DIGITAL SIGNATURE SCHEMES USING NON-SQUARE MATRICES OR SCRAP AUTOMORPHISMS

JIALE CHEN, DIMA GRIGORIEV, AND VLADIMIR SHPILRAIN

ABSTRACT. We offer two very transparent digital signature schemes: one using non-square matrices and the other using scrap automorphisms. The former can be easily converted to a public key encryption scheme.

1. INTRODUCTION

Due to the concern that if large-scale quantum computers are ever built, they will compromise the security of many commonly used cryptographic algorithms, NIST had begun in 2016 a process to develop new cryptography standards and, in particular, solicited proposals for new digital signature schemes [7] resistant to attacks by known quantum algorithms, such as e.g. [9].

One possible way to avoid quantum attacks based on solving the hidden subgroup problem (including the attacks in [9]) is *not* to use one-way functions on sets that have an obvious (semi)group structure. In this paper, we offer two digital signature schemes that are similar in spirit. What they have in common is the following idea. There is a function $F : X \to Y$ that is not necessarily one-way. However, when only a *part* (call it $\bar{Y}$) of Y is published, the restriction $\bar{F} : \bar{X} \to \bar{Y}$ becomes much harder to invert and therefore becomes a candidate for a one-way function.

One of the simplest instantiations of this general idea is employed in our first digital signature scheme. There, X is an invertible square matrix over a commutative ring and Y is X^{-1}. The function F therefore takes a square matrix as the input and outputs the inverse matrix. This is often not a one-way function. However, if we only publish a subset of columns of X^{-1}, then finding the corresponding subset of rows of X appears to be a hard problem. The corresponding digital signature scheme is described in Section 2. We note that the verification part in this scheme is very efficient since it amounts to multiplying a vector of sparse polynomials by a matrix whose entries are sparse polynomials (over $\mathbb{Z}_q$, for a small q).

For our second scheme, we consider an automorphism $\alpha : K \to K$ of an algebra K of polynomials in n variables $x_1, \ldots, x_n$ over a ring R. Such an automorphism can be given by n polynomials $\alpha(x_i)$. It is arguable whether or not computing α^{-1} from all $\alpha(x_i)$ is a hard problem (this might depend on the ring R). Some of the known methods use Gröbner bases techniques (see e.g. [10]) that are not known to be efficient. A more efficient algorithm was reported in [2], but it is still exponential-time in the number of variables, although polynomial-time in the number of polynomials and their degrees. We note that there was a proposal some time ago for a public key cryptosystem based on the alleged hardness of inverting polynomial automorphisms [6], but it was subsequently attacked in [3].

In the present paper, we suggest to publish not all of the $\alpha(x_i)$, but only some of them, with the idea that inverting such a "scrap" automorphism should be harder than inverting the "whole" α. To the best of our knowledge, the only known way to invert a "scrap" automorphism is the "brute force" trial-and-error method.

Our digital signature scheme based on "scrap" automorphisms is described in Section 11.

Finally, we note that our digital signature scheme based on scrap matrices can be easily converted to a public key encryption scheme, see Section 10. We do not make any security claims about this scheme though since that would require a separate security analysis, and this is not in the scope of the present paper.

2. Non-square matrices: scheme description

Let $K = \mathbb{Z}_q[x_1, \ldots, x_n]$ denote the algebra of polynomials in n variables over the ring $\mathbb{Z}_q$ of integers modulo q, where an integer q is not necessarily a prime.

The signature scheme is as follows.

Public:
– $k \times l$ left invertible matrix M, with $k > l$, whose entries are sparse polynomials from the algebra K.
– a hash function H and a (deterministic) procedure for converting values of H to vectors of sparse polynomials from the algebra K.

Private: $l \times k$ right invertible matrix L over K, such that LM is the $l \times l$ identity matrix.

