Cryptanalysis of a Fast Encryption Scheme for Databases

Stéphane Jacob[1]
SECRET Project-Team - INRIA Paris-Rocquencourt
Domaine de Voluceau - B.P. 105 - 78153 Le Chesnay Cedex - France
stephane.jacob@inria.fr

***Abstract*—Protecting the confidentiality in large databases without degrading their performance is a challenging problem, especially when encryption and decryption must be performed at the database-level or at the application-level. We here focus on symmetric ciphers for database encryption since they are the only type of ciphers with acceptable performance for most applications. We point out that stream ciphers are the adequate type of encryption schemes. We present an attack on a dedicated stream cipher proposed by Ge and Zdonic in 2007.**

I. Introduction

With the recent emergence of database externalisation comes the problem of securing them. One of the main issues is to find a suitable way to guarantee the confidentiality of the involved data. This problem is crucial in the *database as a service* paradigm [1], where the data are stored on a host site. It is known to be a complicated problem when the encryption and decryption is performed either at the database-level or at the application-level [2]. Actually, it requires more than standard straightforward encryption to produce a queryable encrypted database with good performance. For instance, classical encryption schemes usually forbids the use of index on encrypted data.

Thus the database community proposed various database dedicated encryption schemes which allow fast search, such as *order-preserving* or *prefix-preserving* encryption. But they generally do not provide a strong protection as they tend to leak a lot of information. Some particular public-key encryption schemes provide interesting features [3], but their low throughput makes them unpractical for most applications. Therefore, other possibilities, like using well-known symmetric encryption algorithms with suitable modes of operation, have to be taken into account.

In this paper, we will first show that stream ciphers, including those based on a block cipher with a suitable mode of operation, are appropriate for database encryption. Then, we focus on a particular stream cipher, named FCE, proposed by Ge and Zdonic in [4] for database dedicated encryption, and we show that this cipher can be broken within a few minutes on a standard PC.

[1]This work was partially supported by the French Agence Nationale de la Recherche under Contract ANR-08-SEGI-007.

II. Modes of Operation for Databases Encryption

A. Expected Properties

To protect a database system with encryption, we have to find a trade-off between the performance of the queries and the security of the encryption. There have been a lot of encryption schemes proposed for database purpose, such as *order-preserving* or *prefix preserving* encryption. They have interesting properties for building indices on encrypted data, but they leak a lot of information about these data. For example, they preserve equality. Even if it is essential when it comes to building an index, it is catastrophic for encrypting the actual database, as it reveals the repartition of the data, which can lead to many associations of ciphertext/plaintext.

In cryptography, a reasonable encryption scheme is meant to be IND-CPA or IND-CCA [5], [6]. Of course, neither *order-preserving* or *prefix preserving* encryption is IND-CPA nor IND-CCA. Not being IND-CPA or IND-CCA is not the only issue with such encryption algorithms. They also both are deterministic, *i.e.* to one plaintext only corresponds one ciphertext. That is problematic because it reveals the repartition of the data and thus, in some cases, it can give a good idea about the meaning of a given ciphertext.

Another way to achieve the first goal is to use encryption algorithms that allow fast comparisons, *i.e.* comparisons that can be done with partial decryption, in other words *early stopping* comparisons. The concept of *early stopping* is described as follows. The comparison of two ciphertexts starts from the most significant byte, proceeds byte by byte and stops once a difference is found. Then, to tell the greater from the smaller, there is only to decrypt the two bytes that differed.

B. Suitable Encryption

To fulfil the above purpose, there are a wide range of possibilities available among well-known and well-studied encryption algorithms and modes of operation. We will focus on symmetric encryption as all known asymmetric encryption algorithms are clearly too expensive. Thus we can either use *block* or *stream ciphers*.

Early stopping can clearly be achieved by stream ciphers. FCE [4] belongs to this category. But, other strong and well-studied stream ciphers also exist. The eSTREAM project [7] studied such algorithms and proposed a portfolio with recommended stream ciphers, such as SOSEMANUK for software

implementation or Grain and Trivium for hardware implementation. The CBC mode of operation [8], [9] also allows fast comparison, as if the beginning of two data is identical, so is the beginning of their corresponding ciphertext. But, to decide which is the smaller, we have to decrypt all the 128-bit blocks until where the difference appears.

