

Fast Encoding of AG Codes over C_{ab} Curves

Peter Beelen, *Member, IEEE*, Johan Rosenkilde, and Grigory Solomatov.

Abstract

We investigate algorithms for encoding of one-point algebraic geometry (AG) codes over certain plane curves called C_{ab} curves, as well as algorithms for inverting the encoding map, which we call “unencoding”. Some C_{ab} curves have many points or are even maximal, e.g. the Hermitian curve. Our encoding resp. unencoding algorithms have complexity $\tilde{O}(n^{3/2})$ resp. $\tilde{O}(qn)$ for AG codes over any C_{ab} curve satisfying very mild assumptions, where n is the code length and q the base field size, and \tilde{O} ignores constants and logarithmic factors in the estimate. For codes over curves whose evaluation points lie on a grid-like structure, notably the Hermitian curve and norm-trace curves, we show that our algorithms have quasi-linear time complexity $\tilde{O}(n)$ for both operations. For infinite families of curves whose number of points is a constant factor away from the Hasse–Weil bound, our encoding algorithm has complexity $\tilde{O}(n^{5/4})$ while unencoding has $\tilde{O}(n^{3/2})$.

Index Terms

Encoding, AG code, Hermitian code, Cab code, norm-trace curve

I. INTRODUCTION

In the following \mathbb{F} is any finite field, while \mathbb{F}_q denotes the finite field with q elements. An \mathbb{F} -linear $[n, k]$ code is a k -dimensional subspace $\mathcal{C} \subseteq \mathbb{F}^n$. A substantial part of the literature on codes deals with constructing codes with special properties, in particular high minimum distance in the Hamming metric. In this context, algebraic geometry (AG) codes, introduced by Goppa [12], have been very fruitful: indeed, we know constructive families of codes from towers of function fields whose minimum distance beat the Gilbert–Varshamov bound [34]. Roughly speaking such codes arise by evaluating functions in points lying on a fixed algebraic curve defined over \mathbb{F} . The evaluation points should be rational, i.e., defined over \mathbb{F} .

The well-known Reed–Solomon (RS) codes is a particularly simple subfamily of the AG codes. Arguably the most famous class of AG codes which are not RS codes is those constructed using the Hermitian curve; the Hermitian curve is a maximal curve, i.e. the number of rational points

meets the Hasse–Weil bound [32, Theorem 5.2.3]. It is an example of the much larger family of C_{ab} curves, which are plane curves given by a bivariate polynomial equation $H(X, Y) \in \mathbb{F}[X, Y]$ with several additional regularity properties. These imply that the function field associated to a C_{ab} curve has a single place at infinity, P_∞ , and any function with poles only here can be represented by a bivariate polynomial $f \in \mathbb{F}[x, y]$ whose degree is bounded by a function of the pole order at the place infinity. Here, x and y are two functions which satisfy $H(x, y) = 0$ and hence $\mathbb{F}[x, y]$ is not a normal polynomial ring.

This means that the computations needed for operating with one-point AG codes over C_{ab} curves is much simpler than the general AG code case. The most well-studied operation pertaining to codes is decoding, i.e. obtaining a codeword from a noisy received word. For general AG codes, the fastest decoding algorithms essentially revert to linear algebra and have complexity roughly $\mathcal{O}(n^3)$, where n is the length of the code, e.g. [21], [31]. However, for one-point C_{ab} codes, we have much faster algorithms, e.g. [1]. In [28], we studied Hermitian codes, i.e. AG codes over the Hermitian curve and obtained a decoding algorithm with complexity roughly $\tilde{\mathcal{O}}(n^{5/3})^1$.

A somewhat overlooked problem for AG codes, however, is the *encoding*, i.e. the computational task of obtaining a codeword $\mathbf{c} \in \mathcal{C}$ belonging to a given message $\mathbf{m} \in \mathbb{F}^k$. Given a message \mathbf{m} and a generator matrix $G \in \mathbb{F}^{k \times n}$ of the code, a natural encoder is obtained as the vector-matrix product $\mathbf{c} = \mathbf{m}G$. In general, this costs roughly $2kn$ operations in the field \mathbb{F} . Asymptotically, keeping the rate k/n fixed, this means that for an arbitrary code the encoding can be done in $\mathcal{O}(n^2)$ operations in \mathbb{F} .

Encoding is mathematically and intuitively a much simpler process than decoding, so it would be highly surprising for an algebraic code to admit faster decoding than encoding. Nonetheless, this is currently the situation in the literature for Hermitian codes, where we can decode in $\tilde{\mathcal{O}}(n^{5/3})$, while to the authors' knowledge, there is no published encoding algorithm beating the naive $\mathcal{O}(n^2)$ complexity. In this article, we study encoding of one-point AG codes over any C_{ab} curve with the aim of leveraging the polynomial ring-like structure of functions with poles at P_∞ . We then show that this algorithm works particularly efficiently on certain C_{ab} codes like the Hermitian codes: indeed, our encoding algorithm for Hermitian codes has quasi-linear complexity $\tilde{\mathcal{O}}(n)$ in the code length.

