HFE and BDDs: A Practical Attempt at Cryptanalysis

Jean-Francis Michon, Pierre Valarcher, and Jean-Baptiste Yunès

Abstract. HFE (Hidden Field Equations) is a public key cryptosystem using univariate polynomials over finite fields. It was proposed by J. Patarin in 1996. Well chosen parameters during the construction produce a system of quadratic multivariate polynomials over $\mathbb{F}_2$ as the public key. An enclosed trapdoor is used to decrypt messages. We propose a ciphertext-only attack which mainly consists in satisfying a boolean formula. Our algorithm is based on BDDs (Binary Decision Diagrams), introduced by Bryant in 1986, which allow to represent and manipulate, possibly efficiently, boolean functions. This paper is devoted to some experimental results we obtained while trying to solve the Patarin's challenge. This approach was not successful, nevertheless it provided some interesting information about the security of HFE cryptosystem.

1. Introduction

1.1. BDDs

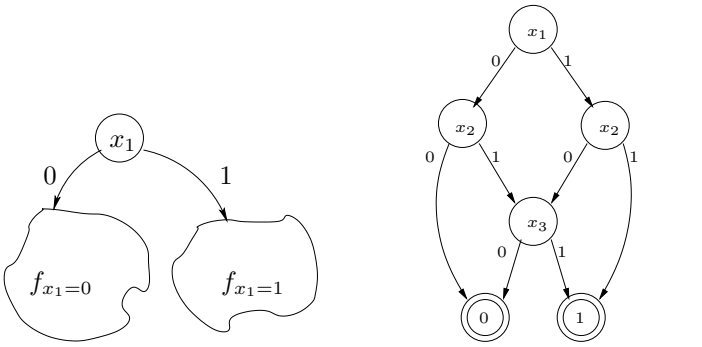
Binary Decision Diagrams (BDDs), introduced in [1], are now commonly used in CAD design or verification as tools to represent and manipulate efficiently boolean functions. They have been recently used in cryptography to cryptanalyze some keystream generators (see [5]).

The construction of the BDD of a boolean function f over n-variables labeled $x_1, \ldots, x_n$, noted $BDD(f)$, is obtained using the following Shannon's decomposition:

$$f(x_1, \ldots, x_n) = x_1.f(1, x_2, \ldots, x_n) + \neg x_1.f(0, x_2, \ldots, x_n)$$

Iterating this decomposition leads to a tree decision diagram in which inner nodes are labeled by variables, leaf nodes by boolean constants and directed edges are rooted from variables and labeled by a boolean assignment (see figure 1(a)). Each boolean assignment of the variables defines a unique path from the root to a boolean constant in the tree, and that constant gives the valuation of the function for the given assignment.

This work was supported by the "ACI Cryptologie" program of the French "Ministère de la Recherche".



(a) A tree decision diagram (b) $OBDD(x_1x_2 + x_1x_3 + x_2x_3)$

By choosing a graph isomorphism and by iterating it to the subgraphs of the tree, one obtains a unique acyclic directed graph, called OBDD (Ordered BDD or Oblivious Free BDD in [10]). The figure 1(b) is an example of such a graph obtained by: decomposing the function (x_1 first, then x_2 and then x_3), merging identical subgraphs and finally deleting useless nodes.

The size of $OBDD(f)$ is the number of its nodes and is denoted by $|OBDD(f)|$, this number is also the OBDD-complexity of the boolean function, denoted by $C_{obdd}(f)$.

One must note that the resulting graphs (form and size) are variable orderings dependent (in the Shannon's decomposition). Given a function, finding the variable ordering that gives the minimal OBDD size is a NP-complete problem.

We define a boolean hard function of n variables as a function whose OBDD-complexity $C(n) = \max_{f\in\mathbb{B}_n} C_{obdd}(f)$ is maximal over the set $\mathbb{B}_n$ of all n-ary boolean functions. We have:

$$\begin{cases} \underline{\lim}_{n\to\infty} \frac{n.C(n)}{2^n} &= 1 \\ \overline{\lim}_{n\to\infty} \frac{n.C(n)}{2^n} &= 2 \end{cases}$$

The practical construction of an $OBBD(f)$ where $f = g \oplus h$ (where $\oplus$ represents any boolean operator) is commonly obtained by application of an algorithm based on a parallel DAG traversal method (called APPLY, see [1]) over $OBDD(g)$ and $OBDD(h)$:

$$OBBD(f) = OBDD(g \oplus h) = APPLY(OBDD(g), \oplus, OBDD(h))$$

1.2. HFE Cryptosystems

The HFE (Hidden Field Equations) public key cryptosystem is based on polynomials over finite fields. This system has been proposed by J. Patarin (see [8]) who

showed how insecure Matsumoto & Imai schema was (see [6]) and then reinforced it. This method can be described as follows.