Signing a message m:

(1) Apply a hash function H (e.g., SHA-512) to m. Convert $H(m)$ to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of l (sparse) polynomials from the algebra K using a deterministic public procedure.

(2) Multiply the vector $\mathbf{U}$ by the (private) matrix L on the right to get a vector $\mathbf{V} = \mathbf{U}L = (Q_1, \ldots, Q_k)$ of k polynomials from K.

(3) The signature is the vector $\mathbf{V}$.

Verification.

1. The verifier computes the hash $H(m)$ and converts $H(m)$ to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of l (sparse) polynomials using a deterministic public procedure.

2. The verifier multiplies the signature vector $\mathbf{V}$ by the public matrix M on the right to get a vector $\mathbf{W}$.

3. The signature is accepted if and only if $\mathbf{W} = \mathbf{U}$.

Correctness is obvious since $\mathbf{W} = \mathbf{V}M = (\mathbf{U}L)M = \mathbf{U}(LM) = \mathbf{U}$.

Remark 1. *One can use a faster, numerical, variant of the signature verification. Namely, instead of computing a vector $\mathbf{W}$ of polynomials, the verifier can assign random numerical values (from $\mathbb{Z}_q$) to all variables x_i in the vectors $\mathbf{U}$, $\mathbf{V}$ and the matrix M, and then do computation in $\mathbb{Z}_q$. To make the probability of the "false positive" decision negligible, assigning random numerical values should be done several times.*

3. Key generation

In this section, we answer crucial questions on key generation in our scheme: how to generate a (left) invertible $k \times l$ matrix, how to compute its inverse, and how to generate a random sparse polynomial. We emphasize that in our scheme, all this is done offline.

3.1. **Generating a random t-sparse polynomial.** We call a polynomial t-sparse if it has t monomials. To generate a "random" t-sparse polynomial, we first generate t monomials as follows.

1. Bound the degree of a monomial by an integer b (one of the parameters of the scheme).

2. Select, uniformly at random, an integer d between 1 and b. This integer will be the degree of our monomial.

3. To select a monomial of degree d, we do a selection of x_i, uniformly at random from $\{x_1, \ldots, x_n\}$, d times. Then build the monomial as a product of the selected x_i.

4. Having selected t monomials like that, build a t-sparse polynomial as a linear combination of t selected monomials with random coefficients from $\mathbb{Z}_q - \{0\}$.

3.2. **Generating a (left) invertible $k \times l$ matrix.** Let $k > l$. To generate a (left) invertible $k \times l$ matrix, one can first generate an invertible square $k \times k$ matrix and then remove $k - l$ columns, selected at random.

To generate an invertible square $k \times k$ matrix, one can do the following.

1. Generate an upper unitriangular $k \times k$ matrix U with random t-sparse polynomials above the diagonal and a lower unitriangular $k \times k$ matrix L with random t-sparse polynomials below the diagonal.

2. Make a product of matrices that would have the form UP_1LP_2, where P_i are matrices corresponding to random permutations of columns and rows of a $k \times k$ matrix. (Entries of P_i are 0s and 1s.)

3.3. **Computing the (left) inverse of a matrix.** Having generated an invertible square matrix $S = UP_1LP_2$ (Section 3.2), we compute its inverse as
$S^{-1} = P_2^{-1}L^{-1}P_1^{-1}U^{-1}$. Computing P_i^{-1} is trivial, and computing the inverse of a unitriangular square matrix X is done by using the adjugate of X. (Note that the determinant of any unitriangular square matrix is equal to 1.)

Now suppose the (left) invertible $k \times l$ matrix M was obtained from the square $k \times k$ matrix S by removing $k - l$ columns $C_{i_1}, \ldots, C_{i_{k-l}}$. Then, to get a left inverse of M, we just remove the corresponding rows $R_{i_1}, \ldots, R_{i_{k-l}}$ from S^{-1}.