¹ Formally, for a function $f(n)$, then $\tilde{\mathcal{O}}(f(n)) = \bigcup_{c=0}^{\infty} \mathcal{O}(f(n) \log^c(f(n)))$.

Encoding of AG codes has received very little attention in the literature. For the particularly simple case of RS codes, it is classical that they can be encoded in quasi-linear complexity [16] by univariate multipoint evaluation (see Section II-C). For carefully tailored asymptotically good sub-codes of AG codes arising from the Garcia-Stichtenoth tower [10], [27] give a encoding algorithm which is faster than $\mathcal{O}(n^{3/2})$ by reducing the encoding to matrix multiplication.

The inverse process of encoding, which we will call *unencoding*, matches a given codeword \mathbf{c} with the sent message \mathbf{m} . If the encoder was systematic this is of course trivial, but for an arbitrary linear encoder computing this inverse requires finding an information set for the code and inverting the generator matrix at those columns. This matrix inverse can be precomputed, in which case the unencoding itself is simply a $k \times k$ vector-matrix multiplication costing $\mathcal{O}(k^2)$, which for a fixed rate equals $\mathcal{O}(n^2)$. We give faster algorithms for unencoding AG codes over any C_{ab} curve; in particular we also obtain quasi-linear complexity for unencoding Hermitian codes.

For the codes we study, encoding can be considered as follows: the entries of the message $\mathbf{m} \in \mathbb{F}^k$ are written as the coefficients of a bivariate polynomial $f_{\mathbf{m}} \in \mathbb{F}[X, Y]$ with bounded degree, and the codeword is then obtained by evaluating $f_{\mathbf{m}}$ at rational points of the C_{ab} curve (in some specific order). This is called a “multipoint evaluation” of $f_{\mathbf{m}}$. Similarly, for unencoding we are given a codeword $\mathbf{c} \in \mathbb{F}^n$ and we seek the unique polynomial $f \in \mathbb{F}[X, Y]$ whose monomial support satisfies certain constraints and such that the entries of \mathbf{c} are the evaluations of f at the chosen rational points of the C_{ab} curve. This is called “polynomial interpolation” of the entries of \mathbf{c} .

Our outset is to find algorithms for multipoint evaluation and interpolation of bivariate polynomials on any point set \mathcal{P} , where we at first do not use the fact that \mathcal{P} are rational points on a C_{ab} curve; we do this in Sections III-A and IV-A respectively. Under mild assumptions, our algorithms for these problems have quasi-linear complexity in the input size when \mathcal{P} is a “semi-grid”, i.e. if we let $\mathcal{Y}_\alpha = \{\beta \in \mathbb{F} \mid (\alpha, \beta) \in \mathcal{P}\}$ for $\alpha \in \mathbb{F}$, then each $|\mathcal{Y}_\alpha|$ is either 0 or equals some constant ν_Y independent of α , see Figure 2 and Definition II.5. This result may be of independent interest. We then apply these algorithms to the coding setting in Sections III-B and IV-B respectively. In Section V we more specifically study the performance for C_{ab} curves with special structure or sufficiently many points.

Contributions:

- We give quasi-linear time algorithms for bivariate multipoint evaluation and interpolation

when the point set is a semi-grid, under some simple conditions of the monomial support. See Remarks III.2 and IV.6.

- We give algorithms for encoding and for unencoding a one-point AG code over an arbitrary C_{ab} curve. Under very mild assumptions on the C_{ab} curve, these algorithms have complexity $\tilde{O}(n^{3/2})$ respectively $\tilde{O}(qn) \subset \tilde{O}(n^2)$. Our interpolation algorithm requires a polynomial amount of precomputation time. The encoding is not systematic. See Theorems III.4 and IV.12.
- We show that for codes whose evaluation points are semi-grids in a particular “maximal” way compared to the C_{ab} curve, both algorithms have quasi-linear complexity in the length of the code. This includes codes over the Hermitian curve and norm-trace curves. See Definition V.1, Propositions V.2 and V.5, and Corollaries V.6 to V.8.
- We show that the algorithms have improved complexity if the C_{ab} curve has sufficiently many rational points. For example, if the number of rational points is a constant fraction from the Hasse–Weil bound, then the encoding algorithm has complexity $\tilde{O}(n^{5/4})$, and the unencoding algorithm has complexity $\tilde{O}(n^{3/2})$, see Theorem V.11.

