Noname manuscript No.
(will be inserted by the editor)

New Subexponential Algorithms for Factoring in $SL(2,\mathbb{F}_{2^n})$

Jean-Charles Faugère[*] · **Ludovic Perret**[*] · **Christophe Petit**[†] ·
Guénaël Renault[*]

the date of receipt and acceptance should be inserted later

Abstract Cayley hash functions are a particular kind of cryptographic hash functions with very appealing properties. Unfortunately, their security is related to a mathematical problem whose hardness is not very well understood, the factorization problem in finite groups. Given a group G, a set of generators $\mathscr{S}$ for this group and an element $g \in G$, the factorization problem asks for a "short" representation of g as a product of the generators. In this paper, we provide a new algorithm for solving this problem for the group $G := SL(2,\mathbb{F}_{2^n})$. We first reduce the problem to the resolution of a particular kind of multivariate equation over $\mathbb{F}_{2^n}$. Then, we introduce a dedicated approach to solve this equation with Gröbner bases. We provide a complexity analysis of our approach that is of independent interest from the point of view of Gröbner basis algorithms. Finally, we give the first subexponential time algorithm computing polynomial-length factorizations of any element g with respect to any generator set $\mathscr{S}$ of $SL(2,\mathbb{F}_{2^n})$. Previous algorithms only worked for specific generator sets, ran in exponential time or produced factorizations that had at least a subexponential length. In practice, our algorithm beats the birthday-bound complexity of previous attacks for medium and large values of n.

1 Introduction

Hash functions are a very important cryptographic primitive, essential for many applications including digital signatures or message authentication codes. Currently used hash functions, like the American standard SHA, follow a heuristic design, somehow similar to block ciphers. Although there exist hash functions based on well-established "hard" number theory problems [32, 19, 17], these functions are too slow in practice compared to "heuristic" functions like the five finalists of NIST' SHA-3 competition[1].

The Tillich-Zémor hash function was proposed at the CRYPTO conference back in 1994 [48]. The function has a strong mathematical structure based on the matrix group $SL(2,\mathbb{F}_{2^n})$. It is rather efficient compared to other "mathematical hash functions". Moreover, the computation of large messages can be efficiently parallelized, a unique property. Although not related to classical assumptions like discrete logarithm or integer factoring, its security relied on the hardness of a mathematical problem, the *factorization problem in finite groups*, in particular in the group $SL(2,\mathbb{F}_{2^n})$. Given a finite group G, a set of generators $\mathscr{S}$ for this group and an element $g \in G$, the factorization problem asks for a "short" representation of g as a product of the generators.

Postdoctoral[†] Research Fellow of the Belgian Fund for Scientific Research (F.R.S.-FNRS) at Université catholique de Louvain (UCL), crypto group.

INRIA[*], Paris-Rocquencourt Center, SALSA Project
UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
CNRS, UMR 7606, LIP6, F-75005, Paris, France
E-mail: jean-charles.faugere@inria.fr, ludovic.perret@lip6.fr,Guenael.Renault@lip6.fr ·
UCL[†] Crypto GroupUniversité catholique de Louvain
Place du levant 3
1348 Louvain-la-Neuve, Belgium
E-mail: christophe.petit@uclouvain.be

[1] `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`

The hardness of this problem is nowadays a widely open problem for essentially all groups G and all generator sets $\mathscr{S}$. The few exceptions include the parameters corresponding to the Tillich-Zémor hash function [35, 44] and the parameters of two other hash functions following the same design [49, 43] (these functions are called *Cayley hash functions*). However, as pointed out in [44], the broken parameters are all very special in some senses and the problem remains open in general. Given the appealing properties of these functions, the factorization problem in finite groups deserves further study.

Another motivation for studying this problem comes from graph theory, in particular from a conjecture of Babai on the diameter of Cayley graphs [1]. The conjecture has recently been proved in many groups of interest [36, 12, 46] but all the proofs that apply to generic parameters are non constructive. The factorization problem in a finite group can in fact be seen as a *constructive* version of Babai's conjecture for this group. Finally, the problem can also be seen as a *routing problem* in the Cayley graphs associated to G and $\mathscr{S}$. Cayley graphs have a lot of applications in computer science, particularly because of their good expander properties [37]. A routing algorithm will certainly be useful for these applications.

For efficiency reasons, the group $SL(2,\mathbb{F}_{2^n})$ is the most appealing one for cryptography. In this paper, we develop and significantly improve the work initiated in [42] for generic generator sets of that group.

1.1 Definitions and Notations

Let $\mathbb{F}_{2^n}$ be the finite field with 2^n elements. We denote by $\mathbb{K}[x_1,x_2,\ldots,x_k]$ the ring of multivariate polynomials in the variables $x_1,\ldots,x_k$ whose coefficients are in a field $\mathbb{K}$. The *special linear group* $G := SL(2,\mathbb{F}_{2^n})$ is the group of 2×2 matrices with determinant 1 and coefficients in $\mathbb{F}_{2^n}$. A *generator set* of G is a set of elements that multiplicatively generate the whole group. We call *Euclidean algorithm matrices* the matrices:

$$\left\{E(t) := \left(\begin{smallmatrix} t & 1 \\ 1 & 0 \end{smallmatrix}\right) | t \in \mathbb{F}_{2^n}\right\}.$$

These matrices naturally appear when applying the Euclidean algorithm to a couple of elements of $\mathbb{F}_2[X]$. Given a matrix $A := \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in SL(2,\mathbb{F}_{2^n})$, we write A' for its transpose matrix $\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$.

Given a set of polynomials $f_1,\ldots,f_\ell \in \mathbb{K}[x_1,x_2,\ldots,x_k]$, the *ideal generated* by $f_1,\ldots,f_\ell$ – denoted $I(f_1,\ldots,f_\ell)$ – is the set of polynomials $\sum_{i=1}^{\ell} g_i f_i$ where $g_i \in \mathbb{K}[x_1,x_2,\ldots,x_k]$. Any element $\prod_{i=1}^{k} x_i^{e_i}$ where $e_i \in \mathbb{N}$ is called a *monomial*. A *monomial ordering* on $\mathbb{K}[x_1,x_2,\ldots,x_k]$ is an ordering $<$ such that $m_1 < m_2 \Rightarrow m_1 m_3 < m_2 m_3$ for any monomials m_1,m_2,m_3 and $m > 1$ for any monomial m. The *lexicographic* ordering with $x_1 > x_2 > \ldots > x_k$ is defined by $\prod_{i=1}^{k} x_i^{e_i} <_{Lex} \prod_{i=1}^{k} x_i^{f_i}$ if the first non-zero term in $e_1-f_1, e_2-f_2,\ldots,e_k-f_k$ is negative. The *reverse lexicographic* ordering of $x_1,\ldots,x_k$ with $x_1 > x_2 > \ldots > x_k$ is defined by $\prod_{i=1}^{k} x_i^{e_i} <_{GrevLex} \prod_{i=1}^{k} x_i^{f_i}$ if $\sum_i e_i < \sum_i f_i$ or if $\sum_i e_i = \sum_i f_i$ and the last non zero term in $e_1-f_1, e_2-f_2,\ldots,e_k-f_k$ is positive. For any $I \subset \{1,\ldots,k\}$, an *elimination ordering* of the variables $\{x_i, i \in I\}$ is an ordering $>$ on $\mathbb{K}[x_1,\ldots,x_k]$ such that for any $e_i, f_i \in \mathbb{Z}^+$, we have $\prod_{i\in I} x_i^{e_i} > \prod_{i\in I} x_i^{f_i} \Rightarrow \prod_{i\in I} x_i^{e_i} \prod_{i\notin I} x_i^{e_i} > \prod_{i\in I} x_i^{f_i} \prod_{i\notin I} x_i^{f_i}$. The *leading term* $\mathrm{LT}(f)$ of a polynomial $f \in \mathbb{K}[x_1,x_2,\ldots,x_k]$ for a given ordering is equal to the term that is the largest one for the ordering. For any polynomial $f \in \mathbb{K}[x_1,x_2,\ldots,x_k]$, we write $\mathrm{Mon}(f)$ for the set of monomials terms of f.

Consider a partition of $n_1 n_2$ variables $\{x_{ij}\}_{1\le j\le n_1}^{1\le i\le n_2}$ into n_2 blocks

$$X_i := \{x_{ij} | j = 1,\ldots,n_1\}.$$

We say that a monomial $m \in \mathbb{F}_2[x_{11},\ldots,x_{n_2 n_1}]$ has *block-degree* at most $(d_1,\ldots,d_{n_2})$ if its degree with respect to the variables of X_i is d_i for any $i, 1 \le i \le n_2$. We say that a polynomial $f \in \mathbb{F}_2[x_{11},\ldots,x_{n_2 n_1}]$ has *block-degree* at most $(d_1,\ldots,d_{n_2})$ if it is a sum of monomials of block-degree at most $(d_1,\ldots,d_{n_2})$. We will also consider the natural extension of this definition to blocks of variables with different sizes.

For any irreducible polynomial p of degree n, the field $\mathbb{F}_{2^n}$ can be seen as the quotient $\mathbb{F}_2[X]/(p(X))$. It is a vector space over $\mathbb{F}_2$. We write $\mathrm{Vec}(w_1,\ldots,w_k)$ for the vector subspace generated by the $w_1,\ldots,w_k \in \mathbb{F}_{2^n}$ When the polynomial p is fixed, any variable x_i over $\mathbb{F}_{2^n}$ can be naturally "deployed" through the substitutions $x_i = \sum_{i=0}^{n-1} x_{i,j} X^j$, where the variables x_{ij} belong to $\mathbb{F}_2$. Similarly, any multivariate polynomial $f \in \mathbb{F}_{2^n}[x_1,x_2,\ldots,x_k]$ can be naturally deployed as n multivariate polynomials $[f]_i^{\downarrow} \in \mathbb{F}_2[x_{1,1},\ldots,x_{k,n}]$, $i = 0,\ldots,n-1$ where

$$f\left(\sum_{i=0}^{n-1} x_{1,j} X^i, \ldots, \sum_{i=0}^{n-1} x_{k,j} X^i\right) = \sum_{i=0}^{n-1} [f]_i^{\downarrow} X^i.$$

We write $[f]^{\downarrow}$ for the vector $([f]^{\downarrow}_1, \ldots, [f]^{\downarrow}_n)$.