3.4. **Converting $H(m)$ to a vector of sparse polynomials.** We suggest using a hash functions from the SHA-2 family, specifically SHA-512. We assume the security properties of SHA-512, including collision resistance and preimage resistance. The following procedure may seem "too ad hoc", but this is needed to make sure that the linear space of possible monomials obtained by this procedure has a large dimension, to prevent the forgery attack described in Section 7.1.

Let $B = H(m)$ be a bit string of length 512. We will convert B to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of l (4-sparse) polynomials from the algebra K, where each monomial has degree at most 10.

We note that this process is deterministic. Of course, this is just one of the many possible conversion procedures; one of our goals here is to only have polynomials P_i of low degree.

(1) Discard several last bits in B to make its length divisible by l. With our suggested parameters (see Section 4), $l = 5$, so one can discard, say, the last 12 bits of B.

(2) Split the remaining 500 bits in two parts: 300 in one part and 200 in the other. The part with 300 bits will determine a map of the set $\{x_1, \ldots, x_{50}\}$ of variables to the set $\{x_1, \ldots, x_{63}\}$, as follows.

Each integer from 1 to 50 can be represented in the binary form using 6 bits. Thus, $300 = 6 \cdot 50$ bits will naturally determine a map $x_i \to x_j$ for each i from 1 to 50. For example, if the first 6 bits of these 300 are 011010, this means that x_1 will be mapped to x_{26} because 011010=26.

This map is not necessarily one-to-one (since some of the 6-bit blocks may be identical) but this is not a problem. Let $\{x_{i_1}, \ldots, x_{i_{50}}\}$ be the new sequence of variables; some of them may be equal.

(3) Split the remaining 200 bits in $l = 5$ 40-bit blocks. Then split each 40-bit block in 4 parts, 10 bits in each. This will correspond to 4 monomials, of degree at most 10 each, as follows. Enumerate bits in each 40-bit block by $x_{i_1}, \ldots, x_{i_{40}}$ (in this order, going left to right). Now each block of 10 bits is converted to a monomial that is a product of x_i corresponding to the places in the bit string where the bit is "1". In particular, each monomial will be of degree at most 10. For example, if the first 10-bit block is 1001100011, then the corresponding monomial will be $x_{i_1} x_{i_4} x_{i_5} x_{i_9} x_{i_{10}}$.

(4) Combine 4 monomials obtained at Step (3) into a 4-sparse polynomial by selecting coefficients from $\mathbb{Z}_q$ as follows. We are going to use the string of 12 bits that we have discarded at Step (1). Split it in 4 blocks of 3 bits. Each block will determine a coefficient at a monomial in a 4-sparse polynomial by converting a 3-bit binary number to decimal and reducing modulo 6.

4. Suggested parameters

For the hash function H, we suggest SHA-512.

For the integer q in $\mathbb{Z}_q$, we suggest $q = 6$.

For the number n of variables, we suggest $n = 64$.

For the dimensions of the matrix M, we suggest $k = 10$, $l = 5$. However, due to current software limitations (see Section 6 for more details), our present implementation uses $k = 5$, $l = 3$.

For the number t of monomials in t-sparse polynomials that entries of the unitriangular matrices, we suggest $t = 3$.

For the bound b on the degree of monomials in t-sparse polynomials, we suggest $b = 3$.

5. What is the hard problem here?

The (computationally) hard problem that we employ in our construction is finding a left inverse of a given left invertible matrix M. That is, solving the matrix equation $XM = I$, where X is the unknown matrix of given dimensions.

If matrices in this equation were considered over a field, then this matrix equation would translate to a system of linear equations (in the entries of the matrix X), and therefore would be easily solvable.

In our situation, where matrices are considered over a polynomial algebra, the problem can still be reduced to a system of linear equations, this time in the coefficients of polynomials that are entries of the matrix X. If the number n of variables x_i is not too small (we suggest $n = 64$), then the number of different monomials (and therefore the number of unknown coefficients of polynomials) is quite large (see Section 7), so that the relevant system of linear equations becomes intractable.