A. Related work on bivariate multipoint evaluation

As outlined above, multipoint evaluation (MPE) and interpolation of bivariate polynomials over given point sets is a computational problem tightly related to encoding and unencoding of AG codes over C_{ab} curves. In fact, any MPE algorithm for bivariate polynomials can immediately be applied for encoding. The situation is somewhat more complicated for interpolation, which we get back to. In this section we review the literature on these problems²

We begin by discussing the MPE problem. The input is a point set $\mathcal{P} \subset \mathbb{F}^2$ and $f \in \mathbb{F}[X, Y]$ with $\deg_X f = d_X$ and $\deg_Y f = d_Y$, and we seek $(f(P))_{\mathcal{P}}$. Let $n := |\mathcal{P}|$. As we will see in Section II-A, the main interest for the application of C_{ab} codes is when $d_X d_Y < n$ and $d_Y \ll d_X$. The former is a common assumption in the literature, but numerous papers assume $d_X \approx d_Y$ and such algorithms will often have poor performance in our case.

Spurred by the quasi-linear algorithm available in the univariate case (see Section II-C), the best we could hope for would be an algorithm of complexity $\tilde{O}(d_X d_Y + n)$, i.e. quasi-linear

²Some of the discussed algorithms apply to more variables than just 2, but we specialise the discussion to the bivariate case to ease comparison with our results.

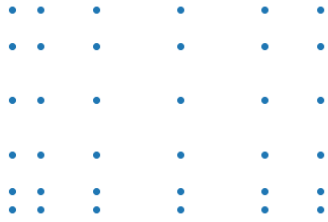


Figure 1: A grid.

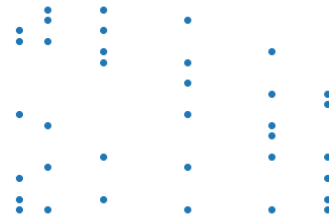


Figure 2: A semi-grid.

in the size of the input, but such a result is still not known in general. We will exemplify the complexities here for use in encoding Hermitian codes, i.e. C_{ab} codes over the Hermitian curve, see Section V-A1: in this case $n = q^3$ where we work over the field \mathbb{F}_{q^2} . We will consider the case where the dimension of the code is in the order of the length, for which we then have $\deg_X f \in \mathcal{O}(n^{2/3})$ and $\deg_Y f < q = n^{1/3}$.

The naive approach is to compute the evaluations of f one-by-one. Using Horner's rule, each such evaluation can be computed in $\mathcal{O}(d_X d_Y)$ time, for a total complexity of $\mathcal{O}(d_X d_Y n)$. For the Hermitian codes, the complexity would be $\mathcal{O}(n^2)$.

One of the first successes were Pan [30] with a quasi-linear algorithm for the case $\mathcal{P} = S_X \times S_Y$ for $S_X, S_Y \subseteq \mathbb{F}$, i.e. evaluation on a grid, see Figure 1. The algorithm works by applying univariate MPE in a “tensored” form. The algorithm can be directly applied for any \mathcal{P} by calling it on the smallest grid $\hat{\mathcal{P}}$ which contains all of \mathcal{P} and then throwing away the unneeded evaluations. If $|\hat{\mathcal{P}}| \gg n$ then the complexity of this approach will not be quasi-linear in the original input size: in the worst case $|\hat{\mathcal{P}}| \approx n^2$ so the complexity becomes $\tilde{\mathcal{O}}(d_X d_Y + n^2)$, which is quadratic in the input size when $d_X d_Y < n$. For Hermitian codes, then $\hat{\mathcal{P}} = \mathbb{F}_{q^2}^2$, hence Pan's algorithm would give complexity $\tilde{\mathcal{O}}(n^{4/3})$. Our MPE algorithm presented in Section III-A is a generalisation of Pan's algorithm which is quasi-linear on point sets with semi-grid structure, see Figure 2; therefore the performance of our algorithm is never worse than Pan's. Moreover, though few C_{ab} curves form a grid, we observe in Section V-A that certain nice families, including the Hermitian curve, form semi-grids, implying that our MPE algorithm has quasi-linear complexity for the point sets of these curves.

Nüsken and Ziegler [29] (NZ) reduced bivariate multipoint evaluation to a variant of *bivariate modular composition*: Write $\mathcal{P} = \{(\alpha_1, \beta_1), \dots, (\alpha_n, \beta_n)\}$ and assume all the α_i are distinct.

Compute $h(X) = \prod_{i=1}^n (X - \alpha_i)$ and $g \in \mathbb{F}[X]$ such that $g(\alpha_i) = \beta_i$ for $i = 1, \dots, n$; both of these can be computed in $\tilde{\mathcal{O}}(n)$ time. If we then compute $\rho(x) = f(x, g(x)) \bmod h(x) \in \mathbb{F}[X]$, we see that $\rho(\alpha_i) = f(\alpha_i, \beta_i)$ for each i , and hence we can compute the evaluations of f at the points \mathcal{P} by a univariate MPE of ρ at the $\alpha_1, \dots, \alpha_n$. The latter can be done in $\tilde{\mathcal{O}}(n)$ time, so all that remains is the computation of $\rho(X)$. Nüsken and Ziegler show how to do this in complexity roughly $\mathcal{O}(d_X d_Y^{1.635+\epsilon} + n d_Y^{0.635+\epsilon})$, where $\epsilon > 0$ can be chosen arbitrarily small, using fast rectangular matrix multiplication [19].