The use of stream ciphers allows fast comparisons but it raises some concerns about the updates in the database. If a user updates the byte i from a value m_i to m'_i, the corresponding encrypted value will change from $c_i = m_i \oplus s_i$ to $c'_i = m'_i \oplus s_i$. Thus, an attacker who has access to the encrypted database can obtain $c_i \oplus c'_i$, which is exactly $m_i \oplus m'_i$. To avoid this issue, s_i should not remain constant throughout the updates of the database but should depend on then. This issue is actually mentioned in the FCE description [4]. Their solution is to process updates by batch, and, for that purpose, generate a new stream for every impacted page. Thus, the stream will not only depend on the key and the page number, but it will also depend on another variable that needs to be stored (update number, update timestamp...).

The choice will also depend on how the index is handled. In [4], it is suggested to build the index on the encrypted database, but this choice has a big drawback. Thanks to the index, an attacker will be able to recover an ordered version of the encrypted table and thus the security of the encryption will be limited to the security of an order-preserving encryption. To avoid this undesirable property, the index has to be computed on the plain data and then encrypted with a different encryption scheme from the one used to encrypt the data in the database. But, as the queries to the database use the index and not the actual database to be processed, it is important that the index is encrypted with an encryption scheme allowing fast comparison; this property is not predominant for encrypting the database. Thus using a stream cipher from the eSTREAM portfolio [7] or a AES with the CTR mode is a suitable solution to encrypt the index, the IV can either use a timestamp, the update number or another unique indicator. AES with the CBC mode would then be appropriate to encrypt the database.

III. Description of FCE

A. Notation and Definitions

The FCE [4] encryption algorithm was built with an open-source column-oriented DBMS called C-Store [10] in mind. C-Store is indeed a read-optimized relational DBMS, which explains why the database updates is not the biggest issue here.

Before describing the encryption algorithm, we first define the notation we will use. The plaintext is divided in pages of p bytes $m_i,\ 0 \leq i < p$. The ciphertext is also divided in pages of p bytes, $c_i,\ 0 \leq i < p$. There is a unique k-bit length key K for the whole database. To each plaintext page corresponds a polynomial $P(x) = ax^3 + bx^2 + cx + d \bmod p$, derived from the page number j and the key K. The keystream will be computed from the polynomial P and the key K.

B. Algorithm

The encryption is done one page at a time. It requires an encryption algorithm E to generate the polynomial P from the key K and the page number j. The choice of E is not detailed in [4], but any standard cipher like AES can be used. We first need to define a notation before displaying the encryption algorithm (*cf.* Algo. 1).

Definition 1. We denote $K_{\{d_i \to d_i+7\}}$ the key byte starting at bit position d_i. The k-bit key is considered as a circular bit string, *i.e.* the bit positions are defined modulo k.

Algorithm 1 Fast Comparison Encryption (FCE)

Require:
- A page of plaintext (p bytes $\{m_i\}_{i\in\{0,\dots,p-1\}}$).
- The page number j.
- The k-bit length key K.

Ensure:
- The ciphertext page (p bytes $\{c_i\}_{i\in\{0,\dots,p-1\}}$).

// Generation of the polynomial for Page j
$(a, b, c, d) \leftarrow E(j, K)$ with $a, b, c, d \in [0, p-1]$
$P(x) = ax^3 + bx^2 + cx + d \bmod p$

// Encryption of this page
for all i from 0 to $p-1$ **do**
  $d_i = P(i) \bmod k$
  $c_i = m_i \oplus K_{\{d_i \to d_i+7\}}$
end for

Clearly, FCE is a synchronous additive stream cipher where the i-th keystream byte equals $K_{\{d_i \to d_i+7\}}$.

FCE is actually a practical but weaker version of r-FCE. The difference between them is that, instead of using a polynomial P derived from the key and the page number, r-FCE uses a random permutation of $\{0, \dots, p-1\}$. Thanks to that, the ideal algorithm r-FCE is proven to be INFO-CPA-DB, a new type of resistance to *chosen plaintext attacks* based on the notion of entropy, defined in [4] by Ge and Zdonik. But the authors stress out that the relationship between this notion and the resources in both time and space required to actually break the scheme remains unknown. Obviously, our attack does not work on r-FCE as it relies on ideal permutations.