5.1. Completing a left invertible matrix to an invertible square matrix. Another
possible way to find a left inverse of a given left invertible matrix M is to complete M to an invertible square matrix by adding more columns, and then find the inverse of this square matrix (which is relatively easy since one can use determinants).

Perhaps surprisingly, it was a major open problem (Serre's problem) in algebra whether or not any left invertible matrix over a polynomial algebra can be completed to an invertible square matrix by adding more rows (or columns). This problem was answered in the affirmative independently by Quillen [8] and Suslin [11].

The question that matters to us though is that of the computational complexity of completing M to an invertible square matrix. It was shown in [1] (for polynomial algebras over an infinite field) and then in [5] (for polynomial algebras over an arbitrary Noetherian ring) that there is a relevant algorithm whose complexity is exponential in the square of the number of variables x_i. This is yet another reason why the number of variables should not be too small.

6. Performance and signature size

Disclaimer. *We have run into a problem with software for symbolic computation with multivariate polynomials, especially with matrices over multivariate polynomials. None of the symbolic computation packages either in Julia or in ANSI C (Flint) is capable of doing expand-and-simplify if the number of monomials becomes as large as several hundred. The only software that is capable of doing that is Maple, but since it is not common to use Maple for cryptographic purposes, we have decided not to use it; instead, we have reduced the values of the most impactful parameters, the dimensions of the public matrix M, from the recommended 10×5 to 5×3. With Flint evolving, it should be soon possible to handle 10×5 matrices over multivariate polynomials.*

For our computer simulations, we used Apple MacBook Pro, M1 CPU (8 Cores), 16 GB RAM computer.

With the suggested parameters, signature verification takes about 0.2 sec on average, which is not bad, but the signature is rather large, about 4,200 bytes on average.

The size of the private key (the matrix L) is about 2,000 bytes, and so is the size of the public key (the matrix M).

We note that we have measured the size of a signature, as well as the size of private/public keys, as follows. We have counted the total number of variables that occurred in relevant polynomial(s) and multiplied that number by 7, the number of bits sufficient to describe the

index of any variable (except x_{64}). To that, we added the number of monomials times 2 (the average number of bits needed to describe a coefficient at a monomial in our construction(s)).

Performance metrics for given parameter values							
# monomials in sparse polynomials (parameter t)	max degree of monomials (parameter b)	dimensions of private matrix L	verification time (sec)	signature size (Kbytes)	public key size (Kbytes)	private key size (Kbytes)	memory usage (Mbytes)
3	3	3x5	0.2	4.2	2	1.9	1.4

7. Linear algebra attacks

One can attempt to recover the private left inverse of the public matrix M by using linear aOne can attempt to recover the private left inverse of the public matrix M by using linear algebra, as follows. Finding the left inverse matrix of M is equivalent to solving the matrix equation $XM = I$, where X is the unknown matrix of given dimensions.

If matrices in this equation were considered over a field, then this matrix equation would translate to a system of linear equations (in the entries of the matrix X), and therefore would be easily solvable.

In our situation, where matrices are considered over a polynomial algebra, the problem can still be reduced to a system of linear equations, this time in the coefficients of polynomials that are entries of the matrix X. If the number n of variables x_i is not too small (we suggest $n = 64$), then the number of different monomials (and therefore the number of unknown coefficients of polynomials) is very large, so that the relevant system of linear equations becomes intractable.

Note that, although monomials in the public matrix M have degrees bounded by 3, in the left inverse of M there can be monomials of significantly higher degree. Let us estimate the number of monomials of degree, say, 15 in 64 variables.

By the formula for the number of combinations with repetitions, the number is $\binom{78}{15} \approx 2^{52}$.

Thus, the total number of potentially possible monomials in a 3×6 matrix with sparse polynomial entries is more than $2^{52} \cdot 15 > 2^{55}$, and this would be the number of unknowns in a system of linear equations. The number of equations would be at least as large.