In general, and in the case of our interest, the X -coordinates of \mathcal{P} will not be distinct. In this case, the points can be “rotated” by going to an extension field K with $[K : \mathbb{F}] = 2$: choose any $\theta \in K \setminus \mathbb{F}$, and apply the map $(\alpha, \beta) \mapsto (\alpha + \theta\beta, \beta)$ to the points \mathcal{P} , and replace f by $\hat{f} := f(X - \theta Y, Y)$. We can now apply the NZ algorithm; that the operations will take place in K costs only a small constant factor compared to operations in \mathbb{F} since the extension degree is only 2. The main problem is that $\deg_Y \hat{f}$ is now generically $\max(d_X, d_Y)$, so assuming $d_Y < d_X$, the complexity of the NZ algorithm becomes roughly $\mathcal{O}((n + d_X^2) d_X^{0.635+\epsilon})$. For Hermitian codes, this yields $\mathcal{O}(n^{1.756+\epsilon})$.

In their celebrated paper [17], Kedlaya and Umans (KU) gave an algorithm for bivariate MPE with complexity $\mathcal{O}((n + d_X^2)^{1+\epsilon})$ bit operations for any $\epsilon > 0$, assuming $d_Y < d_X$. In outline, the algorithm works over prime fields by lifting the data to integers, then performing the MPE many times modulo many small primes, and then reassembling the result using the Chinese Remainder theorem. Over extension fields, some more steps are added for the lift to work. Note that the KU algorithm has quasi-linear complexity when $d_Y \approx d_X$. As mentioned, our main interest is $d_Y \ll d_X$. For our running example of the Hermitian code with dimension in the order of n , then applying the KU algorithm has complexity $\mathcal{O}(n^{4/3+\epsilon})$.

Let us turn to the interpolation problem. The input is now a point set $\mathcal{P} \subset \mathbb{F}^2$ and interpolation values $\mathcal{F} : \mathcal{P} \rightarrow \mathbb{F}$, and we seek $f \in \mathbb{F}[X, Y]$ such that $f(P) = \mathcal{F}(P)$ for each $P \in \mathcal{P}$. There are infinitely many such f , so to further restrict, or even make the output unique, one has to pose restrictions on the monomial support on the output f . We discuss the setting relevant to us in Section II-B: we essentially require that each monomial $x^i y^j$ of f satisfies $ai + jb \leq \hat{m}$ for a suitably chosen constant \hat{m} , where a and b are dictated by the C_{ab} curve.

Because of the many ways the monomial support could be restricted, there are not many interpolation algorithms in the literature that directly apply to the problem. If we let $\mathcal{G} \subset \mathbb{F}[X, Y]$ be the ideal of all polynomials which vanish at the points of \mathcal{P} , note that if f is the sought

solution, then the coset $f + \mathcal{G}$ is exactly the set of all polynomials which satisfy the interpolation conditions. One approach for our setting is therefore to use any interpolation algorithm to first find some $\hat{f} \in \mathbb{F}[X, Y]$ which satisfy the interpolation conditions but possibly has incorrect monomial support, and then reduce this modulo a Gröbner basis $G \subset \mathbb{F}[X, Y]$ of \mathcal{G} under an appropriate monomial order. Using the division algorithm of van der Hoeven [35], this reduction can be computed in quasi-linear time in the size of \hat{f} and G . We give the details in Section IV-A2.

With this observation, our interpolation problem could be solved as follows: first find the smallest grid $\hat{\mathcal{P}}$ which contains \mathcal{P} , and then use the algorithm outlined by Pan [30] to compute \hat{f} in quasi-linear complexity in $|\hat{\mathcal{P}}|$, and then reduce \hat{f} modulo G . As for MPE, this will have poor complexity when $|\hat{\mathcal{P}}| \gg n$. Our algorithm uses exactly this strategy, but where we generalise Pan's interpolation algorithm to one which $\hat{\mathcal{P}}$ can be chosen to be the smallest semi-grid containing \mathcal{P} . When the evaluation points of the C_{ab} code are a semi-grid in a certain maximal sense, we show that this algorithm gives us exactly enough control over the degrees that \hat{f} satisfies our constraints on the monomial support immediately, and hence we can skip the reduction by G . This is what gives us $\tilde{\mathcal{O}}(n)$ complexity for e.g. the Hermitian codes, see Section V-A.

The \hat{f} that our unencoding algorithm first outputs can be given a closed-form expression, see Lemma IV.1. This expression was used in a decoding algorithm for the special case of Hermitian codes in [23], and it was shown in [28] how to compute it fast; that approach can be seen as a special case of our algorithm.