The parameter n will be taken as a complexity or security parameter. We write PPT for probabilistic polynomial time in n. We write O for the "big O" notation: given two functions f and g of n, we say that $f = O(g)$ if there exist $N, c \in \mathbb{Z}^+$ such that $n > N \Rightarrow f(n) \leq cg(n)$. We write o for the "small O" notation: given two functions f and g of n, we say that $f = o(g)$ if for any $\varepsilon \in \mathbb{R}$, there exists $N \in \mathbb{Z}^+$ such that $|f(n)| < \varepsilon |g(n)|$ for all $n > N$. We write $\binom{n}{k}$ for the binomial value $\frac{n!}{k!(n-k)!}$. We write ω for the *linear algebra constant*. Depending on the algorithm used for linear algebra, we have $2.376 \leq \omega \leq 3$.

1.2 The Factorization Problem in Finite Groups

The factorization problem in $SL(2, \mathbb{F}_{2^n})$ can be defined as follows.

Problem 1 *Let $\mathscr{S} = \{A_0, \ldots, A_{k-1}\} \in SL(2, \mathbb{F}_{2^n})$ be a set of generators of $SL(2, \mathbb{F}_{2^n})$ and let $L \in \mathbb{Z}$. Find an algorithm that, given any element $g \in SL(2, \mathbb{F}_{2^n})$, returns a word $m_1 \ldots m_L$ with $m_i \in \{0, \ldots, k-1\}$ such that*

$$\prod_{i=1}^{L} A_{m_i} = g.$$

Without loss of generality, we can assume that $\mathscr{S} = \{A_0, A_1\}$ for two matrices A_0 and A_1. The hardness of this problem clearly depends on n and L. As the parameter n increases, we would like an algorithm running in time $T \leq p_1(n)$ and returning factorizations of length $L \leq p_2(n)$ where p_1, p_2 are polynomials. When the factors A_0^{-1} and A_1^{-1} are allowed as well, polynomial length factorizations exist due to a proof of Babai's conjecture that applies to $SL(2, \mathbb{F}_{2^n})$ [1, 46, 12]. However, the proof is not constructive and no polynomial-time algorithm is known to return these factorizations.

Polynomial time algorithms returning polynomial length factorizations are only known for specific generator sets [44, 2]. For generic generator sets, the algorithm of [45] produces polynomial-length factorizations but only in exponential time, essentially equal to $2^{n/2}$. On the other hand, the algorithm in [42] runs in subexponential time but it only produces subexponential length (not polynomial length) factorizations. This last result exploits self-reductions of the problem from generic generator sets $\mathscr{S}$ and generic elements g to a class of sets $\tilde{\mathscr{S}}$ and elements $\tilde{g}$ with some special structures.

In this paper, we provide a subexponential time algorithm producing polynomial length factorizations for any $\mathscr{S}$ and any g. To obtain this result, we first extend the work of [42] and we reduce the factorization problem to solving a multivariate polynomial equation over $\mathbb{F}_{2^n}$ with additional constraints on the solutions. We call this equation the *trapdoor equation*. After deploying it over $\mathbb{F}_2$, we obtain a system of polynomial equations that can be naturally tackled with *Gröbner basis techniques*.

1.3 Gröbner Basis Algorithms

In the following, we suppose that an ordering on the monomials has been fixed. A *Gröbner basis* [14, 13, 15, 16, 18] of an ideal $I(f_1, \ldots f_\ell)$ is a basis $\{f'_1, \ldots, f'_{\ell'}\}$ of this ideal such that

$$\forall f \in I(f_1, \ldots f_\ell), \exists i \in \{1, \ldots, \ell'\} \text{ such that } \mathrm{LT}(f'_i) | \mathrm{LT}(f).$$

The usual strategy to solve a polynomial system is to compute a "triangular" basis for the ideal generated by the equations, that is a set of polynomials $\{f'_1, \ldots, f'_{\ell'}\}$ that generate the same ideal, such that f'_i only depends on the variables $\{x_j \mid j \in I_i\}$ where $I_{\ell'} \subset I_{\ell'-1} \subset \cdots \subset I_1 = \{1, \ldots, k\}$. Such a form allows efficiently computing the zeros of the ideal. In fact, Gröbner bases for the lexicographic ordering have this triangular form [13, 15, 18].

Since the notion of Gröbner bases is defined by the existence of *relatively* small leading terms, the task of computing a Gröbner basis is essentially that of finding new elements in the ideal with smaller leading terms until no more such elements can be found. Buchberger proved in his PhD thesis [13, 15] that Gröbner bases can be computed by considering only specific polynomials, the so-called S-polynomials [13, 15, 18]. These polynomials are designed to cancel leading terms and they thus potentially produce new elements in the ideal with lower leading terms. In many cases, the degree of the polynomials (named critical pairs) used for producing S-polynomials may increase during the computation, even if the resulting basis only contains polynomials with small degrees. This is the reason why it is sometimes preferable to consider *truncated* d-Gröbner basis. A subset $\{f'_1,\dots,f'_{\ell'}\}$ is said to be a *truncated* d-Gröbner basis of the ideal $I(f_1,\dots f_\ell)$ if it is obtained from a Gröbner basis computation by ignoring critical pairs whose degree is greater than d (see [21]). In general, a truncated d-Gröbner basis is not a basis of the given ideal I. However in the case of a homogeneous ideal, i.e. when the polynomials $f_1,\dots f_\ell$ are homogeneous, this set corresponds to the polynomials up to degree d in the Gröbner basis of I. In particular, it verifies

$$\forall f \in I(f_1,\dots f_\ell)\text{such that}\deg f \leq d, \exists i \in \{1,\dots,\ell'\} \text{ such that } \mathrm{LT}(f'_i)|\mathrm{LT}(f).$$

More recently, improved algorithms have been proposed by Faugère: the F4 and F5 algorithms [21, 22]. These algorithms are considered today as the most efficient ones to compute Gröbner bases. In practice, a lexicographic Gröbner basis is never computed directly. Instead, a Gröbner basis is first computed for the reverse lexicographic ordering and then converted into a lexicographic Gröbner basis with the FGLM algorithm [20].

The complexity of a Gröbner basis computation is

$$O\left(\binom{n+d}{d}^{\omega}\right), \tag{1}$$

where ω is the linear algebra constant and d is the regularity degree of the ideal [4, 5, 6, 8]. Roughly speaking, the *regularity degree* is the maximum degree reached during the computation of a Gröbner basis. In special cases, this quantity can be estimated. For a *generic* (or random) ideal $I(f_1,\dots,f_\ell)$, the regularity degree is

$$d = 1 + \sum_{i=1}^{\ell}(\deg(f_i) - 1), \text{ if } \ell \leq n. \tag{2}$$

Further estimations are known for overdefined sytems i.e. when $\ell \geq n$. However, none of these estimations holds for ideals with special structures – which is typically the case of systems coming from real-life applications. In practice, it is always challenging to *a priori* estimate the degree of regularity for a given system.

In the last two decades, Gröbner basis algorithms have succesfully attacked various cryptographic challenges. They were used in the cryptanalysis of HFE [25, 34] and multi-HFE [9], the Isomorphism of Polynomials [29, 11], McEliece variants [28], the Hidden Matrix cryptosystem [27], algebraic side-channel attacks [33], elliptic curve discrete logarithm [31],... In all these applications, the occuring algebraic systems were not generic and could be solved much more efficiently than generic systems, i.e. the regularity degree was much smaller than what is expected for a generic system of the same size.

The *trapdoor equation* occuring in this paper somehow resembles HFE equations [41, 25, 34]. In both cases, a polynomial equation over $\mathbb{F}_{2^n}$ is deployed as a system of polynomial equations over $\mathbb{F}_2$. In our context however, the equation is multivariate and not linear as in HFE. Besides, the solutions of our trapdoor equation are constrained to belong to some vector subspace of $\mathbb{F}_2^n/\mathbb{F}_2$. Our trapdoor equation also has the particular property to be *affine* in each variable over $\mathbb{F}_{2^n}$. This induces a so-called multi-linear structure. We mention, that bilinear and bi-affine equations were extensively studied in [30] and used against McEliece variants and MinRank [28, 24]. However, the more general multi-linear structure has not been thoroughly investigated so far.

1.4 Contributions of this Paper

The contributions of this paper are three-fold. First, we reduce the factorization problem in $SL(2,\mathbb{F}_{2^n})$ for *arbitrary* generator sets to the resolution of some multivariate polynomial equation over $\mathbb{F}_{2^n}$ with additional constraints on the solution. We call this equation the *trapdoor equation*. To do so, we develop the reductions of [42] to first replace any generic generator set $\mathscr{S}$ by a particular set $\tilde{\mathscr{S}}$ containing *Euclidean algorithm matrices*, i.e.

$$\tilde{\mathscr{S}} := \left\{ \left(\begin{smallmatrix} t+t_i & 1 \\ 1 & 0 \end{smallmatrix}\right) | t_i \in V_{n_1} \right\}$$

where V_{n_1} is some vector subspace of $\mathbb{F}_2^n/\mathbb{F}_2$. We then apply – a slight modification of – Proposition 11 of [42] to obtain a multivariate polynomial equation over $\mathbb{F}_{2^n}$ with additional (linear) constraints on the solutions, namely

$$f(t_1,\dots,t_{n_2}) := \left(\begin{smallmatrix} 1 & 1 \end{smallmatrix}\right) \left[\prod_{i=1}^{n_2} \left(\begin{smallmatrix} t+t_i & 1 \\ 1 & 0 \end{smallmatrix}\right)\right] \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right) = 0, \qquad t_i \in V_{n_1}.$$

Our second contribution is a specific technique to solve such equations for which we can provide an exact complexity analysis. For appropriately balanced parameters n_1, n_2, our algorithm has a complexity:

$$O\left(2^{\omega \frac{n_2 \log n \log n_1}{\log(n/n_1)}}\right),$$

where ω is the linear algebra constant. The idea of the algorithm is to deploy the equation $f = 0$ over $\mathbb{F}_2$. We then add to this system many others equations of the form $m_j f = 0$, with monomials m_j of the type:

$$m_j := \prod_{i=1}^{n_2} t_i^{e_{ij}}.$$

We then prove that we can linearize the problem over $\mathbb{F}_2$ by adding enough equations and choosing the monomials m_j appropriately. With such suitable choices, the number of equations exceeds the number of monomials involved in the equations. The linear independence of the equations is confirmed by experimental results. In fact, our algorithm can be seen as a specialization of generic Gröbner basis algorithms exploiting the particular structure of the problem. Indeed, we show that all the equations that we generate from the equation $m_j f = 0$ are in fact algebraic combinations of the original equations obtained from the equation $f = 0$. Generic Gröbner basis algorithms will therefore recover the same relations, but only after "blindly" generating a lot of high degree relations. By choosing the monomials m_j appropriately, our algorithm takes advantage of two specificities of the problem, namely the block structure and the existence of low degree algebraic relations related to the Frobenius transform. For medium-size parameters, we present some heuristic ideas inspired by the hybrid method [10] that seem to improve the complexity even further.