At least $(2^{55})^{2.3} \approx 2^{126}$ arithmetic operations are needed to solve such a system of linear equations, according to our understanding of the state-of-the-art in solving systems of linear equations.

7.1. **Forgery after collecting multiple signatures.** A smarter linear algebra attack is targeted at forging signatures without recovering the private key and is based on collecting multiple vectors $\mathbf{U} = (P_1, \ldots, P_l)$ and finding a basis of the linear space generated by these vectors.

Since the transformation $\mathbf{U} \to \mathbf{V} = \mathbf{U}L$ is linear, after a basis of the linear space of vectors $\mathbf{U}$ is constructed, the forger can express each newly obtained $\mathbf{U}$ as a linear combination of previously collected ones, and then the signature corresponding to this newly obtained $\mathbf{U}$ will be the same linear combination of the relevant previously published relevant signatures $\mathbf{V}$.

To make this attack infeasible, the dimension of the linear space of possible vectors $\mathbf{U}$ should be quite large. This dimension is equal to the number of monomials that can possibly appear in polynomials $P_1, \ldots, P_l$ under the procedure of converting $H(m)$ to a vector of sparse polynomials, see Section 3.4. The number of monomials of degree $k \leq 64$ in 64 variables is $\binom{64+k-1}{k}$, so in our situation, where the degree can be any number between 1 and 15, the number of possible monomials is at least $\binom{78}{15} \approx 2^{52}$. This is about how many *linearly independent* signatures a forger has to collect. When collecting, say, 1000 linearly independent signatures a second, this would take over 50000 years.

8. Security claims

With suggested parameters, the linear algebra "brute force" attack requires at least 2^{126} arithmetic operations in $\mathbb{Z}_6$ to solve the relevant system of linear equations.

There could be ad hoc attacks on the public key aiming at recovering some of the entries of the private matrix, but recovering only some of the entries cannot make the probability of passing verification non-negligible because getting even just a couple of entries (out of 15) wrong will not increase chances of passing verification.

Forgery without getting a hold of the private key requires accumulation of at least 2^{52} linearly independent signatures.

As for quantum security, it appears that no known quantum algorithms are applicable in our situation since there are no abelian (semi)groups in play here that could be exploited by one of the known quantum attacks.

9. Advantages and limitations of the matrix digital signature scheme

9.1. Advantages.

1. Unparalleled conceptual simplicity.

2. Simplicity of design. Signing a message amounts to just a matrix-vector multiplication, and so does verifying a signature.

3. Bringing into play an interesting hard problem – inverting a non-square matrix over multivariate polynomials. This problem has not been considered in the cryptographic context before, to the best of our knowledge.

4. Efficiency of the signature verification (about 0.2 sec on average with parameters in the current implementation). To be fair, if we did not have to downsize dimensions of the public matrix from 10×5 to 5×3 (see the Disclaimer in Section 6), then we project the signature verification time would be close to 1 sec.

9.2. Limitations.

1. Linearity of the signing function. To avoid linear algebra attacks, one has to use a large number of variables in the polynomial algebra, and this can affect the signature size (but not so much the verification time).

2. Large signature size.

10. Public key encryption

Our first digital signature scheme (Section 2) can be easily converted to a public key encryption scheme, as follows. We do not make any security claims about this scheme though since that would require a separate security analysis, and this is not in the scope of the present paper.

Private: $k \times l$ left invertible matrix M, with $k > l$, whose entries are sparse polynomials from the algebra K.

Public: $l \times k$ right invertible matrix L over K, such that LM is the $l \times l$ identity matrix.

Encrypting a message m:

(1) Convert m to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of l (sparse) polynomials.

(2) Multiply the vector $\mathbf{U}$ by the (public) matrix L on the right to get a vector $\mathbf{V} = (Q_1, \ldots, Q_k)$ of k polynomials from K. This vector $\mathbf{V}$ is the encryption of m.