Let $\mathbb{K}$ be an extension of degree n over the finite field $\mathbb{F}_2$. The field $\mathbb{K}$ can be viewed as a n-dimensional vector space over $\mathbb{F}_2$. Any basis $\{\omega_1, \ldots, \omega_n\}$ of $\mathbb{K}$ defines the following mapping from $\mathbb{K}$ to $(\mathbb{F}_2)^n$:

$$\sum_{i=1}^{n} x_i\omega_i \longleftrightarrow (x_1, \ldots, x_n), \text{ where } x_i \in \mathbb{F}_2. \tag{1}$$

Any quasi-quadratic univariate polynomial $P(X)$ over $\mathbb{K}$

$$P(X) = \sum_{i,j}^{1..r} \alpha_{i,j} X^{2^i+2^j} + \gamma, \text{ with } \alpha_{i,j} \in \mathbb{K} \text{ and } \gamma \in \mathbb{K},$$

can be viewed in $(\mathbb{F}_2)^n$ as the collection of n quasi-quadratic multivariate polynomials over $\mathbb{F}_2$:

$$\begin{cases} P_1(x_1, \ldots, x_n) &= \sum_{j,k}^{1..n} \beta_{1,j,k} x_j x_k + \delta_1 \\ &\vdots & \text{with } \beta_{i,j,k} \in \mathbb{F}_2 \text{ and } \delta_i \in \mathbb{F}_2 \\ P_n(x_1, \ldots, x_n) &= \sum_{j,k}^{1..n} \beta_{n,j,k} x_j x_k + \delta_n \end{cases}$$

Such a system is the public key of HFE.

Encryption is done applying the P_i's on cleartext bits. Decryption is suspected to be as hard as the problem of solving multivariate quadratic boolean equations system is known to be NP-hard (see [2]). But it is also known to be efficiently tractable using Berlekamp's algorithm to extract the roots of $P(X)$ (only if its degree is not too large, see [3]).

Moreover two permutations S and T of $\mathbb{K}$ are introduced as trapdoors. It is important to note at that point that S and T are polynomials over $\mathbb{K}$ and that their expressions over $\mathbb{F}_2$ are invertible linear transformations. The public key is then defined as the expression of $T(P(S(X)))$ over $\mathbb{F}_2$ and it is still a quasi-quadratic multivariate polynomials as before. But now, the polynomial TPS has a very high degree (comparable to the order of the field), the number of its coefficients is much greater than those of P, and is now practically unsolvable if S and T are not known.

It is possible to define such cryptosystems on different fields, but using $\mathbb{F}_2$ allows to encrypt text using simple digital devices as smart cards. The parameter r, which controls the degree of P, must actually be less than 12 (see [4]) to allows Berlekamp's algorithm to run efficiently. To improve his cryptosystem, Patarin published a challenge in which $\mathbb{K} = \mathbb{F}_{2^{80}}$, such a public key size is about 32KB.

1.3. Attack

We chose a ciphertext-only attack, which consist in retrieving the cleartext given only the ciphertext and the public key. We decided not to use algebraic properties of the system. The problem is then reduced to satisfying a boolean formula.

Given a cleartext $X_0 \in \mathbb{K}$, encryption gives $Y_0 = TPS(X_0)$ so breaking the cipher correspond to solving:

$$Y_0 = TPS(X)$$

Using the mapping defined in (1), this can be rewritten as:

$$\begin{cases} y_{0,1} &= P_1(x_1, \ldots, x_n) \\ & \vdots \\ y_{0,n} &= P_n(x_1, \ldots, x_n) \end{cases}$$

then as:

$$\begin{cases} 1 &= 1 + y_{0,1} + P_1(x_1, \ldots, x_n) \\ & \vdots \\ 1 &= 1 + y_{0,n} + P_n(x_1, \ldots, x_n). \end{cases}$$

So, satisfying the formula:

$$R(x_1, \ldots, x_n) = \bigwedge_{i=1}^{n} E_i(x_1, \ldots, x_n)$$

where $E_i(x_1, \ldots, x_n) = 1 + y_i + P_i(x_1, \ldots, x_n)$, solves the problem and gives a possible cleartext.

