Cryptanalysis of a Fast Encryption Scheme for Databases and of its Variant

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, and on a variant of this encryption scheme.**

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 Zdonik 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. Databases Encryption Goals

A. Expected Properties

Before going any further, we first should detail what encryption is expected to achieve. Generally, encryption is a trade-off between three main expectations:

- to prevent data leaking, *i.e.* to ensure that without the encryption key it is impossible to retrieve any information about the plain text from the cipher text;
- to detect data falsification, *i.e.* if the cipher text has been tampered with, any key holder should be able to notice this modification;
- to limit both size and speed overhead, *i.e.* neither the encrypted data should be much bigger than the plain data, or the encryption and decryption process should take too long.

On top of that, database encryption requires that the encryption does not prevent queries on the database, *i.e.* the encryption of either the database or the index should not alter too much the database structure. The encryption scheme should also allow updates such as data modification, addition and deletion. Thus the trade-off between security and usability is much harder to achieve than with standard encryption.

Here we will only focus on preventing data leaking and we will not consider the problem of data integrity.

B. Fast Comparison Encryption

In order to allow fast queries, either over an encrypted database or an encrypted index, the encryption scheme used should allow fast comparisons. Ge and Zdonik explain in [4] that a way to perform fast comparisons is to use an algorithm that permits comparisons that can be done with partial decryption. They call it *early stopping* comparisons. This 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 distinguish the greater from the smaller, it has only to decrypt the two bytes that differed.

C. Naïve but Unsuitable Examples

From the previous remarks, we can easily build naïve encryption schemes that achieves one of our goals. But finally we will see that these naïve algorithms are not at all suitable.

1) Sure but Impracticable Encryption:
A first idea to ensure a strong encryption is to directly encrypt the whole database. For example, this can be done by encrypting the database file using AES in an appropriate mode (CBC...). Even though the security is obvious, so are its drawbacks: to perform a query the whole database has to be decrypted, updating the database requires a full decryption and re-encryption. This is of course out of question.

2) Fast but Insecure Encryption:
On the other side, some fast encryption schemes, which are suitable for database queries, have been proposed and studied by both cryptographic and database communities. *Order-preserving* and *prefix preserving* encryption are such encryption schemes specially proposed to encrypt databases.

They both have interesting properties for building indices on encrypted data, but they leak a lot of information about these data. Also they both are deterministic, *i.e.* to one plaintext only one ciphertext corresponds. Thus, they preserve equality, *i.e.*two identical plaintexts correspond to two identical ciphertexts. 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 many cases, this can give a good idea about the meaning of a given ciphertext.

Furthermore, neither *order-preserving* nor *prefix preserving* encryption is IND-CPA or IND-CCA. This can be proved easily using the definitions of the previous section. For example, in the case of *order-preserving* encryption, let the attacker first chose the messages 4 and 8. She then only has to require the encryption of 6 and compare it to the ciphertext she got to be able to tell if the ciphertext corresponds to 4 or 8.

But these are not the only issues with these encryption algorithms. Performing data addition and deletion is generally not feasible with them and neither is a single data modification in most cases. This is a huge drawback for most database applications.

Thus we saw that, even though achieving a single of our requirements for database encryption at a time is easy, combining them seams much harder.

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 [5] 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. The ciphertext is also divided in pages of p bytes c_i. There is a unique k-bit length key K for the whole database. To each plaintext page corresponds a permutation 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 permutation 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 permutation 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 permutation 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 security against *chosen plaintext attack* 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 permutation polynomials of degree 3 instead of these permutations.

It is worth noticing that, in FCE, the authors do not fully explain how to compute a permutation polynomial. 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 [6].

C. Parameters and Security

As mentioned earlier, FCE was built with the open-source column-oriented DBMS called C-Store [5] 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 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 for the polynomials than for 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 $\left(K' = K \ggg d', P'(x) = ax^3 + bx^2 + cx + (d - d')\right)$ 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. $\widetilde{K}$ 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. The existence of the required preimages is ensured by the fact that the polynomial is a permutation.

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_alpha. 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. The probability of eliminating the good triple (a, b, c) is only $\frac{1}{2^{45}}$. 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 $v_\alpha = K_{\{\widetilde{P}(\alpha) \to \widetilde{P}(\alpha)+7\}} = K_{\{1 \to 8\}} = \widetilde{K}_1, \ldots, \widetilde{K}_7, \widetilde{K}_8$.

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 $\widetilde{P}$. 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 solving 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.