Decryption. Multiply the vector $\mathbf{V}$ by the private matrix M on the right to get $\mathbf{V}LM = \mathbf{U}$.

11. Scrap automorphisms: scheme description

This scheme is similar in spirit to the scheme in Section 2. It is also similar to that scheme in some key generation details, in particular in sampling t-sparse polynomials. Also, converting a hash $H(m)$ to a sparse polynomial is similar, so we chose not to duplicate Section 3.4 here.

Again, let $K = \mathbb{Z}_q[x_1, \ldots, x_n]$ denote the algebra of polynomials in n variables over the ring $\mathbb{Z}_q$ of integers modulo q, for an integer q that is not necessarily a prime.

Let $\alpha : K \to K$ be an automorphism of the algebra K given by n polynomials $\alpha(x_i)$. The signature scheme is as follows.

Public: k polynomials $y_{i_1} = \alpha(x_{i_1}), \ldots, y_{i_k} = \alpha(x_{i_k})$, where $k < n$.

Private: The inverse α^{-1} of the automorphism α, given by n polynomials $z_i = \alpha^{-1}(x_i)$.

Signing a message m:

1. Apply the hash function H (e.g., SHA-512) to m. Convert $H(m)$ to a sparse polynomial Q in $y_{i_1}, \ldots, y_{i_k}$ using a deterministic public procedure (see e.g. Section 3.4).

2. Apply the automorphism α^{-1} to the polynomial Q; the result is $Q(x_{i_1}, \ldots, x_{i_k})$. Denote the latter polynomial by S.

3. The signature is the polynomial $S = \alpha^{-1}(Q)$.

Verification:

1. The verifier computes $H(m)$ and converts $H(m)$ to $Q = Q(y_{i_1}, \ldots, y_{i_k})$ using a deterministic public procedure.

2. The verifier applies the public automorphism α to the polynomial $S = Q(x_{i_1}, \ldots, x_{i_k})$.

3. The signature is accepted if and only if $\alpha(S)$ coincides with $Q = Q(y_{i_1}, \ldots, y_{i_k})$.

Correctness is obvious since $\alpha(S) = \alpha(\alpha^{-1}(Q)) = \alpha(Q(x_{i_1}, \ldots, x_{i_k})) = Q(y_{i_1}, \ldots, y_{i_k})$.

11.1. Generating an automorphism α.

An automorphism α is generated offline, as follows. Recall that a polynomial is *t-sparse* if it only has t monomials.

(1) Choose an index k at random between 1 and n, where n is the number of variables in our polynomial algebra.

(2) Take x_k to $x_k + h(x_1, \ldots, x_n)$, where $h(x_1, \ldots, x_n)$ is a random t-sparse polynomial not depending on x_k. Fix all other variables.

(3) Apply a random permutation to the set $x_1, \ldots, x_n$ of variables.

(4) Repeat steps (1) through (3) l times (we suggest $l = n$).

Keep record of all "elementary" automorphisms (Steps (2) and (3)) used in the process of building α.

11.2. Computing the automorphism α^{-1}.

Having generated the automorphism α (see the previous subsection), we now have to compute α^{-1}, the private key of the signatory.

To do that, we trace back the "elementary" automorphisms (see Section 11.1) and invert them. More specifically, if α is a composition $\alpha = \varepsilon_1 \cdots \varepsilon_r$ (with ε_1 applied first), then $\alpha^{-1} = \varepsilon_r^{-1} \cdots \varepsilon_1^{-1}$.

To invert a permutation is straightforward, and if ε_j is as in Step (2) in the previous Section 11.1, then ε_j^{-1} takes x_k to $x_k - h(x_1, \ldots, x_n)$ and fixes all other variables.