We decided to use OBBDs to manipulate the boolean functions involved in the process and the algorithm is the following:

1. Construct $B_{E_i} = OBDD(E_i)$, for all $i \in [1, n]$,
2. Construct $B_{R_1} = B_{E_1}$,
3. Construct $B_{R_i} = APPLY(B_{R_{i-1}}, \wedge, B_{E_i})$, for all $i \in [2, n]$,
4. Extract the set of all satisfying paths in $B_R = B_{R_n}$.

2. The Library: LiBDD

We first tried to use CMU's library (see [11]) to solve the system. But as performances were not satisfactory, we coded our own library to tune up parameters of the implementation (this be shown later in this paper). Written in C, the library has less than 2,000 lines of source code and nothing but data layout is machine-dependent, however porting onto different platforms is quite easy (and was experimentally done). We then focused on some critical points of practical algorithmics: space and time.

2.1. Memory Management

The general-purpose BDD libraries we looked at commonly use a 96 or 128 bits long structure to code a single BDD-node. One can remark that we only need a small set of variables (actually 80 in the challenge), so we were able to pack a node into 64 bits using the following structure:

```
struct node {
  UINT64 mark : 1; // mark and sweep garbage collector
  UINT64 idx  : 7; // variable's index (0<=idx<127)
  UINT64 pthen:28; // 'true' sub-BDD
  UINT64 pelse:28; // 'false' sub-BDD
};
```

The fields named `pthen` and `pelse` contain the significant parts of the "real" memory pointers of the respectives true and false sub-BDDs. As each of these fields is 28 bits long, a realtime expansion must be done on each reference to produce a full 32 bits pointer. A generic coding of this is possible, however a machine-dependent representation is preferable because computation can be faster.

This implementation gave us the ability to manage boolean functions of at most 128 variables, and BDD size less than 2^{28} nodes. One can note that 2^{28} nodes of 2^3 bytes each need 2 giga-bytes to be stored (a common computer maximum memory size at this time).

The overall memory management has been kept as simple as possible, hence a single continous pool of nodes is allocated at startup time from which, nodes are picked as needed. Clearing an index frees the node but nodes of BDDs are freed when the BDD is freed and the synchronous mark and sweep garbage collector is called later.

2.2. Algorithms and BDD Variant

We used a mixed form of Bryant's algorithms `Apply` and `Reduce` (see [1]) which allows to allocate only useful nodes of the resulting BDD when applying an operator on two BDDs (see [10]). The main difficulty was the tricky management of some auxiliary structures (basically matrices) involved. As such matrices are mostly sparse, we used a mix of hashed and sorted tables.

As we constructed really huge BDDs, we decided to use a variant of BDDs. Recall that the subgraph identification operation in the `Reduce` step of the algorithm can use any suitable graph isomorphism. So we choose to identify two graphs either if they are denoted by the same function f, or if one is denoted by f and the other by $\neg f$. This is known as "BDD with output inverters" (see [7]) and only needs some slight modifications in algorithms and structures. One of the 28 bits devoted to pointers is now used to encode the "negation" mark. The maximum number of nodes decreased to 2^{27} but we expected to identify many more subgraphs and reduce the overall needed memory size.

2.3. Performances

We compared three different BDD libraries to solve the HFE problems we generate. As we can see in figure 1, the library LiBDD is slightly better than CuDD (VLSI/CAD Laboratory of Colorado University - see [12]), which is far better than CMU's.