TABLE I
COST OF THE ATTACK

Step	Cost	for the proposed parameters
1	k XORs	2^{15}
2	$7k$ masks and comparisons	7.2^{15}
3	$\left(\frac{k}{2^7}\right)^3$ 3×3 systems resolutions	2^{22}
4	k calls to the block cipher	2^{15}

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 P
$\mathcal{E}_1 = \{\alpha \text{ odd } | v_{\alpha\{0 \to 6\}} = v_0 \gg 1\}$
$\mathcal{E}_2(x) = \{\beta \, | v_{\beta\{0 \to 6\}} = (v_0 \gg 2, x)\}$
$\mathcal{E}_3(x, y) = \{\gamma \text{ odd } | v_{\gamma\{0 \to 6\}} = (v_0 \gg 3, x, y)\}$

// 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 the 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, 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

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. CRYPTANALYSIS OF FCE VARIANT

A. FCE variant

The authors of FCE also suggest a variant of this encryption scheme [7], which limits the value of the number of bytes per page, p to powers of a prime number; for obvious reasons we focus on $p = 2^n$. Let Q be an irreducible polynomial over $GF(2^n)$ and ω be a root of Q. Then there is a natural bijection ϕ between $[0, 2^n - 1]$ and $GF(2^n)$: to the element $x = \sum_{i=0}^{n-1} x_i 2^i$ of $[0, 2^n - 1]$ corresponds the element $x = \sum_{i=0}^{n-1} x_i \omega^i$ of $GF(2^n)$, where $\forall i \in [0, n-1],\ x_i \in \{0, 1\}$. Moreover, for any integers i and j, we have

$$\phi(i \oplus j) = \phi(i) + \phi(j) \tag{2}$$

where $\oplus$ denotes the bitwise exclusive or when integers are identified with n-bit words, and $+$ denotes the addition over the finite field $GF(2^n)$. P is then defined as a polynomial of degree 3 over $GF(2^n)$ and the modulo k is taken on the value of $P(x)$ mapped to $[0, 2^n - 1]$, *i.e.* on $\phi^{-1}(P(x))$.

More precisely, the encryption procedure in Algo. 1 is now:

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

This FCE variant using polynomials over $GF(2^n)$ has some implications on the cryptanalysis, which thus has to be modified accordingly into the cryptanalysis we now describe. We will here retrieve the key, and thus all the permutations of all the pages, from a full page of both plaintext and ciphertext.

B. Principle of the Attack

In this part, we will use the notation $k = 2^\kappa$ for the key size and $p = 2^n$, and thus $\kappa = 15$, and $n = 16$ 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.

Here, the constant coefficient d of P plays a particular role. Indeed, for a given page, for $P(x) = \widetilde{P}(x) + d$, with $d = P(0)$, we deduce from (2) that

$$d_i = \phi^{-1}(P(\phi(i))) = \phi^{-1}\left(\widetilde{P}(\phi(i))\right) \oplus \phi^{-1}(d) = \widetilde{d}_i \oplus d_0.$$

Our problem is then the following. We want to retrieve (a, b, c) from the knowledge of $K_{\{d_i \to d_i + 7\}} = \widetilde{K}_{\left\{\widetilde{d}_i \oplus d_0 \to \left(\widetilde{d}_i \oplus d_0\right) + 7\right\}}$. Actually, we will show that (a, b, c) can be recovered from the keystream by an exhaustive search on the least significant bits of d_0.

The knowledge of all $\widetilde{K}_{\left\{\widetilde{d}_i \oplus d_0 \to \left(\widetilde{d}_i \oplus d_0\right) + 7\right\}}$ provides us some information on the preimages under $\widetilde{P}$ of "successive" elements. For a given value of d_0, it actually gives us sets of triples (α, β, γ), among which all the preimages of $(1, \omega, 1+\omega)$ by $\widetilde{P}$ are present.

We want to find some sets of possible preimages for $(1, \omega, 1 + \omega)$. Indeed, the following table shows the relations between $d_0 = \phi^{-1}(d)$ modulo 4 and the values of $\phi^{-1}(1 + d)$, $\phi^{-1}(\omega + d)$ and $\phi^{-1}(1 + \omega + d)$.