Using our Gröbner basis algorithm to solve the trapdoor equation, we obtain a new algorithm for the factorization problem in $SL(2,\mathbb{F}_{2^n})$, the third contribution of this paper. Our algorithm is not yet a dramatic threat for current practical parameters of Cayley hash functions. However, it warns that the hardness of the problem does not scale as well as previously expected when the parameter sizes increase. In particular for appropriately chosen parameters, our algorithm can produce polynomial length factorizations in subexponential time for any generator set $\mathscr{S}$ and any matrix g. Moreover, balancing the parameters in different ways provides various interesting trade-offs between time and factorization lengths. In contrast, all previous algorithms either produced subexponential length factorizations [42], ran in exponential time [45, 42] or were limited to specific generator sets [3, 44].

1.5 Roadmap

The remaining of this paper is organized as follows. In Section 2, we reduce the factorization problem to what we call the trapdoor equation. We study and solve the trapdoor equation in Sections 3 and 4. In Section 5, we give a new algorithm for factoring in $SL(2,\mathbb{F}_{2^n})$ and we conclude the paper in Section 6.

2 Reduction to a Multilinear Equation over $\mathbb{F}_{2^n}$

2.1 Euclidean Algorithm Matrices

To avoid considering a distinct factorization problem for every generator set $\mathscr{S}$, it is useful to reduce some instances of the problem to other ones from a limited set. In this section, we show that it is sufficient to consider generator sets with a particularly simple special structure.

For any $w \in \mathbb{F}_{2^n}$, let $O(w) := \left(\begin{smallmatrix} w+1 & w \\ w & w+1 \end{smallmatrix}\right)$. The matrices $\{O(w) | w \in \mathbb{F}_{2^n}\}$ form the orthogonal subgroup of $SL(2,\mathbb{F}_{2^n})$. It was shown in [42] that any factorization problem in $SL(2,\mathbb{F}_{2^n})$ can be reduced to another factorization problem in $SL(2,\mathbb{F}_{2^n})$ for a generator set containing one symmetric matrix and a subgroup of the orthogonal subgroup of $SL(2,\mathbb{F}_{2^n})$.

Proposition 2 [42] *Let $\mathscr{S} := \{A, B\}$ generating $SL(2,\mathbb{F}_{2^n})$. Let $n_1 < n$. The factorization problem for $\mathscr{S}$ is reducible to another factorization problem in $SL(2,\mathbb{F}_{2^n})$ for $\tilde{\mathscr{S}} = \{M\} \cup \mathscr{O}_{n_1}$ where M is a symmetric matrix and $\mathscr{O}_{n_1} = \{O(w_i) | i = 1, ..., n_1\}$ for some $w_1, ..., w_{n_1} \in \mathbb{F}_{2^n}$.*
Given an algorithm that returns factorizations for $\tilde{\mathscr{S}}$ containing at most L matrices M, the factorizations for $\mathscr{S}$ returned by the reduction algorithm have a length bounded by $4(n_1+1)3^{n_1-1}L$.

We deduce the following.

Proposition 3 *Let $\mathscr{S} := \{A,B\}$ generating $SL(2,\mathbb{F}_{2^n})$. The factorization problem in $SL(2,\mathbb{F}_{2^n})$ for $\mathscr{S}$ is reducible to a factorization problem for*

$$\tilde{\mathscr{S}} := \{E(t+t_i) | t_i \in V_{n_1}\} \tag{3}$$

where V_{n_1} is some vector subspace of $\mathbb{F}_{2^n}/\mathbb{F}_2$ and $t \in \mathbb{F}_{2^n}$. The reduction increases the bound on the factorization lengths by a factor at most $4(n_1+1)3^{n_1-1}$.

PROOF: The proof mainly follows the proof of Proposition 12 in [42]. By Proposition 2, we can start from a set $\mathscr{S}_2 := \{M\} \cup \{O(w_i) | 1 \le i \le n_1\}$ where M is a symmetric matrix. Let t be the trace of M. Lemma 6 in [42] implies the existence of $\tilde{w} \in \mathbb{F}_{2^n}$ such that

$$O(\tilde{w})MO(\tilde{w}) = \left(\begin{smallmatrix} t & 1 \\ 1 & 0 \end{smallmatrix}\right).$$

Let

$$S := \left(\begin{smallmatrix} \tilde{w}+1 & \tilde{w} \\ \tilde{w} & \tilde{w}+1 \end{smallmatrix}\right)\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1+t \end{smallmatrix}\right).$$

We have

$$S^{-1}MS = \left(\begin{smallmatrix} t & 1 \\ 1 & 0 \end{smallmatrix}\right)$$

Moreover, for any $w \in \mathbb{F}_{2^n}$, we have

$$\begin{aligned} S^{-1}O(w)S &= \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1+t \end{smallmatrix}\right)^{-1} O(\tilde{w})O(w)O(\tilde{w}) \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1+t \end{smallmatrix}\right) \\ &= \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1+t \end{smallmatrix}\right)^{-1} O(w) \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1+t \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & wt \\ 0 & 1 \end{smallmatrix}\right) \end{aligned}$$

and

$$S^{-1}O(w)MS = S^{-1}O(w)S \cdot S^{-1}MS = \left(\begin{smallmatrix} 1 & wt \\ 0 & 1 \end{smallmatrix}\right)\left(\begin{smallmatrix} t & 1 \\ 1 & 0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} t+tw & 1 \\ 1 & 0 \end{smallmatrix}\right).$$

Let $g \in SL(2,\mathbb{F}_{2^n})$ and suppose there exists a factorization algorithm for $\tilde{\mathscr{S}} := \{E(t+t_j) | t_j \in V_{n_1}\}$ where $V_{n_1} := \mathrm{Vec}(tw_1, ..., tw_{n_1})$. Applying this algorithm to $S^{-1}gS$, we deduce a factorization of g as a product of the elements of $\mathscr{S}_3 = \{O(w)M, w \in F\}$ hence of $\mathscr{S}_2$. The bound on the factorization lengths follows by Proposition 2. $\square$

We point out the similarity of this generator set with the generators of the Tillich-Zémor hash function, broken in [35, 44]:

$$\mathscr{S}_{TZ} := \{\left(\begin{smallmatrix} X & 1 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} X+1 & 1 \\ 1 & 0 \end{smallmatrix}\right)\}.$$

However, in order to generalize the attacks against the Tillich-Zémor hash function to $\tilde{\mathscr{S}}$, we would first need to generalize Mesirov and Sweet's algorithm [40] to arbitrary partial quotients. The results of Lauder [38] imply that this is only possible for very few sets of partial quotients. We therefore use a different approach to tackle generic parameters.

2.2 Trapdoor Matrices for the Factorization Problem

Trapdoor matrices are matrices such that factoring any single one of them would allow factoring any element in the group [42]. Proposition 11 of [42] provides an interesting class of trapdoor matrices. We slightly modify it as follows.

Proposition 4 *Let $\mathscr{S}$ be a generator set of $SL(2,\mathbb{F}_{2^n})$ containing only symmetric matrices. Suppose that we know the factorization of a matrix T that has $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ as eigenvector and suppose that the corresponding eigenvalue has algebraic degree n. Then we can factor any matrix as a product of the elements of $\mathscr{S}$.*

PROOF: The proof closely follows the proof of Proposition 11 of [42] so we only sketch it here and we refer the reader to [42] for the details. Since the generators are symmetric, we obtain a factorization of the transpose of T by reading the factorization of T in reverse order. Since T has eigenvector $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$, the matrix $T^kT'^k$ is orthogonal for any k. Moreover, the whole orthogonal subgroup can be generated as a product of at most n matrices from the set $\{T^kT'^k | k = 1, ..., n\}$. Finally, any matrix of $SL(2,\mathbb{F}_{2^n})$ can be decomposed as a product of 4 orthogonal matrices and 3 arbitrary matrices. $\square$

Note that the proof is quite efficient: if the factorization of T has length L, then we obtain factorizations of any element with lengths bounded by $8n^2L+3$. We remark that the condition "$T = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ has eigenvector $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ " is equivalent to $a+b=c+d$ or $a+b+c+d=0$. Propositions 3 and 4 therefore lead to the following *trapdoor equation*:

$$f(t_1,\dots,t_{n_2}) := \left(\begin{smallmatrix} 1 & 1 \end{smallmatrix}\right) \left[\prod_{i=1}^{n_2} \left(\begin{smallmatrix} t+t_i & 1 \\ 1 & 0 \end{smallmatrix}\right)\right] \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right) = 0, \qquad t_i \in V_{n_1} \tag{4}$$

where V_{n_1} is a vector subspace of $\mathbb{F}_2^n/\mathbb{F}_2$ with dimension n_1. This is in fact a multivariate polynomial equation with additional linear constraints on the solutions.

3 Solving the Trapdoor Equation: First Observations

We present in this part a dedicated technique for solving Equation (4). We first remark that (4) has total degree n_2 and degree 1 in each variable t_i. It also has a symmetry property $f(t_1,\dots,t_{n_2}) = f(t_{n_2},\dots,t_1)$.

Let $\{\theta_1,\dots,\theta_n\}$ be a basis of $\mathbb{F}_{2^n}$ as a vector space over $\mathbb{F}_2$ and let $\{v_1,\dots,v_{n_1}\}$ be a basis of V_{n_1}. We introduce binary variables $t_{ij}, i=1,\dots,n_2, j=1,\dots,n_1$ such that $t_i = \sum_{j=1}^{n} t_{ij}v_j$. We consider Equation (4) both over $\mathbb{F}_{2^n}$ and as a system of polynomial equations in the variables t_{ij}. Indeed, we can write

$$0 = f\left(\sum_{j=1}^{n_1} t_{1j}v_j,\dots,\sum_{j=1}^{n_1} t_{n_2 j}v_j\right) = \sum_{k=0}^{n-1} [f]_k^{\downarrow}\,\theta_k \tag{5}$$

where $[f]_k^{\downarrow} \in \mathbb{F}_2[t_{11},\dots,t_{n_2 n_1}]$. This implies

$$[f]_0^{\downarrow} = 0, [f]_1^{\downarrow} = 0, \dots, [f]_{n-1}^{\downarrow} = 0. \tag{6}$$

Each of these equations has total degree at most n_2 and degree at most 1 in each block of variables $\{t_{i1},\dots,t_{in_1}\}$.

