11. 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.
12. 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.
13. S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending query rewriting techniques for fine-grained access control," in *SIGMOD 2004*, 2004.
14. D. Coppersmith, "The data encryption standard (DES) and its strength against attacks," IBM Journal of Research and Development, 38(3), pp. 243–250, 1994.
15. J. Daemen and V. Rijmen, "Rijndael, the advanced encryption standard," Dr. Dobb's Journal , Vol. 26, No. 3, pp. 137–139, March 2001.
16. S. Fluhrer and D. McGrew, "Statistical Analysis of the Alleged RC4 Keystream Generator," Fast Software Encryption, Springer-Verlag, 2000.
17. Ronald Rivest, "The MD5 Message-Digest Algorithm," Internet Engineering Task Force RFC 1321, April 1992.
18. U. S. National Bureau of Standards, "Digital signature standard," Technical Report FIPS186, U.S. Department of Commerce, May 1994.
19. H.Krawczyk, M.Bellare, and R.Canetti, "HMAC:Keyedhashing for message authentication," Internet Engineering Task Force RFC 2104, February 1997.
20. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser, "Incremental cryptography and application to virus protection," Symposium on the Theory of Computing (STOC), 1995.