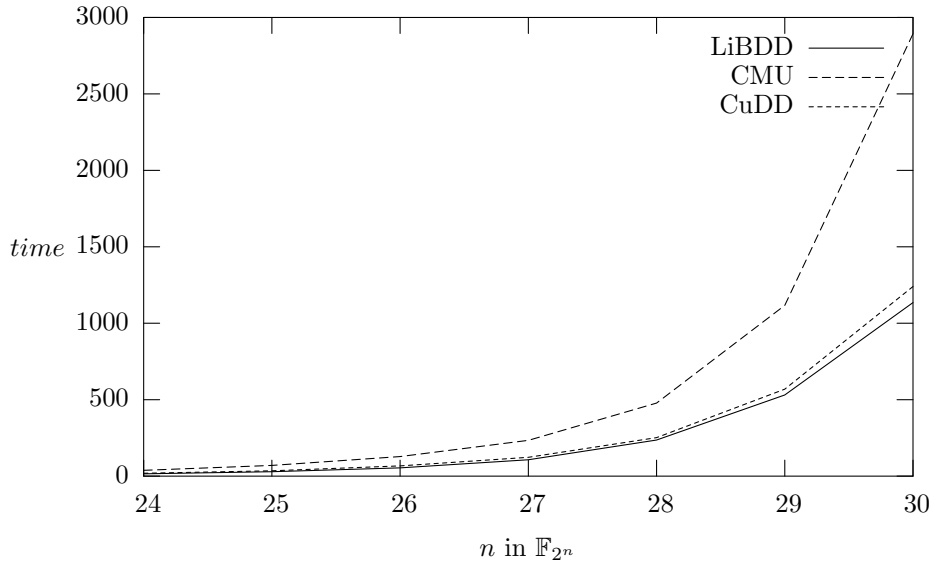


FIGURE 1. Comparing libraries on HFE systems

3. Experimental Results

As we needed many different HFE cryptosystems, we designed a program to generate such systems that was written in `C++` and uses Shoup's NTL library (Number Theory Library - see [14]).

The input of the main program is a HFE public key in challenge's format. The following is one equation taken from a system generated from a quasi-quadratic polynomial of degree 2,304=2^{11}+2^8 over $\mathbb{F}_{2^{24}}$:

$$y_1 = 1 + x_1 + x_3 + x_4 + x_7 + x_{10} + x_{11} + x_{12} + x_{17} + x_1x_2 + x_1x_3 + x_1x_5 + x_1x_{10} + x_1x_{11} + x_1x_{13} + x_1x_{14} + x_1x_{17} + x_1x_{21} + x_1x_{22} + x_1x_{23} + x_1x_{24} + x_2x_3 + x_2x_6 + x_2x_7 + x_2x_9 + x_2x_{13} + x_2x_{14} + x_2x_{15} + x_2x_{19} + x_2x_{20} + x_2x_{22} + x_2x_{24} + x_3x_6 + x_3x_7 + x_3x_8 + x_3x_9 + x_3x_{15} + x_3x_{16} + x_3x_{18} + x_3x_{21} + x_3x_{23} + x_3x_{24} + x_4x_9 + x_4x_{10} + x_4x_{11} + x_4x_{16} + x_4x_{17} + x_4x_{18} + x_4x_{20} + x_4x_{21} + x_4x_{22} + x_4x_{24} + x_5x_7 + x_5x_8 + x_5x_{10} + x_5x_{11} + x_5x_{13} + x_5x_{15} + x_5x_{17} + x_5x_{19} + x_6x_7 + x_6x_8 + x_6x_{12} + x_6x_{13} + x_6x_{16} + x_6x_{19} + x_6x_{22} + x_6x_{24} + x_7x_{10} + x_7x_{12} + x_7x_{15} + x_7x_{16} + x_7x_{18} + x_8x_{12} + x_8x_{15} + x_8x_{18} + x_8x_{19} + x_8x_{22} + x_8x_{23} + x_9x_{12} + x_9x_{13} + x_9x_{14} + x_9x_{22} + x_9x_{24} + x_{10}x_{15} + x_{10}x_{18} + x_{10}x_{19} + x_{10}x_{20} + x_{10}x_{22} + x_{10}x_{24} + x_{11}x_{13} + x_{11}x_{19} + x_{11}x_{21} + x_{11}x_{24} + x_{12}x_{13} + x_{12}x_{17} + x_{12}x_{18} + x_{12}x_{19} + x_{12}x_{22} + x_{12}x_{24} + x_{13}x_{14} + x_{13}x_{15} + x_{13}x_{18} + x_{13}x_{19} + x_{13}x_{20} + x_{13}x_{21} + x_{13}x_{22} + x_{13}x_{23} + x_{13}x_{24} + x_{14}x_{15} + x_{14}x_{21} + x_{14}x_{22} + x_{14}x_{23} + x_{15}x_{16} + x_{15}x_{17} + x_{15}x_{20} + x_{15}x_{21} + x_{15}x_{22} + x_{15}x_{23} + x_{16}x_{20} + x_{16}x_{21} + x_{16}x_{22} + x_{16}x_{23} + x_{17}x_{19} + x_{17}x_{20} + x_{17}x_{22} + x_{17}x_{23} + x_{17}x_{24} + x_{18}x_{19} + x_{18}x_{21} + x_{18}x_{24} + x_{19}x_{23} + x_{20}x_{22} + x_{20}x_{23} + x_{20}x_{24} + x_{21}x_{23} + x_{21}x_{24} + x_{22}x_{23} + x_{22}x_{24}$$