A very different, and very flexible, interpolation algorithm is simply to solve the interpolation constraints as a system of linear constraints in the coefficients to the monomials in the monomial support. Solving the resulting $n \times n$ linear system using Gaussian elimination would cost $\mathcal{O}(n^\omega)$, where $\omega < 2.37286$ is the exponent of matrix multiplication [20]. We can do much better by observing that for the monomial support we require, the linear system would have low *displacement rank*, namely a , so we could use the algorithm for structured system solving by Bostan et al. [3], [4] for a cost of $\tilde{\mathcal{O}}(a^{\omega-1}n)$. For the Hermitian codes this yields a complexity of roughly $\tilde{\mathcal{O}}(n^{1.458})$. For general C_{ab} codes, this is our main contender, and we compare again in Sections IV-B and V-B.

II. PRELIMINARIES

A. Codes from C_{ab} curves

In this subsection, we discuss in some detail the family of AG codes for which we want to find fast encoders and unencoders. Note that these AG codes and the algebraic curves used to construct them were previously studied in [24], [25] and includes the results mentioned here. Also they occur as a special case of the codes and curves studied in [8], [14].

For a bivariate polynomial $H = \sum_{i,j} a_{ij} X^i Y^j \in \mathbb{F}[X, Y]$ with coefficients in a finite field \mathbb{F} , we define $\text{supp}(H) = \{X^i Y^j \mid a_{ij} \neq 0\}$.

Definition II.1. *Let a, b be nonnegative, coprime integers. We say that a bivariate polynomial $H \in \mathbb{F}[X, Y]$ is a C_{ab} polynomial if:*

- $X^b, Y^a \in \text{supp}(H)$,
- $X^i Y^j \in \text{supp}(H) \implies ai + bj \leq ab$,
- *The ideal $\langle H, \frac{\partial H}{\partial X}, \frac{\partial H}{\partial Y} \rangle \subseteq \mathbb{F}[X, Y]$ is equal to the unit ideal $\mathbb{F}[X, Y]$.*

Remark II.2. *In our algorithms the two variables X and Y are treated differently, which entails that complexities are not invariant under swapping of X and Y in H . We will commit to the arbitrary choice of “orienting” our algorithms such that their complexities depend explicitly only on a , which means that whenever the input is not assumed to have special structure which depends a and b , it is ofcourse sensible to permute X and Y such that $a < b$. In such cases we will sometimes assume w.l.o.g. that $a < b$.*

Define $\text{deg}_{a,b}$, to be the (a, b) -weighted degree of a bivariate polynomial. More concretely: $\text{deg}_{a,b}(X^i Y^j) = ai + bj$. The first two conditions imply that $H(X, Y) = \alpha X^b + \beta Y^a + G(X, Y)$, where $\alpha, \beta \in \mathbb{F} \setminus \{0\}$ and $\text{deg}_{a,b}(G(X, Y)) < ab$. In particular, the polynomial H is absolutely irreducible [14, Cor. 3.18]. The theory of Newton polygons, i.e., the convex hull of $\{(i, j) \mid X^i Y^j \in \text{supp}(H)\}$, can also be used to conclude this [9].

This implies that the a C_{ab} polynomial defines an algebraic curve. Following [24], the type of algebraic curves obtained in this way are called C_{ab} curves. As observed there, these curves, when viewed as projective curves, have exactly one point at infinity P_∞ , which, if singular, is a cusp. What this means can be explained in a very simple way using the language of function fields. A given C_{ab} polynomial H defines a C_{ab} curve, with function field $F = \mathbb{F}(x, y)$ obtained by extending the rational function field $\mathbb{F}(x)$ with a variable y satisfying $H(x, y) = 0$. Since H

is absolutely irreducible, \mathbb{F} is the full constant field of F . The statement that the point P_∞ , if it is a singularity, is a cusp, just means that the function x has exactly one place of F as a pole. With slight abuse of notation, we denote this place by P_∞ as well. The defining equation of a C_{ab} curve, directly implies that for any $i, j \in \mathbb{Z}$, the function $x^i y^j$ has pole order $\deg_{a,b}(x^i y^j) = ai + bj$ at P_∞ . In particular, x has pole order a and y has pole order b at P_∞ .

The *genus* of a function field is important for applications in coding theory, since it occurs in the Goppa bound on the minimum distance of AG codes. It is observed in [24] that the genus of the function field $F = \mathbb{F}(x, y)$ defined above equals $g = (a-1)(b-1)/2$. Indeed, this is implied by the third condition in Definition II.1, also see [2, Theorem 4.2]. We collect some facts in the following proposition. These results are contained in [24], expressed there in the language of algebraic curves.