$d_0 \mod 4$	$\phi^{-1}(1 + d)$	$\phi^{-1}(\omega + d)$	$\phi^{-1}(1 + \omega + d)$
0	$d_0 + 1$	$d_0 + 2$	$d_0 + 3$
1	$d_0 - 1$	$d_0 + 2$	$d_0 + 1$
2	$d_0 + 1$	$d_0 - 2$	$d_0 - 1$
3	$d_0 - 1$	$d_0 - 2$	$d_0 - 3$

Thus, depending on the last two bits of d_0, the preimages of $(1, \omega, 1 + \omega)$ will be in one of the following sets.

$$\begin{aligned}
\mathcal{E}_{-3}(t, u) &= \left\{\delta \mid v_{\delta\{1\to 7\}} = \left(u, t, v_{0\{0\to 4\}}\right)\right\} \\
\mathcal{E}_{-2}(t) &= \left\{\delta \mid v_{\delta\{1\to 7\}} = \left(t, v_{0\{0\to 5\}}\right)\right\} \\
\mathcal{E}_{-1} &= \left\{\delta \mid v_{\delta\{1\to 7\}} = v_{0\{0\to 6\}}\right\} \\
\mathcal{E}_{1} &= \left\{\delta \mid v_{\delta\{0\to 6\}} = v_{0\{1\to 7\}}\right\} \\
\mathcal{E}_{2}(x) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(v_{0\{2\to 7\}}, x\right)\right\} \\
\mathcal{E}_{3}(x, y) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(v_{0\{3\to 7\}}, x, y\right)\right\}
\end{aligned}$$

When trying to build the key K we will pick the elements in the proper set, according to the previous table. The average size of these sets is 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 $|\mathcal{E}_i| \sim 2^8$ for all i.

Then, for every possible value of $d_0 \mod 4$, we solve the Vandermonde systems given by the triples (α, β, γ) in $GF(2^n)$ from the sets $\mathcal{E}_i$ determined by $d_0 \mod 4$:

$$\begin{pmatrix} \alpha^3 & \alpha^2 & \alpha \\ \beta^3 & \beta^2 & \beta \\ \gamma^3 & \gamma^2 & \gamma \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ \omega \\ 1 + \omega \end{pmatrix} \tag{3}$$

That gives us 4 sets $\mathcal{L}_i$ of possible values for (a, b, c). One solution is to directly build a larger set $\mathcal{L} = \bigcup_{i \in \{0 \ldots 3\}} \{(a, b, c, i) | (a, b, c) \in \mathcal{L}_i\}$. Another idea is to build a fourth set $\mathcal{E}_4$, computed as $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$, in which is stored elements j such as $P(j) = 4$ is plausible from the overlapping properties of the keystream. Then we search for an element j of $\mathcal{E}_4$ such as $\phi^{-1}(\widetilde{P}(\phi(j))) \mod 4 = i$, where $\widetilde{P}$ is derived from $(a, b, c) \in \mathcal{L}_i$. This will reduce the size of $\mathcal{L}$ slightly but not significantly.

Eventually, we go throughout $\mathcal{L}$ and for every 4-tuple (a, b, c, δ), with $0 \leq \delta \leq 3$, we try to rebuild the key K for every d such that $\phi^{-1}(d) \equiv \delta \bmod 4$. If we succeed, we finally only have to check if the 4-tuple (a, b, c, d) corresponds to the 4-tuple derived from the key we just build and the page number.

The attack algorithm on FCE variant is detailed in Algo. 3.

Algorithm 3 Attack on FCE variant - Part I

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

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

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

// Step 2: Compute the sets for all $(x, y, u, t) \in GF(2^n)$

$$\begin{aligned}
\mathcal{E}_{-3}(t,u) &= \left\{\delta \mid v_{\delta\{0\to6\}} = \left(u, t, v_{0\{0\to4\}}\right)\right\}\\
\mathcal{E}_{-2}(t) &= \left\{\delta \mid v_{\delta\{1\to7\}} = \left(t, v_{0\{0\to5\}}\right)\right\}\\
\mathcal{E}_{-1} &= \left\{\delta \mid v_{\delta\{1\to7\}} = v_{0\{0\to6\}}\right\}\\
\mathcal{E}_{1} &= \left\{\delta \mid v_{\delta\{1\to7\}} = v_{0\{1\to7\}}\right\}\\
\mathcal{E}_{2}(x) &= \left\{\delta \mid v_{\delta\{0\to6\}} = \left(v_{0\{2\to7\}}, x\right)\right\}\\
\mathcal{E}_{3}(x,y) &= \left\{\delta \mid v_{\delta\{0\to6\}} = \left(v_{0\{3\to7\}}, x, y\right)\right\}
\end{aligned}$$

C. Complexity

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

TABLE III
COST OF THE ATTACK ON FCE VARIANT

Step	Cost	Experiments
1	p XORs	2^{16}
2	$13p$ masks and comparisons	13.2^{16}
3	$4\left(\frac{p}{2^7}\right)^3 3\times 3$ systems resolutions (bitwise operations)	2^{29}
4	$\lvert\mathcal{L}\rvert p = 2^{16}\left(\frac{p}{2^7}\right)^3$ rebuilding of the key	2^{43}

In practice, there is also only one call to the block cipher, as the probability of properly rebuilding a wrong key is extremely low. Thus the main cost is the 2^{43} attempts to rebuild the key.