A cleartext x is then randomly chosen and encoded with the system, giving the ciphertext y which is then used to solve the boolean system as described before:

first, equations E_i are BDD encoded then they are combined to produce functions R_i. At last, vectors that satisfy R_n are produced. Figure 2 is a typical output.

```
GF2(24)
Degree 2304
X : 110101100010111011001110
Y : 111111011001111111100111
E1  : 0s 12283 nodes
E2  : 0s  9179 nodes
E3  : 0s 10235 nodes
E4  : 0s  8185 nodes
E5  : 0s 12283 nodes
E6  : 0s  7161 nodes
E7  : 0s 10229 nodes
E8  : 0s  6901 nodes
E9  : 0s  9205 nodes
E10 : 1s 10235 nodes
E11 : 1s 12279 nodes
E12 : 1s 12285 nodes
E13 : 1s 12281 nodes
E14 : 1s  7675 nodes
E15 : 1s 10235 nodes
E16 : 1s 12285 nodes
E17 : 1s 10235 nodes
E18 : 1s 10233 nodes
E19 : 2s  7285 nodes
E20 : 2s 11773 nodes
E21 : 2s  9209 nodes
E22 : 2s 12283 nodes
E23 : 2s  9213 nodes
E24 : 2s 10231 nodes

R1  :  2s  12283 nodes
R2  :  3s 210735 nodes
R3  :  6s 589695 nodes
R4  :  9s 490549 nodes
R5  : 11s 318192 nodes
R6  : 12s 199306 nodes
R7  : 13s 123517 nodes
R8  : 14s  78076 nodes
R9  : 14s  46515 nodes
R10 : 14s  26830 nodes
R11 : 14s  16579 nodes
R12 : 14s  11242 nodes
R13 : 14s   8080 nodes
R14 : 14s   5524 nodes
R15 : 14s   3591 nodes
R16 : 14s   2307 nodes
R17 : 14s   1359 nodes
R18 : 14s    883 nodes
R19 : 14s    546 nodes
R20 : 14s    332 nodes
R21 : 14s    234 nodes
R22 : 14s    121 nodes
R23 : 14s     67 nodes
R24 : 14s     25 nodes
X    : 110101100010111011001110
Sol 1: 110101100010111011001110 <--
Biggest bdd      :    589695 nodes
Pure CPU time    : 14 seconds
```

FIGURE 2. A typical output

Figure 3 shows that a computation with different parameters can produce more than one cleartext (a satisfying assignment of the corresponding boolean function).

```
GF2(29)
Degree 33
X : 11010110001011101100111001101
Y : 11101011001111110010011011000
E1  :   0s 57339 nodes
E2  :   1s 57339 nodes
...
R27 : 532s 172 nodes
R28 : 532s 126 nodes
R29 : 532s 101 nodes
X    : 11010110001011101100111001101
Sol 1: 00010101101001110110000110011
Sol 2: 10101010101010001011001100011
Sol 3: 11010110001011101100111001101 <--
Sol 4: 11101000000101110000000101101
Biggest bdd      :  11809274 nodes
Pure CPU time    : 532 seconds
```

FIGURE 3. Another output

3.1. Analysis

When we first tried to solve Patarin's challenge, we quickly saw that it would exceed any today computer capability. We then decided to try our method on several different HFE systems.

Figure 4 shows how the size of the BDD of R_i typically evolves. As we combined a few E_i, the size of the result R_i growed exponentially up to a very high