In our analysis, we will treat V_{n_1} as a randomly chosen vector subspace of dimension n_1, an assumption that was also taken in [42]. Assuming that the distribution of $f(t_{11},\dots,t_{n_2 n_1})$ is close to uniform for random assignments of the variables t_{ij}, we approximate the number of solutions of Equation (4) by $2^{n_1 n_2 - n}$. We will therefore usually assume $n_1 n_2 \geq n$.

According to Proposition 3, the parameter n_1 should be kept as small as possible to decrease the factorization lengths. On the other hand, the degree of regularity *a priori* increases with the degree of the equations. As a consequence, we would also like to keep n_2 as small as possible.

3.1 Regularity Degree: First Experimental Results

We first study the influence of n_1, n_2 and n on the degree of regularity of System (6). In our experiments, we chose $n \in \{10,15,20,25,30\}$ and we progressively decreased n_1 starting from $n_1 := n$. For each value n_1, we picked a random V_{n_1} and we chose $n_2 = \lceil \frac{n}{n_1} \rceil$. When n was larger than $n_1 n_2$, we randomly fixed the last binary variables of t_{n_2}. We tried to solve System (6) using Magma's implementation of the F4 algorithm [21]. When the computation exceeded a few hours, we stopped it and we changed the current value of n for the next one. When the computation was successful and the number of solutions was minimal (1 in general but 2 sometimes due to the symmetry), we recorded the degree of regularity. Three data points were collected for each parameter set.

The results are shown in Figure 1. Despite some variability within identical experiences, two observations can be made. First, the degree of regularity seems to increase more or less linearly with n_2 when n is fixed. The proportionality constant seems to be slightly larger than 1. Second, this constant seems to increase slowly with n. System (6) therefore seems much easier to solve than generic systems (a.k.a random system). Indeed, its degree of regularity would be equal to $1+n(n_2-1)$ if it was generic [5, 6, 8, 4]. This experimentally shows that Gröbner basis computations on System (6) terminate at a particularly low degree compared to generic systems. Before explaining this phenomenon, we try to accelerate the Gröbner basis computation by overdetermining the system using new low-degree equations.

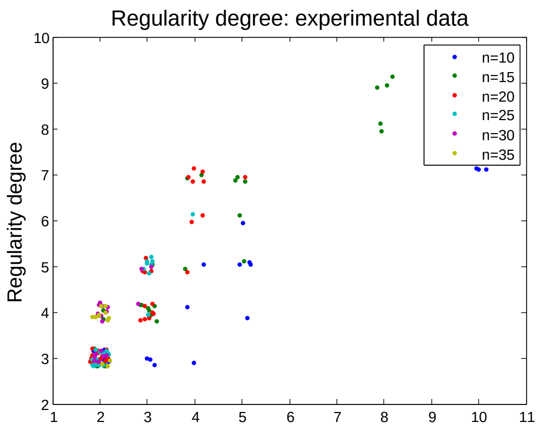

Fig. 1 Experimental degree of regularity (each point is randomly perturbed to make the figure more readable).

3.2 Useless Equations: Frobenius Transforms

Adding new equations is a common way to decrease the cost of a Gröbner basis computation. However, the new equations only help if they are linearly independent of the initial ones. For instance, we consider the Frobenius images of Equation (4). Interestingly, these images produce new equations over $\mathbb{F}_2$ of degree n_2. However, we have

$$0 = f^2 = \left([f]_1^{\downarrow}\theta_1 + \cdots + [f]_n^{\downarrow}\theta_n\right)^2 = \left([f]_1^{\downarrow}\right)^2\theta_1^2 + \cdots + \left([f]_n^{\downarrow}\right)^2\theta_n^2$$
$$= [f]_1^{\downarrow}\theta_1^2 + \cdots + [f]_n^{\downarrow}\theta_n^2$$

since for any i, $[f]_i^{\downarrow}$ is a polynomial over $\mathbb{F}_2$. The set $\{\theta_1^2, \ldots, \theta_n^2\}$ is another basis of $\mathbb{F}_{2^n}$ as a vector space over $\mathbb{F}_2$. Thus, we can write $\theta_i^2 = \sum_{j=1}^n a_{ij}\theta_j$ for some $a_{ij} \in \mathbb{F}_2$. We finally obtain

$$0 = f^2 = \left(\sum_{i=1}^n a_{i1}[f]_i^{\downarrow}\right)\theta_1 + \cdots + \left(\sum_{i=1}^n a_{in}[f]_i^{\downarrow}\right)\theta_n.$$

Hence

$$\sum_{i=1}^n a_{ij}[f]_i^{\downarrow} = 0, \qquad \forall j, 1 \le j \le n.$$

We see that the new equations obtained this way are linear combinations – over $\mathbb{F}_2$ – of the equations occuring in System (6). For this reason, they fail in accelerating the Gröbner basis computation. We now describe a more successful method.

3.3 Adding New Equations

To illustrate the main idea behind our linearization algorithm, let us consider the equation $t_1 f(t_1, \ldots, t_{n_2}) = 0$ over $\mathbb{F}_{2^n}$. Since f is affine in each variable, we can write

$$f(t_1, \ldots, t_{n_2}) = t_1 f_1(t_2, \ldots, t_{n_2}) + f_2(t_2, \ldots, t_{n_2})$$

for some $f_1, f_2 \in \mathbb{F}_{2^n}[t_2, \ldots, t_{n_2}]$. It follows that

$$t_1 f(t_1, \ldots, t_{n_2}) = t_1^2 f_1(t_2, \ldots, t_{n_2}) + t_1 f_2(t_2, \ldots, t_{n_2}). \tag{7}$$

As $t_1^2 = \sum_{j=1}^{n_1} t_{1j} v_j^2$, Equation (7) is deployed over $\mathbb{F}_2$ as a set of n multivariate polynomials with block-degrees $(1, \ldots, 1)$. Interestingly, the new equations are *a priori* linearly independent of the equations of System (6). Indeed, every monomial in every new equation is divisible by exactly one variable among $\{t_{11}, \ldots, t_{1n_1}\}$, whereas *a priori* many

Table 1 Adding $n2^{n_2}$ equations: time and memory improvements.

	n equations		$n2^{n_2}$ equations		Gain	
	time (s)	mem. (MB)	time (s)	mem. (MB)	time	mem.
$n=20, n_1=6$	350	3200	5	72	70	44
$n=20, n_1=5$	1200	6400	25	212	48	30
$n=20, n_1=4$	86500	206000	187	2540	462	81

terms in the equations of System (6) have degree 0 in all of these variables. More precisely, there are $(n_1+1)^{n_2-1}$ monomials with degree 0 in $\{t_{11},\dots,t_{1n_1}\}$ that can appear in the equations of System (6). These terms can not all be canceled via linear combinations of the n equations.

Similarly, we obtain $n2^{n_2}$ new equations with degrees 1 in each block of variables by deploying one equation $mf=0$ for each $m \in \mathbb{F}_{2^n}[t_1,\dots,t_{n_2}]$ which is a monomial of degree at most 1 in every variable. With the help of these new equations, we obtain substantial practical improvements in time and memory as can be seen in Table 1. We can obtain even more equations if we multiply f by monomials of higher degree, but this time at the cost of rising the block-degrees in the deployed equations.

For example, the equations obtained by deploying

$$t_1^2 f(t_1,\dots,t_{n_2}) = t_1^3 f_1(t_2,\dots,t_{n_2}) + t_1^2 f_2(t_2,\dots,t_{n_2})$$

have block-degrees at most $(2,1,\dots,1)$ since t_1^3 is deployed as a quadratic polynomial over $\mathbb{F}_2$. Similarly, the equations obtained by deploying $t_1^{2^k} f$ where $k \in \{1,\dots,n\}$ also have block-degrees at most $(2,1,\dots,1)$ and the equations obtained by deploying $\left(\prod_{i=1}^{n_2} t_i^{2^{k_i}}\right) f$ where $k_i \in \{1,\dots,n\}$ have block degree at most $(2,\dots,2)$. More generally, we consider the equations obtained by deploying

$$\left(\prod_{i=1}^{n_2} t_i^{e_i}\right) f. \tag{8}$$

Thus, if

$$\max_{1\le i\le n_2} \max\left(\mathrm{HW}(e_i), \mathrm{HW}(e_i+1)\right) \le d,$$

where $\mathrm{HW}(e)$ is the Hamming weight of the binary representation of e, then the equations obtained by deploying (8) have block-degrees at most $(d,\dots,d)$. Interestingly – and in a first approximation – the number of equations obtained evolves with d as

$$n\left(2n^{d-1}\right)^{n_2}$$

whereas the number of monomials involved in the equations evolves roughly as

$$\left((n_1+1)^d\right)^{n_2}.$$

We will give exact estimates in Section 4. As d increases, the number of equations goes beyond the number of monomials involved in these equations. Unless unexpected linear dependencies exist between the equations, Equation (4) can then be solved by a simple linearization strategy.

3.4 Another Look at the New Equations

Let us once again consider the monomial $m = \prod_{i=1}^{n_2} t_i^{e_i} \in \mathbb{F}_{2^n}[t_1,\dots,t_{n_2}]$. This monomial is deployed over $\mathbb{F}_2$ as follows:

$$m = \sum_{i=1}^{n} p_i(t_{11},\dots,t_{n_2 n_1})\theta_i$$

for some polynomials $p_i \in \mathbb{F}_2[t_{11},\dots,t_{n_2 n_1}]$ having a block-degree at most $\left(\mathrm{HW}(e_1),\dots,\mathrm{HW}(e_{n_2})\right)$.