Proposition II.3 ([24]). *Let $H \in \mathbb{F}[X, Y]$ be a C_{ab} polynomial and $F = \mathbb{F}(x, y)$ the corresponding function field. Then F has genus $g = \frac{1}{2}(a-1)(b-1)$. The place P_∞ is rational and a common pole of the functions x and y and in fact the only place which is a pole of either x or y . For any $i, j \in \mathbb{Z}$, the function $x^i y^j \in F$ has pole order $\deg_{a,b}(x^i y^j) = ai + bj$ at P_∞ .*

For a divisor D of the function field F , we denote by $\mathcal{L}(D)$ the Riemann–Roch space associated to D . The third condition in Definition II.1 implies that a C_{ab} curve cannot have singularities, apart from the possibly singular point at infinity. This has two important consequences. In the first place, all rational places of F distinct from P_∞ , can be identified with the points $(\alpha, \beta) \in \mathbb{F}^2$ satisfying $H(\alpha, \beta) = 0$. We will call these places the finite rational places of F . Throughout the paper we will, by a slight abuse of notation, use a finite place $P_{\alpha,\beta}$ and its corresponding rational point (α, β) interchangeably. A second consequence, as observed in [24], is that the bivariate polynomials $\mathbb{F}[x, y]$ are the *only* functions in F with poles only at P_∞ ; in other words, $\mathcal{L}(mP_\infty) = \{f \in \mathbb{F}[x, y] \mid \deg_{a,b}(f) \leq m\}$. Note that $\mathbb{F}[x, y]$ is *not* an ordinary bivariate polynomial ring since x and y satisfy $H(x, y) = 0$. As a result, any $f \in \mathbb{F}[x, y]$ can be uniquely written as a polynomial with y -degree at most $a-1$. We will call this the *standard form* of f . We are now ready to define C_{ab} codes.

Definition II.4. *Let H be a C_{ab} polynomial and F the corresponding function field. Further, let P_1, \dots, P_n be distinct, finite rational places of F and let m be a nonnegative integer. Then the*

C_{ab} code of order m is defined to be:

$$\mathcal{C}_H(\mathcal{P}, m) := \{(\text{ev}_{\mathcal{P}}(f) \mid f \in \mathcal{L}(mP_{\infty})\} \subseteq \mathbb{F}^n, \text{ where}$$

$$\text{ev}_{\mathcal{P}}(f) = (f(P_1), \dots, f(P_n)) .$$

In the standard notation for AG codes used for example in [32], the code $\mathcal{C}_H(\mathcal{P}, m)$ is equal to the code $C_{\mathcal{L}}(D, mP_{\infty})$, with $D = P_1 + \dots + P_n$. Since the divisor mP_{∞} is a multiple of a single place, the codes $\mathcal{C}_H(\mathcal{P}, m)$ are examples of what are known as one-point AG codes. Using for example [32, Theorem 2.2.2], we obtain that $\mathcal{C}_H(\mathcal{P}, m)$ is an $[n, k, d]$ linear code, where $k = \dim(\mathcal{L}(mP_{\infty})) - \dim(\mathcal{L}(mP_{\infty} - D))$ and $d \geq n - m$. In particular, $k = n$ if $m > n + 2g - 2$. Therefore we will from now on always assume that $m \leq n + 2g - 1$. If $m < n$, then $k = \dim(\mathcal{L}(mP_{\infty})) \geq m + 1 - g$ and if additionally $2g - 2 < m$, then $k = m + 1 - g$. The precise minimum distance of a C_{ab} code is in general not known from just the defining data. Lastly, we also note the obvious bound $n \leq q^2$, where $q = |\mathbb{F}|$, due to the identification of rational places with points in \mathbb{F}^2 .

When comparing algorithms pertaining to AG codes, as well as many other types of codes, it is customary to assume that the dimension k grows proportional to n , denoted $k \in \Theta(n)$, i.e. that the rate goes to some constant as $n \rightarrow \infty$. For a family of C_{ab} codes, this implies that $m \in \Theta(n)$. This in turn means that any message polynomial $f \in \mathcal{L}(mP_{\infty})$ in standard form satisfies $\deg_Y f < a$ and $\deg_X f < m/a \in \Theta(n/a)$. A C_{ab} code is considered good if n is relatively large compared to the field size q and the genus g is small, as measured e.g. against the Hasse–Weil bound, see Section V-B. For such codes, then $g \ll n$, so $ab \ll n$. This means that in the cases of most interest to us, the message polynomials tend to have very different X and Y degrees.

B. The evaluation-encoding map

An encoding for a linear code such as $\mathcal{C}_H(\mathcal{P}, m)$ is a linear, bijective map $\phi : \mathbb{F}^k \rightarrow \mathcal{C}_H(\mathcal{P}, m) \subseteq \mathbb{F}^n$. Computing the image of ϕ for some $\mathbf{m} \in \mathbb{F}^k$ is called “encoding” \mathbf{m} . The process of computing the inverse, i.e. given a codeword $\mathbf{c} \in \mathcal{C}_H(\mathcal{P}, m)$ recover the message $\mathbf{m} := \phi^{-1}(\mathbf{c})$, is often unnamed in the literature. For lack of a better term (and since “decoding” is reserved for error-correction), we will call it “unencoding”.