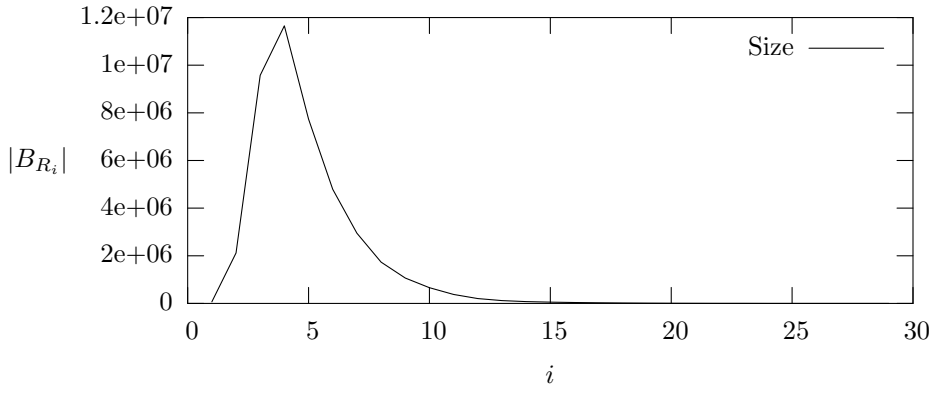


FIGURE 4. Partial results BDD size

peak, then the size decreased down to a very little number. This suggests that not only it is difficult to build a BDD for each E_i, but combining a few is even harder.

Figure 5 shows how this phenomenon is persistent even if we use different fields.

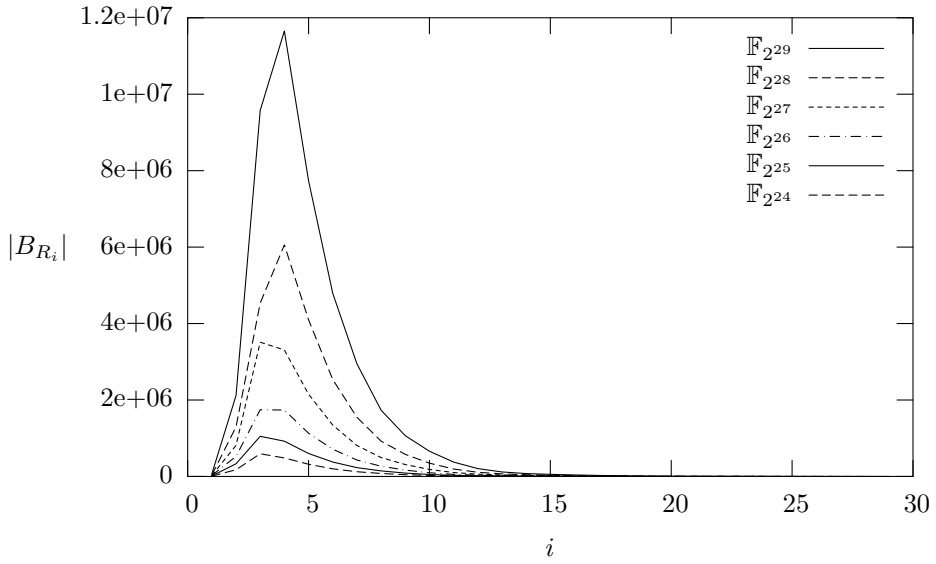


FIGURE 5. Partial results BDD size (different fields)

One should be careful in interpreting these results: we measured the size of BDDs, which is not easily related to the number of solutions satifying the associated boolean formula. What BDD size reveals is in a certain sense the "complexity" of the boolean function; remember also that this complexity is variable ordering dependent. Moreover what our algorithm builds is a set of boolean functions R_i where the sets of solutions $\ldots \subset S_{i+1} \subset S_i \subset \ldots$ are all included one into the other. Thus, even if the number of solutions decreases as i grows, the complexity of the caracteristic function of the satisfying set evolves as the previous figures indicate.

Figure 6 illustrates how the size of the peak evolves as the size of the field grows and how the size of the biggest generated BDD is close to the function $2^x/x$.