Let $a_{ij}^{(k)} \in \mathbb{F}_2$ be such that $\theta_i\theta_j = \sum_{k=1}^{n} a_{ij}^{(k)}\theta_k$. By multiplying Equation (5) by m we get

$$0 = mf(t_1,\dots,t_{n_2}) = \sum_{i,j=1}^{n} p_i(t_{11},\dots,t_{n_2n_1})\,[f]_j^{\downarrow}\,\theta_i\theta_j = \sum_{i,j,k=1}^{n} a_{ij}^{(k)}\cdot p_i(t_{11},\dots,t_{n_2n_1})\,[f]_j^{\downarrow}\,\theta_k.$$

Hence

$$\sum_{j=1}^{n} p_{jk}(t_{11},\dots,t_{n_2n_1})\,[f]_j^{\downarrow} = 0, \qquad \forall k, 1 \le k \le n. \tag{9}$$

where the polynomials $p_{jk} := \sum_{i=1}^{n} a_{ij}^{(k)} p_i$ have block-degrees at most $\big(\mathrm{HW}(e_1),\dots,\mathrm{HW}(e_n)\big)$. We deduce that each new equation is an algebraic combination of Equations (6). The new equations can therefore be recovered by any standard Gröbner basis algorithm. However, even though the new equations have total degree at most

$$\sum_{i=1}^{n_2} \max(\mathrm{HW}(e_i),\mathrm{HW}(e_i+1)),$$

they would only be recovered by generic Gröbner basis algorithm at the degree

$$n_2 + \sum_{i=1}^{n_2} \mathrm{HW}(e_i).$$

since the polynomials p_{jk} have block-degrees at most $\big(\mathrm{HW}(e_1),\dots,\mathrm{HW}(e_n)\big)$. For appropriately chosen exponents, the degree drop in the new equations can be as large as n_2 (a degree drop of 1 per block of variables).

3.5 Analyzing Linear Dependencies

A linearization strategy only succeeds if most equations involved in the system are linearly independent. In this section, we identify two causes of linear dependencies. However, we also argue that they can be easily prevented with an appropriate choice of the monomials used to generate the new equations.

The first kind of linear dependencies are due to what we call a *saturation effect*. To understand this effect, let us suppose that we increase the degree of the equations with respect to the first block only. Let

$$\mathscr{M}_d := \left\{ \prod_{i=1}^{n_2} t_i^{e_i} \,|\, \max\big(\mathrm{HW}(e_1),\mathrm{HW}(e_1+1)\big) \le d \right\}, \text{ for some fixed } e_2,\dots,e_{n_2} \in \mathbb{N}.$$

Let $g := \big(\prod_{i=2}^{n_2} t_i^{e_i}\big) f$ and let $[g]_j^{\downarrow}, j = 1,\dots,n$ be the n equations obtained by deploying g over $\mathbb{F}_2$. Following the analysis of the previous section, each of the $n|\mathscr{M}_d|$ equations obtained by deploying $m \cdot f$ for each $m \in \mathscr{M}_d$ can be written as

$$\sum_{j=1}^{n} p_{m,j}(t_{11},\dots,t_{1n_1})\,[g]_j^{\downarrow} = 0$$

for some polynomials $p_{m,j}$ involving only variables of the first block and that have degree at most d.

Let $\mu(d)$ be the number of monomials of degree d in the variables $\{t_{11},\dots,t_{1n_1}\}$. Clearly the polynomials $p_{m,j}$ are linear combinations of these monomials. Hence the number of *linearly independent* equations that can be generated from $\mathscr{M}_d$ is bounded by $n \cdot \mu(d)$, even if many more equations could at first sight be obtained from different monomials. We call this phenomenon the *saturation effect*. The effect independently appears in each block of variables. As soon as we increase the degree of the equations with respect to one block of variables, the ratio between the number of linearly independent equations and the number of monomials involved in these equations may first increase by an exponential factor (roughly $(n/n_1)^d$) but it will then necessarily *saturate* to some constant value. When the first block of variables is saturated, the ratio can still grow up to 1 by adding new equations with higher degrees with respect to the other blocks. Note that if we simultaneously increase the degree in each block, saturation in the individual blocks will not occur before the total number of equations exceeds the number of monomials.

A second type of linear dependencies is related to the Frobenius transforms investigated in Section 3.2. As an example, let us consider the equations obtained by deploying $\left(\prod_{i=1}^{n_2} t_i\right) f = 0$ over $\mathbb{F}_2$. We have

$$\left(\prod_{i=1}^{n_2} t_i\right) f = f^2 + \sum_{m \in \mathrm{Mon}(f) \setminus \left\{\prod_{i=1}^{n_2} t_i\right\}} mf.$$

Hence

$$\left[\left(\prod_{i=1}^{n_2} t_i\right) f\right]^{\downarrow} = \left[f^2\right]^{\downarrow} + \sum_{m \in \mathrm{Mon}(f) \setminus \left\{\prod_{i=1}^{n_2} t_i\right\}} [mf]^{\downarrow}.$$

Since the equations obtained by deploying $f^2 = 0$ or $f = 0$ are equivalent, it is useless to deploy both the equation $t_1 \cdots t_{n_2} f = 0$ and all the equations $mf = 0$ for $m \in \mathrm{Mon}(f) \setminus \left\{\prod_{i=1}^{n_2} t_i\right\}$.

More generally if $e_1, \ldots, e_{n_2} \in \mathbb{Z}$ are all odd, we have

$$\left(\prod_{i=1}^{n_2} t_i^{e_i}\right) f = \left[\left(\prod_{i=1}^{n_2} t_i^{\lfloor e_i/2 \rfloor}\right) f\right]^2 + \sum_{m \in \mathrm{Mon}\left(\left(\prod_{i=1}^{n_2} t_i^{e_i - 1}\right) f^2\right) \setminus \left\{\prod_{i=1}^{n_2} t_i^{e_i}\right\}} mf.$$

So, the equations obtained by deploying the left-hand term are linear combinations of the equations obtained by deploying other multiples of f with lower degrees.

This second kind of dependencies only appear when $e_1, \ldots, e_{n_2}$ are all odd, so they can be easily prevented by discarding a small fraction of all the monomials leading to equations of some given degrees.

Finally and like in any algebraic system, some linear dependencies necessarily occur between the new equations because of *trivial syzygies*, *i.e.* for example $[f]_i^{\downarrow} [f]_j^{\downarrow} = [f]_j^{\downarrow} [f]_i^{\downarrow}$ [22]. Similarly, the multilinearity structure of the system is likely to give additional linear dependencies [30]. All these dependencies seem hard to prevent at the time of generating the equations (with an appropriate choice of the monomials). On the other hand, their effect also seems smaller than the previous ones. Below, we will experimentally show that a simple randomization strategy does effectively prevent them.

4 Solving the Trapdoor Equation: Algorithm and Analysis

We now give a dedicated new algorithm for solving Equation (4). As suggested above, our algorithm linearizes System (5) by generating enough randomly chosen new equations. We first present the algorithm, then we rigourously analyze its complexity based on a linear independence hypothesis and we experimentally validate this hypothesis. Finally, we describe an improved version of the algorithm and we give some running times for medium-size parameters.

4.1 A Linearization Algorithm

Let n, n_1, n_2 be the parameters of Equation (4). We assume that $n_1 n_2 \geq n$. Let Δ be a small integer. Our algorithm works as follows:

1. Fix d as the smallest integer such that $E(d) - M(d) \geq \Delta$ where $M(d)$ and $E(d)$ are defined in Equations (12) and (13) below.
2. Initiate an empty list of exponents E and an empty list of equations S.
3. For $k = 1, \ldots, \left\lceil \frac{M(d)+\Delta}{n} \right\rceil$ do
   (a) Randomly pick an exponent $e := (e_1, \ldots, e_{n_2}) \in \{0, \ldots, 2^n - 2\}^{n_2}$ such that $e \notin E$, $\prod_{i=1}^{n_2} e_i = 0 \mod 2$ and

$$\max_{i=1,\ldots,n_2} \max\left(\mathrm{HW}(e_i), \mathrm{HW}(e_i + 1)\right) \leq d.$$

(b) Deploy the equation

$$\left(\prod_{i=1}^{n_2} t_i^{e_i}\right) f(t_1,\ldots,t_{n_2}) = 0$$

over $\mathbb{F}_2$.

(c) Add the resulting n equations to S and add e to E.

4. Build a linear system over $\mathbb{F}_2$ with the equations of S, each equation corresponding to one row and each monomial term of degree at most $(d,\ldots,d)$ corresponding to one column.
5. Use a linear algebra algorithm over $\mathbb{F}_2$ to solve the linear system.

The values of $M(d)$ and $E(d)$ will be derived in Section 4.3. The condition in Step 1 ensures that the linear system constructed in Step 4 has more rows than columns. The purpose of the (at least) Δ additional equations is to compensate the expected rank defect of a random square matrix over $\mathbb{F}_2$. In practice, $\Delta = 10$ is sufficient with a probability 99.9%. The algorithm maintains a list with all the monomials already used in order to avoid duplicating equations. The condition "$\prod_{i=1}^{n_2} e_i = 0 \mod 2$" in Step 3 prevents possible linear combinations caused by Frobenius transforms. The saturation effect is prevented too since the degree bound in the equations is the same with respect to all blocks of variables. Trivial syzygies and the additional syzygies coming from the multilinear structure will be ignored in the algorithm and its analysis. We now show that their effect is small and only appears when $E(d)$ is very close to $M(d)$.

4.2 Validating the Linear Independence Hypothesis

To validate the linear independence hypothesis, we implemented our algorithm in Magma and we tested it on various values of the parameters. The results are presented in Table 2. For all these experiments, we fixed $\Delta := 10$. For each value of n and n_1 in the table, the value n_2 was set to $\lceil n/n_1 \rceil$. When $n_1 n_2$ was larger than n, we randomly fixed the last variables of the last block. The value of h was then fixed according to Equation (10) rather than Equation (12). We generated new equations of block-degree at most $(d,\ldots,d)$ from randomly chosen monomials as in the algorithm of Section 4.1.

To compute the rank of the corresponding linear system using Magma, we first multiplied each equation by a power of a homogenization variable (in such a way that all the equations were homogeneous and had the same degree). We then computed a *truncated* Gröbner basis up to the degree $n_2 d$ and we counted the number of elements in this basis. Since the system was homogeneous and we limited the computation to its degree, the program could only perform linear algebra. The *rank deffect* of the original system was deduced as the number of monomials of block-degree at most $(d,\ldots,d)$ minus the number of elements in the truncated basis. Finally, we recovered all the solutions of the system by computing a Gröbner basis of the dezhomogenized truncated Gröbner basis and the field equations. For each set of parameters, we repeated our experiments three times.