But r-FCE is impractical as storing each permutation would require to store at least $\log_2(p!)$ random bits. For example, if $p = 64$ KBytes, $\log_2(p!) \sim 954037$, and this is clearly too much of an overhead. That is why the authors of [4] proposed to use polynomials of degree 3 instead of these permutations.

It is worth noticing that, in FCE, the authors do not consider the use of permutation polynomials, while the use of a permutation is mandatory in r-FCE. But it is actually very simple to ensure that the polynomials used to encrypt each page are permutation polynomials. Indeed, a polynomial $P(x) = ax^3 + bx^2 + cx + d$ modulo 2^m is a permutation polynomial if and only if a and b are even and c is odd [11].

C. Parameters and Security

As mentioned earlier, FCE was built with the open-source column-oriented DBMS called C-Store [10] in mind. Thus it

is interesting to consider the parameters used in the context of C-Store. They are the following:

- key size: $k = 2^{15}$ bits (32 Kbit),
- page size: $p = 2^{16}$ bytes (64 KBytes),
- (a, b, c, d) size: 64 bits per 64 KByte page.

In this case, the key size is quite big. Since the security level of a symmetric cipher usually corresponds to its key size (in other words, the exhaustive search for the key is the most efficient attack), it is interesting to find out what security gives us these parameters in the case of a known plaintext attack.

We consider that we have a page of plaintext and the corresponding page of ciphertext (we will see later that it is not necessary), and thus the keystream that consists of key bytes in the order determined by the polynomial. The first straightforward idea to retrieve the key is to perform an exhaustive search on all the possible (a, b, c, d). For every polynomial, we compute its values over $[0, p-1]$ and try to rebuild the key from the keystream. If all the pieces match, we then derive the polynomial from this key and this page number. If it is the one we are working with, the couple (key, polynomial) is the one we wanted to find. The cost of this is 2^{5p}, *i.e.* 2^{80} with the proposed parameter set, if we consider that the pieces of the keystream will only fit together with the right polynomial. This method is already much better than an exhaustive search for the 2^{15}-bit key and we see that the security parameter of FCE is not the size of its key.

IV. Cryptanalysis of FCE

The attack we now present is a *known plaintext attack*, as defined previously. To retrieve the whole keystream, we need a page of plaintext and the corresponding page of ciphertext. From them, we will retrieve the key, and thus all the permutations of the different pages much faster than with the previous exhaustive search method.

A. Relationship between Parameters p and k

In the proposed parameter set, the key size $k = 2^{15}$ equals half the number of bytes $p = 2^{16}$ in each page. Thus, the polynomial $P(x) = ax^3 + bx^2 + cx + d \bmod 2^{16}$ is actually only used modulo 2^{15}, as $d_i = P(i) \bmod 2^{15}$. Since $\forall i \in [0, 2^{15}-1],\ P(i+2^{15}) = P(i) \bmod 2^{15}$, we could have defined P modulo 2^{15} instead of modulo 2^{16}.

On top of that, this property gives $\forall i \in [0, 2^{16}-1],\ d_i = d_{i+2^{15}}$ and thus only half a page of keystream, *i.e.* 2^{15} bytes of both plain and ciphertext, is needed to retrieve the whole keystream, its second half being the exact same as the first. It implies that, if we only have a full page of ciphertext, we get the XOR of the plaintext. Indeed, $m_{i+2^{15}} \oplus m_i = c_{i+2^{15}} \oplus c_i$.

Of course, even without this unsuitable property, *i.e.* if we consider a modified version of FCE with $p = k$, our attack works, but then requires a whole page of plaintext and the corresponding page of ciphertext.

B. Principle of the Attack

In this part, we will use the notation $k = 2^\kappa$ for the key size, and thus $\kappa = 15$ for the FCE parameter set.

As we previously saw, it is much faster to perform an exhaustive search on the polynomials than on the key. The idea of the attack is to reduce the set of possible polynomials to do the search on.