This is due to the fact that we cannot decorrelate the search on d from the search on the triple (a, b, c) in this variant. But the cost of the attack is still much lower than the cost of the naïve attack, which consists here in trying the rebuild the key for every possible 4-tuple (a, b, c, d) and thus costs 2^{64} attempts to rebuild the key.

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.

Algorithm 4 Attack on FCE variant - Part II

// Step 3: Filtering the (a, b, c)
// Case $d_0 \bmod 4 = 0$
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**
      Solve system (3)
      for all $i \in \mathcal{E}_4$ **do**
        if $ai^3 + bi^2 + ci \bmod 4 = 0$ **then**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c, 0)\}$
          continue
        end if
      end for
    end for
  end for
end for

// Case $d_1 \bmod 4 = 1$
for all $\alpha \in \mathcal{E}_{-1}$ **do**
  for all $\gamma \in \mathcal{E}_1$ **do**
    $x_\gamma \leftarrow$ msb bit of v_γ
    for all $\beta \in \mathcal{E}_2(x_\gamma)$ **do**
      Solve system (3)
      for all $i \in \mathcal{E}_4$ **do**
        if $ai^3 + bi^2 + ci \bmod 4 = 1$ **then**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c, 1)\}$
          continue
        end if
      end for
    end for
  end for
end for

D. Simulations

Here we cannot perform as many simulations as in the previous FCE variant as the complexity is much higher here: on the tests we performed, in took between 50 minutes in the best case and 8 days in the worst case.

Up to step 4, the attack algorithm is quite fast and takes less than a minute. The slowest part of the algorithm is clearly the 5^{th} step. Our simulations show that the worst case scenario, *i.e.* when both (a, b, c) and d are at the end of their list of possibilities, take less than 60 days. But these results could benefit from software optimisations, and from parallelization.

Therefore, even though this variant is more robust, it does not provide a satisfactory protection against our attack.

VI. 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

Algorithm 5 Attack on FCE variant - Part III

```
// Case d_2 mod 4 = 2
for all γ ∈ E_{-1} do
  x_γ ← lsb bit of v_γ
  for all β ∈ E_{-2}(x_γ) do
    for all α ∈ E_1 do
      Solve system (3)
      for all i ∈ E_4 do
        if a i^3 + b i^2 + c i mod 4 = 2 then
          L ← L ∪ {(a, b, c, 2)}
          continue
        end if
      end for
    end for
  end for
end for

// Case d_3 mod 4 = 3
for all α ∈ E_{-1} do
  x_α ← lsb bit of v_α
  for all β ∈ E_{-2}(x_α) do
    y_β ← lsb bit of v_β
    for all γ ∈ E_{-3}(x_α, y_β) do
      Solve system (3)
      for all i ∈ E_4 do
        if a i^3 + b i^2 + c i mod 4 = 3 then
          L ← L ∪ {(a, b, c, 3)}
          continue
        end if
      end for
    end for
  end for
end for

// Step 5: Reducing L and finding d
for all (a, b, c, δ) ∈ L do
  for all d such that d mod 4 = δ do
    if we succeed to build K then
      (a', b', c', d') ← E(j, K) (j, the page nb)
      if (a', b', c', d') == (a, b, c, d) then
        return K
      end if
    end if
  end for
end for
```

recover the key from the knowledge of at most 2^{16} 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 [8].

ACKNOWLEDGEMENT

Many thanks to Anne Canteaut for her great and essential contribution to this work, to Matthieu Finiasz for his helpful ideas, to Luc Bouganim for his meaningful comments and to Yanli Guo for her useful state of the art.

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] http://db.csail.mit.edu/projects/cstore/.

[6] R. L. Rivest, "Permutation polynomials modulo 2^w," in *Finite Fields and their Applications*, vol. 7, 1999, pp. 287–292.

[7] T. Ge, 2010, Private Conversation.

[8] ECRYPT - EUROPEAN NETWORK OF EXCELLENCE IN CRYPTOLOGY, "The eSTREAM Stream Cipher Project," http://www.ecrypt.eu.org/stream/, 2005.

[9] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[10] M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*, 2005, http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html.

[11] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *FOCS*, 1997, pp. 394–403.

[12] A. Menezes, P. van Oorshot, and S. Vanstone, *Handbook of applied cryptography*. CRC Press, 1997, http://www.cacr.math.uwaterloo.ca/hac/.