Table 2 Experimental rank deffect and number of solutions

n	n_1	n_2	d	$M(d)$	$E(d)$	Rank deffect			Nb Sol		
						Exp1	Exp2	Exp3	Exp1	Exp2	Exp3
10	2	5	1	243	310	0	2	2	0	2	2
10	3	4	1	128	150	3	2	5	1	0	3
10	3	4	2	686	1400000	0	4	2	0	4	2
10	4	3	2	484	70000	1	0	0	1	0	0
11	4	3	2	847	102487	2	0	2	2	0	2
12	4	3	2	1331	145152	0	4	0	0	4	0
13	4	4	2	2662	5198102	0	0	4	0	0	4
13	5	3	2	1792	199927	1	2	0	1	2	0
14	5	3	2	2816	268912	1	2	0	1	2	0
16	7	3	2	3332	458752	1	1	1	1	1	1

For all but one set of parameters, we observed that the rank deffect was equal to the number of solutions of the system, supporting therefore the linear independence hypothesis. For the only "pathological" parameters $(n,n_1) = (10,3)$, we observed a difference of 2 between the rank deffect and the number of solutions. We repeated this experiment 20 times more and we observed a difference of 2 for 19 of them and a difference of 3 in the remaining case. When we tried to increase Δ to 20, we still observed a difference of 2 in all cases. On the other hand, the difference disappeared when we increased d to 2.

These experiments validate the linear independence hypothesis between the new equations. The rank deffects observed when $(n, n_1) = (10, 3)$ are smaller than n, so they cannot be due to linear dependencies between one full block of n equations (corresponding to one monomial over $\mathbb{F}_{2^n}$) and one or more other blocks of equations. For these parameters, $E(d)$ was very close to $M(d)$. Therefore, trivial syzygies and syzygies coming from the block structure (that were not taken into account in our analysis) may have decreased the number of equations available over $\mathbb{F}_2$ below the number of monomials. Although these syzygies may make a strict linearization strategy fail to a small extend in borderline cases, they also *a priori* accelerate Gröbner basis algorithms [30].

The study of syzygies coming from the block structure is beyond the scope of this paper. We emphasize that syzygies are classically ignored in the complexity analysis of Gröbner basis algorithms, because they do not change the general (asymptotical) behavior of Gröbner basis computation. For instance, the complexity given in (1) (Section 1.3) ignore the effect of trivial syzygies. Similarly, our experimental results seem to indicate the same in our context for syzygies coming from the block structure. They only rarely and marginally affect the linear independence hypothesis.

4.3 Complexity Analysis

Since we ignore trivial syzygies, the complexity analysis of our algorithm simply amounts to some combinatorics. The number of monomials of $\mathbb{F}_2[t_{11}, \ldots, t_{n_2 n_1}]$ with degree *exactly* d_1 in the variables of the first block, degree *exactly* d_2 in the variables of the second block, ..., degree *exactly* d_{n_2} in the variables of the last block is

$$\prod_{i=1}^{n_2} \binom{n_1}{d_i}.$$

If the blocks have sizes $n_{1,1}, \ldots, n_{1,n_2}$, the number of monomials of $\mathbb{F}_2[t_{11}, \ldots, t_{n_2 n_1}]$ with degrees at most d with respect to each block is

$$M(d) := \prod_{j=1}^{n_2} \left(\sum_{i=0}^{d} \binom{n_{1,j}}{i} \right). \tag{10}$$

If all the blocks are of the same size – and of degree *at most d* with respect to each block – this simplifies to:

$$M(d) := \left(\sum_{i=0}^{d} \binom{n_1}{i} \right)^{n_2}. \tag{11}$$

The exponents $e \in \{0, \ldots, 2^n - 1\}$ such that the maximum of $\mathrm{HW}(e)$ and $\mathrm{HW}(e+1)$ is equal to d can be separated in two categories. If e is even, those exponents must have Hamming weight equal to $d-1$. If e is odd, they must have Hamming weight equal to d. The two conditions together are equivalent to the following one: the first $n-1$ bits of e have Hamming weight $d-1$. Therefore, the number of such exponents is

$$2\binom{n-1}{d-1}$$

and exactly half of them are odd. Due to the Fröbenius effect explained in Section 3.5, we remove the monomials such that all the exponents are odd. The number of equations available for Step 3 is therefore:

$$E(d) := n(2^{n_2} - 1) \left(\sum_{i=0}^{d-1} \binom{n-1}{i} \right)^{n_2}.$$

For small values of d we have $E(d) < M(d)$. However, $E(d)$ clearly increases much faster than $M(d)$. Therefore, there exists an integer d such that $E(d) > M(d)$. The complexity of Steps 1 to 4 in our algorithm is not much higher than the time to write the matrix constructed; that is essentially $M(d)^2$. The following proposition summarizes the results of this section:

Proposition 5 *Let $M(d)$ be the number of monomials in $\mathbb{F}_2[t_{11},\dots,t_{n_2 n_1}]$ with degree* at most *d with respect to each block, i.e.*

$$M(d) := \left(\sum_{i=0}^{d}\binom{n_1}{i}\right)^{n_2}. \tag{12}$$

Let $E(d)$ be the number of equations generated by the algorithm in degree less than d, that is

$$E(d) := n(2^{n_2}-1)\left(\sum_{i=0}^{d-1}\binom{n-1}{i}\right)^{n_2}. \tag{13}$$

The total time of the linearization-based algorithm described in Section 4.1 is determined by the linear algebra in Step 5 and is bounded by

$$O(M(d)^{\omega})$$

where d is the minimal integer such that $E(d) \geq M(d)$ and w is the linear algebra constant. The total memory cost is $M(d)^2$.

4.4 Asymptotic Complexity Estimates

We now derive asymptotic bounds on the maximal degree d reached in the computation when n_1 and n_2 are fixed as particular functions of n. We will need the following well-known result on binomial coefficients.

Lemma 6 *Let n be an integer and let $\delta, 0 < \delta < 1/2$ be a number such that $\delta n \in \mathbb{N}$. Let $\nu := \frac{\delta}{1-\delta}$. Then the following inequalities hold:*

$$\binom{n}{\delta n} < \sum_{i=0}^{\delta n}\binom{n}{i} < \frac{1}{1-\nu}\binom{n}{\delta n}.$$

PROOF: Let f be the function defined by $f(d) = \frac{d}{n+1-d}$ for $d, 1 \leq d \leqslant n$. The function f is increasing and it satisfies $\binom{n}{d-1} = f(d)\binom{n}{d}$. For any $d, 0 \leqslant d \leqslant \delta n$, we obtain

$$\binom{n}{d-1} = f(d)\binom{n}{d} \leqslant f(\delta n)\binom{n}{d} < \frac{\delta n}{n-\delta n}\binom{n}{d} = \nu\binom{n}{d}.$$

We deduce that $\binom{n}{d-i} \leqslant \nu^i\binom{n}{d}$ for all $i \in \{0,\dots,d\}$. Since $0 < \nu < 1$, we finally get

$$\sum_{i=0}^{\delta n}\binom{n}{i} < \sum_{i=0}^{\delta n}\nu^{\delta n - i}\binom{n}{\delta n} < \frac{1}{1-\nu}\binom{n}{\delta n}.$$

□

We deduce:

Proposition 7 *Let either*

(a) $n_1 := \lceil \alpha n^{\alpha'}\rceil$ and $n_2 := \lceil \beta n^{\beta'}\rceil$ for some constant $\alpha,\beta,\alpha',\beta'$ satisfying $0 < \alpha',\beta' < 1$ and $\alpha,\beta > 0$;
(b) $n_1 := \lceil k\log n\rceil$ and $n_2 := \lceil n/n_1\rceil$ for some constant $k > 0$;
(c) $n_2 := \lceil k\log n\rceil$ and $n_1 := \lceil n/n_2\rceil$ for some constant $k > 0$.

Let also $d := \lceil \frac{\log(n-n_1)}{\log((n-n_1)/n_1)}\rceil$ and let $M(d)$ and $E(d)$ be as in Proposition 5. For any large enough n value, we have $E(d) \geq M(d)$.

PROOF: Let us suppose by contradiction that $M(d) > E(d)$. From the definitions of M and E, we get

$$\sum_{i=0}^{d}\binom{n_1}{i} > (2^{n_2}-1)^{\frac{1}{n_2}} n^{\frac{1}{n_2}}\sum_{i=0}^{d-1}\binom{n-1}{i} > n^{\frac{1}{n_2}}\sum_{i=0}^{d-1}\binom{n-1}{i}.$$

Let $\delta := \frac{d}{n_1}$ and let $\nu := \frac{\delta}{1-\delta}$. For each case of the proposition, we have $\delta < \frac{1}{3}$ when n is large enough. Therefore we also have $\nu < \frac{1}{2}$ and $\frac{1}{1-\nu} < 2$. Using Lemma 6, we get

$$2\frac{n_1(n_1-1)\cdots(n_1-d+1)}{d!} = 2\binom{n_1}{d} > \frac{1}{1-\nu}\binom{n_1}{d} > n^{\frac{1}{n_2}}\binom{n-1}{d-1} = n^{\frac{1}{n_2}}\frac{(n-1)(n-2)\cdots(n-d+1)}{(d-1)!}$$

hence

$$2n_1^d > dn^{\frac{1}{n_2}}(n-d+1)^{d-1} > dn^{\frac{1}{n_2}}(n-n_1)^{d-1}.$$

Taking logarithms, we obtain

$$1 + d\log n_1 > \log d + \frac{\log n}{n_2} + (d-1)\log(n-n_1)$$

hence

$$d\log((n-n_1)/n_1) < \log(n-n_1) - (\log d - 1) - \frac{\log n}{n_2} < \log(n-n_1)$$

and finally

$$d < \frac{\log(n-n_1)}{\log((n-n_1)/n_1)},$$

which provides a contradiction. □

From this, we immediately have:

Corollary 8 *Let n_1, n_2 be as in Proposition 7. Then the asymptotic time complexity of the linearization-based algorithm described in Section 4.1 is:*

$$O(2^{\omega\tau}),\ \textit{where}\ \tau = \frac{n_2\log(n-n_1)\log 2n_1}{\log((n-n_1)/n_1)} \approx \frac{n_2\log n\log n_1}{\log(n/n_1)}. \tag{14}$$

Remark 1 We point out that if System (5) was generic, its resolution with Gröbner basis algorithms would take a time $2^{\omega\tau'}$ with $\tau' = (n_2 n - n + 1)\log n \approx n_2 n \log n$. Our algorithm saves a factor $\frac{n\log(n/n_1)}{\log n_1}$ in the exponent thanks to the particular structure of the problem! For example in Case (a) of Proposition 7, we obtain $\tau \approx \frac{\beta\alpha'}{1-\alpha'}n^{\beta'}\log(n)$ instead of $\tau' \approx \beta n^{1+\beta'}\log(n)$.