The first reduction is made thanks to the following remark. For a given page, the couples of key and polynomial $\big(K' = K \ggg d', P'(x) = ax^3 + bx^2 + cx + (d - d')\big)$ are equivalent for all d', meaning that given a page of plaintext, any of these couples will result in the same page of ciphertext. Therefore, we will search for $\widetilde{K} = K \ggg d$ and $\widetilde{P}(x) = ax^3 + bx^2 + cx$ and we will focus on d only once we find $\widetilde{K}$ and $\widetilde{P}$.

Our problem is then the following. We want to retrieve (a, b, c) from the knowledge of $\widetilde{K}_{\{\widetilde{P}(i) \to \widetilde{P}(i)+7\}} = K_{\{P(i) \to P(i)+7\}}$ where $\widetilde{P}(i) = ai^3 + bi^2 + ci \bmod 2^\kappa$.

This problem will be eased thanks to the information given by $\widetilde{K}_{\{\widetilde{P}(i) \to \widetilde{P}(i)+7\}}$ on the preimages under $\widetilde{P}$ of successive elements. It actually gives us a set of triples (α, β, γ) among which all the preimages of $(1, 2, 3)$ by $\widetilde{P}$ are present; even though there will still remain some unsatisfying triples.

Then, for every possible triple (α, β, γ) in this set, we search, in the ring $\mathbb{Z}/2^\kappa\mathbb{Z}$, a solution of the following Vandermonde system:

$$\begin{pmatrix} \alpha^3 & \alpha^2 & \alpha \\ \beta^3 & \beta^2 & \beta \\ \gamma^3 & \gamma^2 & \gamma \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} 2^\kappa. \tag{1}$$

We can even drop all the triples where α or γ are even, as it is obvious from this system that they both have to be odd.

We now have to solve this Vandermonde system. In this purpose, let $M = \begin{pmatrix} \alpha^3 & \alpha^2 & \alpha \\ \beta^3 & \beta^2 & \beta \\ \gamma^3 & \gamma^2 & \gamma \end{pmatrix}$. Thus, we search for a solution to the following system, when M is given:

$$M \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} 2^\kappa.$$

Once multiplied by the adjugate matrix of M, ${}^tM^C$, we have:

$$\det M \begin{pmatrix} a \\ b \\ c \end{pmatrix} - {}^tM^C \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} 2^\kappa = {}^tM^C \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix},$$

that can be simplified as such:

$$\det M \begin{pmatrix} a \\ b \\ c \end{pmatrix} + 2^\kappa \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} m_a \\ m_b \\ m_b \end{pmatrix}.$$

Thus, each coefficient $\alpha \in \{a, b, c\}$ is a solution of an equation of the form $\det(M)\alpha + 2^\kappa\lambda = m_\alpha$ to solve. Such a system has a solution if and only if $\gcd(\det M,\ 2^\kappa)$ divides m_α. Let (u, v) denote the corresponding Bézout coefficients, *i.e.*, a pair of integers satisfying $\det(M)u + v2^\kappa = \gcd(\det M,\ 2^\kappa)$, when the system has a solution. Then, the solutions to the equation are:

$$\alpha = u\frac{m_\alpha}{\gcd(\det M,\ 2^\kappa)} + n\frac{2^\kappa}{\gcd(\det M,\ 2^\kappa)},$$

where $n \in \mathbb{Z}$.

This generally gives only a few solutions as, most of the time, the gcd is equal to 4. But it can grow bigger and then the search for (a, b, c) just gets too expensive. There are two ways to solve this issue. The first one consists in using preimages of $4, 5 \ldots$ as well and ensures to find a system that has fewer solutions. The second way is to simply ignore the triple (i, j, k) that leads to the big gcd. It works most of the times for it eliminates the triple (a, b, c) we are looking for with probability $\frac{1}{2^{45}}$. This probability being very low, we can do that at first as we only meet such triple (i, j, k) on average twice for every attack. If the attack fails, we can always go back to the previous case and test these triples.

We now have a set of possible values for (a, b, c) and can proceed with a search, as described previously, but limited to this smaller set.