In light of Definition II.4, we can factor ϕ as $\phi = \text{ev}_{\mathcal{P}} \circ \varphi$, where $\varphi : \mathbb{F}^k \rightarrow \mathcal{L}(mP_{\infty})$ is linear and injective. If we choose φ sufficiently simple and such that it outputs elements of $\mathbb{F}[x, y]$ in

standard form, the computational task of applying ϕ reduces to computing $\text{ev}_{\mathcal{P}}$, i.e. multipoint evaluation of bivariate polynomials of (a, b) -weighted degree at most m . A natural basis for $\mathcal{L}(mP_{\infty})$ is

$$B = \{x^i y^j \mid \deg_{a,b}(x^i y^j) \leq m \wedge j \leq a - 1\}. \quad (\text{II.1})$$

If $k = \dim(\mathcal{L}(mP_{\infty}))$, then $|B| = k$, and we therefore choose φ as taking the elements of a message \mathbf{m} as the coefficients to the monomials of this basis in some specified order. Then applying φ takes no field operations at all.

If $k < \dim(\mathcal{L}(mP_{\infty}))$ then $|B| > k$, and this may happen when $m \geq n$. We should then choose a subset $\hat{B} \subset B$ of k monomials such that the vectors $\{\text{ev}_{\mathcal{P}}(x^i y^j)\}_{x^i y^j \in \hat{B}}$ are linearly independent. For our encoding algorithms, the choice of \hat{B} will not matter. However, for unencoding, we will assume that this choice has been made so that the monomials in \hat{B} are, when sorted according to their (a, b) -weighted degrees, lexicographically minimal. Put another way, a monomial $x^i y^j \in B$ is *not* in \hat{B} exactly when there is a polynomial $g \in \mathbb{F}[x, y]$ whose monomial of maximal (a, b) -weighted degree is $x^i y^j$ and such that $g \in \ker(\text{ev}_{\mathcal{P}})$. Such monomials $x^i y^j$ are what we will call “reducible” monomials in Section IV-A2.

\hat{B} is easy to precompute: start with $\hat{B} = \emptyset$, and go through the monomials of B in order of increasing $\deg_{a,b}$. For each such $x^u y^v$ if $\text{ev}_{\mathcal{P}}(x^u y^v)$ is linearly independent from $\{\text{ev}_{\mathcal{P}}(x^i y^j)\}_{x^i y^j \in \hat{B}}$, then add to \hat{B} .

C. Notation and computational tools

For any point set $\mathcal{P} \subseteq \mathbb{F}^2$ we define $\mathcal{X}(\mathcal{P}) := \{\alpha \in \mathbb{F} \mid \exists \beta \in \mathbb{F} \text{ s. t. } (\alpha, \beta) \in \mathcal{P}\}$, i.e. the set of all X -coordinates that occur in \mathcal{P} . We write $n_X(\mathcal{P}) := |\mathcal{X}(\mathcal{P})|$ for the number of distinct X -coordinates. Similarly, for any $\alpha \in \mathbb{F}$ we define $\mathcal{Y}_{\alpha}(\mathcal{P}) := \{\beta \in \mathbb{F} \mid (\alpha, \beta) \in \mathcal{P}\}$, i.e. the set of Y -coordinates that occur for a given X -coordinate α , and we let

$$\nu_Y(\mathcal{P}) := \max_{\alpha \in \mathcal{P}} |\mathcal{Y}_{\alpha}(\mathcal{P})|.$$

In discussions where it is clear from the context which point set \mathcal{P} we are referring to, we may simply write $\mathcal{X}, \mathcal{Y}, n_X, \nu_Y$. Note that if \mathcal{P} are a subset of the rational points of a C_{ab} curve with polynomial $H(X, Y)$, then $\nu_Y(\mathcal{P}) \leq a =: \deg_Y(H)$ since for any given value of α , there are at most a solutions to the resulting equation in $H(\alpha, Y)$.

Definition II.5. A point set $\mathcal{P} \subset \mathbb{F}^2$ is a semi-grid if $|\mathcal{Y}_{\alpha}(\mathcal{P})| \in \{0, \nu_Y(\mathcal{P})\}$ for each $\alpha \in \mathbb{F}$.

As outlined in Section II-A, we distinguish between the bivariate polynomial ring $\mathbb{F}[X, Y]$ and the subset of functions in the C_{ab} function field spanned by x and y , denoted $\mathbb{F}[x, y]$. However, there is a natural inclusion map of functions $f \in \mathbb{F}[x, y]$ in standard form into a polynomial $f(X, Y) \in \mathbb{F}[X, Y]$ of Y -degree less than a . In discussions and algorithms, we sometimes abuse notation by more or less explicitly making use of this inclusion map.