4.5 Experimental Results for "Medium" Parameters

The above linearization algorithm provides complexity upper bounds for solving Equation 4. In practice, substantial improvements can often be obtained with Gröbner basis algorithms. However, we emphasize that Gröbner bases must be used with caution over $\mathbb{F}_2$. Except when the regularity degree of the input system is "abnormally small"[2], Gröbner basis algorithms should usually not be used directly. Over small finite fields, the so-called hybrid method [10] (a tricky mix of exhaustive search and Gröbner basis computations) can typically save an exponential factor both in theory and in practice. In the Boolean case, recent work [7] has shown that the hybrid method even allows solving a system of quadratic equations more efficiently than with the exhaustive search, in the sense that the complexity becomes lower than the 2^n barrier.

The equations produced by the algorithm described in Section 4.1 are highly structured. They have a multilinear structure. In fact, the systems have a multihomogeneous structure, but it is enough in practice to apply a Gröbner basis algorithm to a *subset* of the whole system consisting of the multilinear equations only. The system of equations is also greatly overdetermined, i.e. the number of equations is much bigger than the number of variables. Finding the best algorithm for solving multilinear equations over finite fields and determining its complexity is an open problem that is beyond the scope of the present paper. Instead, we design a new *specific* algorithm based on our experiments and intuition.

We now roughly describe the idea behind our algorithm. As a first step, let us first consider the bilinear case. Let F be a set of bilinear equations, i.e. each $f \in F$ is linear with respect to each block of variables X_i for $i = 1, 2$. In [30],

[2] For instance, this is the case of the HFE problem [26] where the regularity degree is a *constant* when the degree of the secret polynomial is fixed.

it was proved that the complexity of computing a Gröbner basis of F strongly depends on $\min(\#X_1, \#X_2)$. Therefore, a natural strategy for solving an overdetermined bilinear Boolean system is to introduce asymmetry by fixing some variables in one block. This is made at the cost of an exhaustive search on the values of these variables. More formally, let k and d be two parameters that we will fix later. Let us assume that $\#X_1 = \#X_2 = n_1$. Our algorithm does the following:

1. Compute a truncated d-Gröbner basis G of F for an elimination ordering $X \gg (X_1 \backslash X) \cup X_2$ where X contains the first k elements of X_1. Hence, $G \subset \mathbb{F}_2[X_1' \cup X_2]$ where $X_1' := X_1 \backslash X$.
2. Perform an exhaustive search on X_1' and compute the resulting Gröbner basis:

```
    for all $\mathbf{z} = (z_1, \ldots, z_{n_1-k}) \in \mathbb{F}_2^{n_1-k}$
        compute a Gröbner basis of $G$ after substituting each variable $x \in X_1'$ by the corresponding $\mathbf{z}$
```

For the parameters, we can choose $d = 2$ and the maximal k such that G is not empty. Since we obtain a linear system in the last step (all the variables of X_1 have been eliminated or specified), the complexity of the algorithm is simply $O(2^{n_1-k} n_1^{\omega})$.

In the general case (multilinear case), the strategy we used to perform actual computations is a natural generalisation of the previous algorithm. Let us assume that all blocks have the same size n_1. Let $[d_1, \ldots, d_{n_2}]$ and $[k_1, \ldots, k_{n_2}]$ be two lists of integer parameters to be fixed later. Suppose we are given G_0, a multilinear system over $\mathbb{F}_2[X_1, \ldots, X_{n_2}]$, i.e. each $g \in G_0$ is linear with respect to each block X_i.

```
For $i \in \{1, \ldots, n_2 - 1\}$ do
  – Compute a truncated $d_i$-Gröbner basis $H_i$ of $G_{i-1}$ for an elimination ordering $X^{(i)} \gg (X_i \backslash X) \cup X_{i+1} \cup \cdots \cup X_{n_2}$
    where $X^{(i)}$ contains the first $k_i$ elements of $X_i$. Hence $H_i \subset \mathbb{F}_2[X_1' \cup \cdots X_i' \cup X_{i+1} \cup \cdots \cup X_{n_2}]$ where $X_i' := X_i \backslash X^{(i)}$.
  – Perform an exhaustive search on $X_i'$ and compute the resulting Gröbner basis:
        for all $\mathbf{z} = (z_1, \ldots, z_{n_1-k_i}) \in \mathbb{F}_2^{n_1-k_i}$
            $G_i :=$ a truncated $d_i-$Gröbner basis of $H_i$ computed after substituting
            each variable $x \in X_i'$ by the corresponding element of $\mathbf{z}$.
```

The last step (when $i = n_2$) reduces to solving a linear system. In Table 3, we report some real experiments. To perform the Gröbner basis computations, we used the FGb package [23]. This software allows displaying the exact number of word operations needed for computing each Gröbner basis. Therefore, we can accurately estimate the complexity of the whole algorithm. For instance when $n = 51$, $n_1 = 17$ and $n_2 = 3$ we chose $d_1 := 3, k_1 := 0$, $d_2 := 2, k_2 := 7$. The whole complexity was $2^{17} \cdot (37429 + 2^{17-7} \cdot 24) = 2^{32.9}$ word operations.

Table 3 Experimental number of operations to solve the Boolean system

n	n_1	n_2	Nb. of Operations
50	25	2	$2^{26.7}$
100	50	2	$2^{53.7}$
51	17	3	$2^{32.9}$
72	24	3	$2^{46.3}$
81	27	3	$2^{52.8}$
60	15	4	$2^{44.9}$
80	20	4	$2^{57.1}$
100	25	4	$2^{71.1}$

5 A New Algorithm for Factoring in $SL(2, \mathbb{F}_{2^n})$

We can now come back to our original problem, namely the factorization problem in $SL(2, \mathbb{F}_{2^n})$.

5.1 Algorithm

Let $\mathscr{S} := \{A, B\}$ be a generator set for $SL(2, \mathbb{F}_{2^n})$ and let $g \in SL(2, \mathbb{F}_{2^n})$. The results of Sections 2 and 4 together lead to the following factoring algorithm:

1. Fix $n_2 < n$ and $n_1 := \lceil \frac{n}{n_2} \rceil$.
2. Replace $\mathscr{S}$ by some $\tilde{\mathscr{S}} := \{E(t + t_i) | t_i \in V_{n_1}\}$ where V_{n_1} is some vector subspace of $\mathbb{F}_{2^n}/\mathbb{F}_2$ obtained with Proposition 3. Similarly, replace g by some $\tilde{g}$.
3. Arbitrarily fix one variable in the last $n_1 n_2 - n$ blocks.
4. Solve Equation 4 using the algorithm of Section 4.1 (extended to arbitrary block sizes).
5. Factor $\tilde{g}$ as a product of the elements of $\tilde{\mathscr{S}}$ using Proposition 4.
6. Deduce a factorization of g as a product of A and B using Proposition 3.

5.2 Complexity Analysis

The value n_2 determines the trade-off between time and message lengths. The solution of Equation (4) in Step 4 provides a factorization of length n_2 of a matrix with eigenvector $\binom{1}{1}$ that can be used in Proposition 4. Therefore, Step 5 computes factorizations of lengths roughly bounded by $8n^2 n_2$. According to Proposition 3, Step 6 provides factorizations of length bounded by

$$32(n_1 + 1)n_2 n^2 3^{n_1 - 1}.$$

These factorizations can always be returned in polynomial-time about

$$4n_1(n_1 + 1)n_2 n^2$$

if they are not returned *explicitly* but in the form of *straight-line programs* [39] (see [42]). A closer look at the proofs of Propositions 10 and 11 in [42] shows that the factorizations lengths are in fact bounded by $32n_2 n^2 3^{n_1 - 1}$ and can be returned in time $4n_1 n_2 n^2$.[3] The overall time and memory complexities are therefore dominated by Step 4, studied in the previous section. Exact values can be computed with the formulae of Section 4.3.

The results for $\omega = 2.807$ (corresponding to Strassen's algorithm [47]), $n \in \{160, 256, 512, 1024\}$ and $n_2, 2 \le n_2 \le n$ are shown in Figure 5.2 together with the $2^{n/2}$ bound of the "birthday search" attacks of [45, 42]. As expected, the parameter n_2 determines the tradeoff between shorter time and memory complexities on one side and shorter factorization lengths on the other side. For "small" parameters including the size $n \approx 160$ originally proposed by Tillich and Zémor, the time complexity of our new attack does not improve very much the $2^{n/2}$ time complexity of the attacks of [45] and [42], whereas the factorizations produced here are larger. However, the interest of our attack increases for larger values of n. For example for $n = 1024$ and $n_2 = 6$, we obtain factorizations of length 2^{306} using time 2^{332} and memory 2^{236}. These numbers should be taken with some caution due to small constants that may be hidden in our estimates, particularly regarding Strassen's algorithm. The unexpected "teeth" in the figures happen when the d value fixed in Step 2 of the algorithm described in Section 4.1 decreases from 2 to 1 or from 3 to 2.

In practice, better complexities can probably be obtained for medium-sized parameters with the heuristic ideas described in Section 4.5.

5.3 Asymptotic Estimates

Asymptotically, our algorithm takes time and memory respectively $2^{\omega\tau}$ and $2^{2\tau}$ where

$$\tau \approx \frac{n_2 \log n \log n_1}{\log(n/n_1)}.$$

As long as $n_1 n_2 \ge n$, we can balance the parameters n_1 and n_2 in different ways. We thereby obtain various tradeoffs between factorization lengths and computing time and memory.

[3] The reason is that any factorization returned by our algorithm for the set $\tilde{\mathscr{S}} = \{M\} \cup \mathscr{O}_{n_1}$ of Proposition 2 contains each orthogonal matrix an even number of times. Therefore, its *normal form* (as defined in the proof of Proposition 10 of [42]) does not contain any orthogonal matrix and the *precomputing phase* in the proof of Proposition 10 of [42] can be avoided.