C. Detailed Description of the Attack

The attack algorithm is detailed in Algo. 2. The first step consists in computing all $v_i = m_i \oplus c_i$. The obtained v_i then correspond to all the bytes of K starting respectively at position $P(0), P(1), \ldots, P(2^\kappa - 1)$ with $P(x) = ax^3 + bx^2 + cx + d \bmod 2^\kappa$, but without their ordering. That will allow us to determine a small set of triples (α, β, γ) among which are all the potential preimages of $(1, 2, 3)$ by $\widetilde{P}$.

If $\widetilde{P}$ is the wanted permutation, v_0 gives some information on α such that $\widetilde{P}(\alpha) = 1$. Actually the first 7 bits of v_α correspond to the last 7 bits of v_0 since we have $v_0 = \widetilde{K}_0, \widetilde{K}_1, \ldots, \widetilde{K}_7$ and $v_\alpha = \widetilde{K}_{\{\widetilde{P}(\alpha) \to \widetilde{P}(\alpha)+7\}} = \widetilde{K}_{\{1 \to 8\}} = \widetilde{K}_1, \ldots, \widetilde{K}_7, \widetilde{K}_8$.

That enables us to build three sets, $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ containing respectively the possible preimages of 1, 2 and 3 by $\widetilde{P}$. More precisely, for $x, y \in \{0, 1\}$, we define:

$$\begin{array}{lcl} \mathcal{E}_1 & = & \left\{\alpha \text{ odd } | v_{\alpha\{0\to6\}} = v_0 \gg 1\right\} \\ \mathcal{E}_2(x) & = & \left\{\beta \ | v_{\beta\{0\to6\}} = (v_0 \gg 2, x)\right\} \\ \mathcal{E}_3(x, y) & = & \left\{\gamma \text{ odd } | v_{\gamma\{0\to6\}} = (v_0 \gg 3, x, y)\right\} \end{array}$$

Their average sizes are given by the number of all possible values for the preimage divided by 2^7 since 7 bits of the corresponding keystream byte are known. For the given parameters, we then have $|\mathcal{E}_1| \sim 2^7$, $|\mathcal{E}_2(x)| \sim 2^8$, and $|\mathcal{E}_3(x, y)| \sim 2^7$.

The third step consists in resolving the Vandermonde systems given by the triples (α, β, γ) from $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$. That gives us a small set $\mathcal{L}$ of possible values for (a, b, c), in which we know that the triple we are looking for is.

In the fourth and last step we try to build the key corresponding to every possible triple, and, if we succeeded we search for the right d corresponding to this page.

D. Complexity

We now have to evaluate the cost of this attack. Table I sums up the cost of the 4 steps of the attack.

Algorithm 2 Attack on FCE

Require:
- Half a page of plaintext, $\{m_i\}_{i \in \{0,\ldots,2^\kappa-1\}}$.
- Half a page of ciphertext, $\{c_i\}_{i \in \{0,\ldots,2^\kappa-1\}}$.

Ensure:
- The key, the polynomials and the plaintext.

// Step 1: Keystream retrieval
for all i from 0 to $2^\kappa - 1$ **do**
  $v_i \leftarrow m_i \oplus c_i$
end for

// Step 2: Finding the preimages of $(1, 2, 3)$ by $\widetilde{P}$
$\mathcal{E}_1 = \left\{\alpha \text{ odd } | v_{\alpha\{0\to6\}} = v_0 \gg 1\right\}$
$\mathcal{E}_2(x) = \left\{\beta \ | v_{\beta\{0\to6\}} = (v_0 \gg 2, x)\right\}$
$\mathcal{E}_3(x, y) = \left\{\gamma \text{ odd } | v_{\gamma\{0\to6\}} = (v_0 \gg 3, x, y)\right\}$

// Step 3: Filtering the (a, b, c)
$\mathcal{L} \leftarrow \emptyset$
for all $\alpha \in \mathcal{E}_1$ **do**
  $x_\alpha \leftarrow$ msb bit of v_α
  for all $\beta \in \mathcal{E}_2(x_\alpha)$ **do**
    $y_\beta \leftarrow$ msb bit of v_β
    for all $\gamma \in \mathcal{E}_3(x_\alpha, y_\beta)$ **do**
      if system (1) has a solution (a, b, c) **then**
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c)\}$
      end if
    end for
  end for
end for