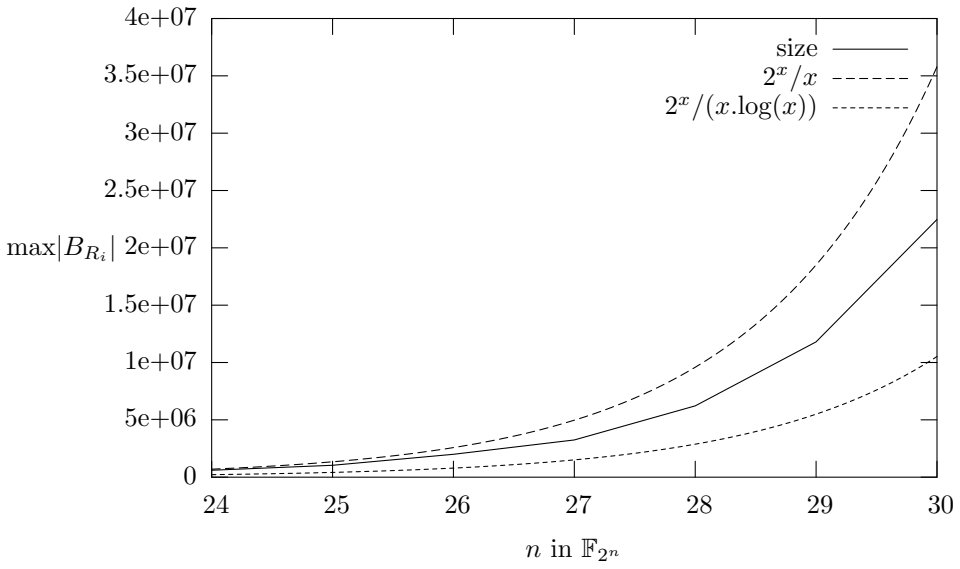


FIGURE 6. Peak size

Some more experiments also showed that the observed complexities are independent to the different variables or equations orderings. More surprisingly there was nothing to note when we modified P, S or T, even when trying a polynomial as simple as $P(X) = X^3$ with $S(X) = T(X) = X$.

Many more experimental results can be found in [13].

4. Conclusion

All those experiments suggest that this method as it is is not suitable to break the HFE cryptosystem. However, it is worth noting that although this method is

not really tractable, is does work and its complexity is smaller than the exhaustive one.

A side effect is that our algorithm can be used to extract roots of any quasi-quadratic polynomial over a finite field. Our experiments showed that a standard computer with 1Gb of memory can break any HFE system defined in $\mathbb{F}_{2^{30}}$ in about 20 minutes. It will be easy to extend it to extract roots of any polynomial over a finite field.

References

[1] R.E. Bryant, *Graph-based algorithms for boolean function manipulation*. In IEEE Transaction on Computers. C-35 volume **8** (1986). 677–691.

[2] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. (1979). W.H. Freeman company.

[3] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. (1999). Cambridge University Press.

[4] A. Kipnis and A. Shamir. *Cryptanalysis of the HFE public key cryptosystem by relinearization*. In LNCS 1966, CRYPTO'99 (1999). Springer-Verlag. 19–30.

[5] M. Krause. *BDD-based cryptanalysis of keystram generators*. In LNCS 2332, EUROCRYPT'2002 (2002). Springer-Verlag. 222–237.

[6] T. Matsumoto and H. Imai. *Public quadratic polynomial-tuples for efficient signature-verification and message encryption*. In LNCS 330, Advances in Cryptology EUROCRYPT'88 (1988). Springer-Verlag. 419–453.

[7] S.I. Minato, N. Ishiura, and S. Yajima. *Shared binary decision diagram with attributed edges for efficient boolean function manipulation*. In 27-th ACM/IEEE Design Automaton Conference (1990). 52–57.

[8] J. Patarin. *Cryptanalysis of the matsumoto and imai public key scheme of eurocrpyt'88*. In LNCS 963, Advances in Cryptology CRYPTO'95 (1995). Springer-Verlag. 248–261.

[9] J. Patarin. *Hidden fields equations (HFE) and isomorphism of polynomials (IP): two new families of asymetric algorithms*. In LNCS 1070, EUROCRYPT'96 (1996). Springer-Verlag. 33–48.

[10] I. Wegener. *BDDs - design, analysis, complexity and applications*. `http://ls2-www.cs.uni-dortmund.de/~wegener/papers/BDDdesign.ps`

[11] *CMU package*. `http://www.cs.cmu.edu/afs/cs/project/modck/pub/www/bdd.html`

[12] *CuDD Package*. `ftp://vlsi.colorado.edu/pub/`

[13] *HFE experiments web page*. `http://www.liafa.jussieu.fr/~yunes/HFE/`

[14] *NTL Library*. `http://www.shoup.net/`

J.F. Michon, LIFAR, Université de Rouen, 76821 Mont Saint-Aignan, FRANCE
E-mail address: Jean-Francis.Michon@univ-rouen.fr

P. Valarcher, LIFAR, Université de Rouen, 76821 Mont Saint-Aignan, FRANCE
E-mail address: Pierre.Valarcher@univ-rouen.fr

J.B. Yunès, LIAFA, Université Paris 7, 75251 Paris FRANCE
E-mail address: Jean-Baptiste.Yunes@liafa.jussieu.fr