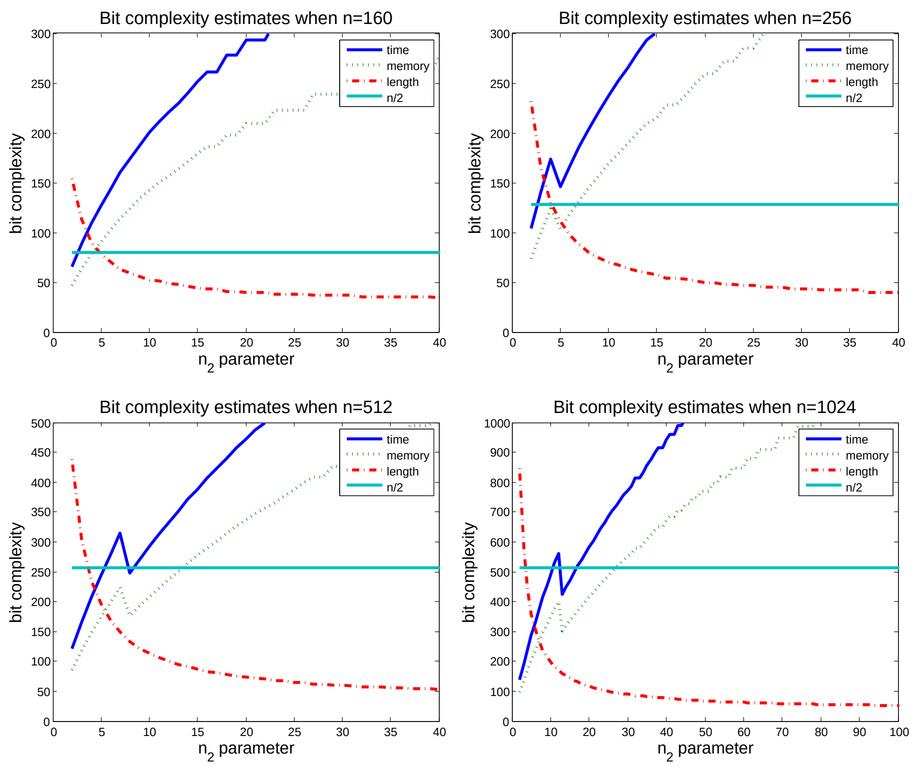

Fig. 2 Complexity bounds for factoring in $SL(2,\mathbb{F}_{2^n})$

In particular, by fixing $n_1 = k\log n$ and $n_2 = \frac{n}{n_1}$, for some "large" $k \in \mathbb{N}$, we obtain factorizations with *polynomial* lengths in *subexponential* time and memory respectively $2^{\omega\tau}$ and $2^{2\tau}$, where

$$\tau \approx \frac{n\log(k\log n)}{k\log(\frac{n}{k\log n})}.$$

In contrast, all previous algorithms [45, 42] working for any generator sets either produced at least subexponential length factorizations or ran in exponential time. On the other hand by fixing $n_2 = k\log n$ and $n_1 = \frac{n}{n_2}$, for some $k \in \mathbb{N}$, we obtain factorizations of *subexponential* lengths in time and memory respectively $2^{\omega\tau}$ and $2^{2\tau}$ where τ is bounded by $\log^3 n$. More generally by choosing $n_2 = n^\alpha$ and $n_1n_2 \approx n$ for any $\alpha, 0 \le \alpha \le 1$, we obtain *subexponential* factorization lengths about

$$32n^3 3^{n^{1-\alpha}}$$

in *subexponential* time and memory roughly

$$2^{\omega(\alpha^{-1}-1)n^\alpha \log n} \quad \text{and} \quad 2^{2(\alpha^{-1}-1)n^\alpha \log n}.$$

This largely improves over the subexponential algorithm of [42] that asymptotically produces factorizations of length $3^{\frac{n}{\alpha\log n}}$ in time and memory roughly $2^{\frac{n}{\alpha\log n}}$.

6 Conclusion

In this paper, we proposed a new algorithm for solving the factorization problem in the group $SL(2,\mathbb{F}_{2^n})$ for any generator set. Our algorithm is the first one to produce factorizations of polynomial lengths in subexponential time. All previous algorithms either ran in exponential time, produced subexponential length factorizations in subexponential time, or were specific to particular generator sets.

To obtain our results, we first reduced the factorization problem in $SL(2,\mathbb{F}_{2^n})$ to the resolution of some constrained multi-linear equation over $\mathbb{F}_{2^n}$. Then, we developed and analyzed a new Gröbner basis algorithm dedicated to the resolution of this equation. Compared to generic Gröbner basis algorithms, our algorithm takes advantage of both the multi-linear structure of the problem and the existence of low degree algebraic relations between the equations.

Our factorization algorithm shows that the factorization problem in $SL(2,\mathbb{F}_{2^n})$ is not as hard as previously expected, even for generic parameters. Our Gröbner basis algorithm is of independent interest.

References

1. L. Babai and Ákos Seress. On the diameter of permutation groups. *Eur. J. Comb.*, 13(4):231–243, 1992.
2. L. Babai, G. Hetyei, W. M. Kantor, A. Lubotzky, and Á. Seress. On the diameter of finite groups. In *FOCS*, volume II, pages 857–865. IEEE, 1990.
3. L. Babai, W. Kantor, and A. Lubotzky. Small-diameter Cayley graphs for finite simple groups. *European Journal of Combinatorics*, 10:507–552, 1989.
4. M. Bardet. *Etude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*. PhD thesis, Université Paris 6, 2004.
5. M. Bardet, J.-C. Faugère, and B. Salvy. Complexity of Gröbner basis computation for semi-regular overdetermined sequences over F_2 with solutions in F_2. Technical Report 5049, INRIA, December 2003. Available at `http://www.inria.fr/rrrt/rr-5049.html`.
6. M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proc. International Conference on Polynomial System Solving (ICPSS)*, pages 71–75, 2004.
7. M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of solving quadratic boolean systems. preprint, 2011.
8. M. Bardet, J.-C. Faugère, B. Salvy, and B.-Y. Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *Proc. of MEGA 2005, Eighth International Symposium on Effective Methods in Algebraic Geometry*, 2005.
9. L. Bettale, J.-C. Faugère, and L. Perret. Cryptanalysis of Multivariate and Odd-Characteristic HFE Variants. In D. C. et al., editor, *Public Key Cryptography - PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 441–458. Springer-Verlag, 2011.
10. Bettale, L. and Faugère, J.-C. and Perret, L. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, 3(3):177–197, 2010.
11. C. Bouillaguet, J.-C. Faugère, P.-A. Fouque, and L. Perret. Practical Cryptanalysis of the Identification Scheme Based on the Isomorphism of Polynomial with One Secret Problem. In D. C. et al., editor, *Public Key Cryptography - PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2011.
12. E. Breuillard, B. Green, and T. Tao. Approximate subgroups of linear groups. arXiv:1005.1881v1, May 2010.
13. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.
14. B. Buchberger. Gröebner bases: an algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*. D. Reidel Publishing Company, 1985.
15. B. Buchberger. Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *J. Symb. Comput.*, 41(3-4):475–511, 2006.
16. B. Buchberger. Comments on the translation of my phd thesis. *J. Symb. Comput.*, 41(3-4):471–474, 2006.
17. S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.
18. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer Verlag, Berlin, Heidelberg, New York, 2005.
19. I. Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.
20. J. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.
21. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, June 1999.

22. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.
23. J.-C. Faugère. FGb: A Library for Computing Gröbner Bases. In K. Fukuda, J. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87, Berlin, Heidelberg, September 2010. Springer Berlin / Heidelberg.
24. J.-C. Faugère, M. S. E. Din, and P.-J. Spaenlehauer. Computing loci of rank defects of linear matrices using gröbner bases and applications to cryptology. In W. Koepf, editor, *ISSAC*, pages 257–264. ACM, 2010.
25. J.-C. Faugère and A. Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003.
26. J.-C. Faugère and A. Joux. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In B. Dan, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer Berlin / Heidelberg, 2003.
27. J.-C. Faugère, A. Joux, L. Perret, and J. Treger. Cryptanalysis of the Hidden Matrix Cryptosystem. In M. Abdalla and P. Barreto, editors, *Progress in Cryptology - LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 241–254. Springer Berlin / Heidelberg, 2010.
28. J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic Cryptanalysis of McEliece variants with compact keys. In *Proceedings of Eurocrypt 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer Verlag, 2010.
29. J.-C. Faugère and L. Perret. Cryptanalysis of $2r^-$ schemes. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 357–372, 2006.
30. Faugère, J.-C. and Safey El Din, M. and Spaenlehauer, P.-J. Gröbner Bases of Bihomogeneous Ideals Generated by Polynomials of Bidegree (1,1): Algorithms and Complexity. *Journal of Symbolic Computation*, 46:406–437, 2011. Available online 4 November 2010.
31. P. Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symb. Comput.*, 44(12):1690–1702, 2009.
32. S. Goldwasser, S. Micali, and R. L. Rivest. A "paradoxical" solution to the signature problem (extended abstract). In *FOCS*, pages 441–448. IEEE, 1984.
33. C. Goyet, J.-C. Faugère, and G. Renault. Algebraic side channel analysis. In *COSADE'11: The 2nd International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 1–6, Fraunhofer SIT, 2011.
34. L. Granboulan, A. Joux, and J. Stern. Inverting HFE is quasipolynomial. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.
35. M. Grassl, I. Ilic, S. S. Magliveras, and R. Steinwandt. Cryptanalysis of the Tillich-Zémor hash function. *J. Cryptology*, 24(1):148–156, 2011.
36. H. A. Helfgott. Growth and generation in $SL_2(Z/pZ)$, 2005.
37. S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.
38. A. Lauder. Continued fractions of laurent series with partial quotients from a given set. Acta Arithmetica XC.3, 1999.
39. N. A. Lynch. Straight-line program length as a parameter for complexity analysis. *J. Comput. Syst. Sci.*, 21(3):251–280, 1980.
40. J. P. Mesirov and M. M. Sweet. Continued fraction expansions of rational expressions with irreducible denominators in characteristic 2. *Journal of Number Theory*, 27:144–148, 1987.
41. J. Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In *EUROCRYPT*, pages 33–48, 1996.
42. C. Petit. Towards factoring in $SL(2,\mathbb{F}_{2^n})$. Preprint, 2011.
43. C. Petit, K. Lauter, and J.-J. Quisquater. Full cryptanalysis of LPS and Morgenstern hash functions. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2008.
44. C. Petit and J.-J. Quisquater. Preimages for the Tillich-Zémor hash function. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 282–301. Springer, 2010.

45. C. Petit, J.-J. Quisquater, J.-P. Tillich, and G. Zémor. Hard and easy components of collision search in the Zémor-Tillich hash function: New attacks and reduced variants with equivalent security. In M. Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2009.
46. L. Pyber and E. Szabó. Growth in finite simple groups of Lie type. arXiv:1001.4556v1, Jan 2010.
47. V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.
48. J.-P. Tillich and G. Zémor. Hashing with SL_2. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1994.
49. J.-P. Tillich and G. Zémor. Collisions for the LPS expander graph hash function. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2008.