// Step 4: Reducing $\mathcal{L}$ and finding d
for all $(a, b, c) \in \mathcal{L}$ **do**
  if we succeed to build $\widetilde{K}$ **then**
    for all $d \in \{0, \ldots, 2^\kappa - 1\}$ **do**
      $K \leftarrow \left(\widetilde{K} \ggg d\right)$
      $\left(\widetilde{a}, \widetilde{b}, \widetilde{c}, \widetilde{d}\right) \leftarrow E(j, K)$ (j is the page nb)
      if $\left(\widetilde{a}, \widetilde{b}, \widetilde{c}, \widetilde{d}\right) == (a, b, c, d)$ **then**
        return K
      end if
    end for
  end if
end for

TABLE I
COST OF THE ATTACK

Step	Cost	for the proposed parameters
1	k XORs	2^{15}
2	k masks and comparisons	2^{15}
3	$\frac{1}{4}\left(\frac{k}{2^7}\right)^3$ 3×3 systems resolutions	2^{22}
4	k calls to the block cipher	2^{15}

The cost of the attack is thus 2^{15} calls to the block cipher algorithm plus $\sim 2^{25}$ multiplications on 16 bits.

As our attack shows, the assumption that FCE is as secure as any underlying block cipher is clearly wrong. Indeed, we see that the attack with any underlying block cipher, either a secure one as AES or a weaker one as DES, works with the same complexity, the difference being the encryption time of this algorithm.

E. Simulations

We launched 300 attacks on two distinct computers (150 on each). We adopted the method where we ignore the triples (i, j, k) when they give a gcd greater than 16. We got a 100% success. The number of ignored triples varied between 0 and 6 and is worth 1.85 on average. The time taken by the attack is displayed in Table II.

TABLE II
TIME OF THE ATTACKS (IN SECONDS)

Type of processor	Time		
	Min.	Max.	Av.
Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz	283 s	784 s	514 s
Intel(R) Xeon(R) CPU 5120 @ 1.86GHz	479 s	1295 s	828 s

V. CONCLUSIONS

Ge and Zdonik's work [4] shows that a fast comparison encryption scheme should be used for protecting the confidentiality of big databases, and especially that an additive stream cipher is particularly relevant in this case.

However the algorithm they proposed, FCE, albeit its great performances, does not ensure a sufficient security for one can recover the key in only a few minutes from the knowledge of 2^{15} bytes of plaintext and ciphertext with the suggested parameters set. We thus suggest to use either a standard encryption algorithm such as AES-CTR, or one of the stream cipher from the eSTREAM portfolio [7].

ACKNOWLEDGEMENTS

Many thanks to Anne Canteaut for her great and essential contribution to this work, to Matthieu Finiasz for his helpful ideas and to Luc Bouganim for his meaningful comments.

REFERENCES

[1] H. Hacigümüs, S. Mehrotra, and B. Iyer, "Providing database as a service," in *International Conference on Data Engineering - ICDE 2002*. IEEE Computer Society, 2002, pp. 29–39.
[2] L. Bouganim and Y. Guo, "Database encryption," in *Encyclopedia of Cryptography and Security*. Springer, 2010, 2nd Edition.
[3] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Advances in Cryptology - CRYPTO 2007*, ser. Lecture Notes in Computer Science, vol. 4622. Springer, 2007, pp. 535–552.
[4] T. Ge and S. Zdonik, "Fast, secure encryption for indexing in a column-oriented DBMS," in *International Conference on Data Engineering - ICDE 2007*. IEEE, 2007, pp. 676–685.
[5] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.
[6] M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*, 2005, http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html.
[7] ECRYPT - EUROPEAN NETWORK OF EXCELLENCE IN CRYPTOLOGY, "The eSTREAM Stream Cipher Project," http://www.ecrypt.eu.org/stream/, 2005.
[8] FIPS 81, "DES Modes of Operation," Federal Information Processing Standards Publication 81, 1980, U.S. Department of Commerce/National Bureau of Standards.
[9] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *FOCS*, 1997, pp. 394–403.
[10] http://db.csail.mit.edu/projects/cstore/.
[11] R. L. Rivest, "Permutation polynomials modulo 2^w," in *Finite Fields and their Applications*, vol. 7, 1999, pp. 287–292.