11.3. Sampling a t-sparse polynomial.

The number t of monomials is one of the parameters of the scheme, see Section 11.4. Sampling a t-sparse polynomial can be done the same way as in Section 11.3.

11.4. Suggested parameters.

Again, for the integer q in $\mathbb{Z}_q$, we suggest $q = 6$.

For the number n of variables, we suggest $n = 32$.

For the number k of published polynomials $y_i = \alpha(x_i)$, we suggest $k = 16$.

For the number t of monomials in t-sparse polynomials, we suggest $t = 3$.

For the degree bound b of monomials in t-sparse polynomials, we suggest $b = 3$.

For the number s of elementary automorphisms that compose to an automorphism α, we suggest $s = 16$.

11.5. Estimating the size of the key space with suggested parameters.

As we have mentioned in the Introduction, the only known way to invert a "scrap" automorphism is the "brute force" trial-and-error method. It is therefore natural to ask for a complexity estimate of such a method. In other words, what is the size of the key space that a brute force attacker would have to exhaust?

First, let us estimate the number of monomials of degree $d \leq 3$ in 32 variables. By the formula for the number of combinations with repetitions, this number is $\binom{34}{3} + \binom{33}{2} + \binom{31}{1} = 6,543$.

Thus, the number of t-sparse polynomials with $t = 3$ is at least $\binom{6,543}{3} \cdot 5^3 > 2^{42}$. (The factor 5^3 appears because our polynomial is a linear combination of 3 monomials with coefficients from $\mathbb{Z}_6 - \{0\}$.) Then the number of "elementary" automorphisms (as in Section 11.1) is

at least $2^{42} \cdot 32 = 2^{47}$. The number of compositions of $s = 16$ elementary automorphisms is therefore at least $(2^{42})^{16} = 2^{672}$.

References

[1] L. Caniglia, G. Cortilnas, S. Danon, J. Heintz, T. Krick and P. Solerno, *Algorithmic aspects of Suslin's proof of Serre's conjecture*, Comput. Complexity **3** (1993), 31–55.

[2] A. L. Chistov, D. Yu. Grigoriev, *Complexity of quantifier elimination in the theory of algebraically closed fields*, in: Mathematical foundations of computer science (Prague, 1984), Lecture Notes in Comput. Sci. **176**, Springer, Berlin, 1984, 17–31.

[3] J. Ding, T. Hodges, *Cryptanalysis of an implementation scheme of the tamed transformation method cryptosystem*, J. Algebra Appl. **3** (2004), 273–282.

[4] D. Yu. Grigoriev, M. Karpinski, M. F. Singer, *The interpolation problem for k-sparse sums of eigenfunctions of operators*, Adv. in Appl. Math. **12** (1991), 76–81.

[5] H. Lombardi, I. Yengui, *Suslin's algorithms for reduction of unimodular rows*, J. Symbolic Comput. **39** (2005), 707–717.

[6] T. T. Moh, *A public key system with signature and master key functions*, Comm. Algebra **27** (1999), 2207–2222.

[7] NIST: Post-Quantum Cryptography: Digital Signature Schemes, `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf`

[8] D. Quillen, *Projective modules over polynomial rings*, Invent. Math. **36** (1976), 167–171.

[9] P. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput. **26** (1997), 1484–1509.

[10] V. Shpilrain and J.-T. Yu, *Polynomial automorphisms and Gröbner reductions*, J. Algebra **197** (1997), 546–558.

[11] A. Suslin, *Projective modules over polynomial rings are free*, Dokl. Akad. Nauk SSSR **229** (1976), 1063–1066.

Department of Mathematics, The City College of New York, New York, NY 10031
Email address: `jchen056@citymail.cuny.edu`

CNRS, Mathématiques, Université de Lille, 59655, Villeneuve d'Ascq, France
Email address: `Dmitry.Grigoryev@univ-lille.fr`

Department of Mathematics, The City College of New York, New York, NY 10031
Email address: `shpilrain@yahoo.com`