For ease of notation, our algorithms use *lookup tables*, also known as dictionaries or associative arrays. This is just a map $\mathcal{A} \rightarrow \mathcal{B}$ between a finite set \mathcal{A} and a set \mathcal{B} but where all the mappings have already been computed and stored, and hence quickly be retrieved. We use the notation $\mathcal{F} \in \mathcal{B}^{\mathcal{A}}$ to mean a lookup table from \mathcal{A} to \mathcal{B} . For $a \in \mathcal{A}$, we write $\mathcal{F}[a] \in \mathcal{B}$ for the mapped value stored in \mathcal{F} . Note that this is for notational convenience only: all our uses of lookup tables could be replaced by explicit indexing in memory arrays, and so we will assume that retrieving or inserting values in tables costs $O(1)$.

Our complexity analyses count basic arithmetic operations in the field \mathbb{F} on an algebraic RAM model. We denote by $M(n)$ the cost of multiplying two univariate polynomials in $\mathbb{F}[X]$ of degree at most n . We can take $M(n) \in O(n \log n \log \log n)$ [5], or the slightly improved algorithm of [13] with cost $M(n) \in O(n \log n 8^{\log^* n})$, both of which are in $\tilde{O}(n)$. For precision, our theorems state complexities in big- \mathcal{O} including all log-factors, and we then relax the expressions to soft- \mathcal{O} for overview.

Our algorithms take advantage of two fundamental computational tools for univariate polynomials: *fast multipoint evaluation* and *fast interpolation*. These are classical results, see e.g. [36, Corollaries 10.8 and 10.12].

Proposition II.6. *There exists an algorithm UnivariateMPE which inputs a univariate polynomial $h \in \mathbb{F}[Z]$ and evaluation points $\mathcal{S} \subseteq \mathbb{F}$, and outputs a table $\mathcal{F} : \mathbb{F}^{\mathcal{S}}$ such that $\mathcal{F}[\alpha] = h(\alpha)$ for every $\alpha \in \mathcal{S}$. It has complexity*

$$\mathcal{O}(M(\deg h + |\mathcal{S}|) \log(\deg h + |\mathcal{S}|)) \subset \tilde{\mathcal{O}}(\deg h + |\mathcal{S}|)$$

operations in \mathbb{F} .

Proposition II.7. *There exists an algorithm UnivariateInterp which inputs evaluation points $\mathcal{S} \subseteq \mathbb{F}$ and evaluation values $\mathcal{F} \in \mathbb{F}^{\mathcal{S}}$, and outputs the unique $f \in \mathbb{F}[Z]$ such that $\deg f < k$ and $f(\alpha) = \mathcal{F}[\alpha]$ for each $\alpha \in \mathcal{S}$, where $k = |\mathcal{S}|$. It has complexity $\mathcal{O}(M(k) \log(k)) \subset \tilde{\mathcal{O}}(k)$ operations in \mathbb{F} .*

III. A FAST ENCODING ALGORITHM

Let us now consider an algorithm for computing the encoding map for C_{ab} codes. We are given a message vector $f \in \mathcal{L}(mP_\infty) \subset \mathbb{F}[X, Y]/\langle H \rangle$ and n rational places P_1, \dots, P_n of F ; we wish to compute $f(P_1), \dots, f(P_n)$. We translate this problem into bivariate polynomial multipoint-evaluation by lifting f to a polynomial in $\mathbb{F}[X, Y]$ in standard form, and identifying each P_i with a pair $(\alpha_i, \beta_i) \in \mathbb{F}^2$ such that $H(\alpha_i, \beta_i) = 0$ for $i = 1, \dots, n$. In the following subsection we will focus on the evaluation problem at hand, while in Section III-B we will apply the results to encoding of codes over C_{ab} curves.

A. Multipoint-Evaluation of Bivariate Polynomials

Let us for now forget that we originally came from the setting of codes. Suppose that we are given a set $\mathcal{P} \subseteq \mathbb{F}^2$ of points with $|\mathcal{P}| = n$ and a bivariate polynomial $f \in \mathbb{F}[X, Y]$ with $\deg_X f = d_X$ and $\deg_Y f = d_Y$. We can evaluate f at each point individually, which will cost us $\mathcal{O}(d_X d_Y)$ operations for each point. This naive approach will have complexity $\mathcal{O}(n d_X d_Y)$.

We will generalise Pan's multipoint evaluation algorithm [30] (see Section I-A), and show that it performs well on point sets \mathcal{P} where most $|\mathcal{Y}_\alpha(\mathcal{P})|$ are roughly the same size for each $\alpha \in \mathcal{X}(\mathcal{P})$.

The idea of the algorithm is the following: we write

$$f(X, Y) = f_0(X) + f_1(X)Y + \dots + f_{d_Y}(X)Y^{d_Y} \quad , f_i \in \mathbb{F}[X] \quad ,$$

and then proceed by $d_Y + 1$ univariate multipoint evaluations of the polynomials $f_i(X)$, $i = 0, \dots, d_Y$, each evaluated on the values $\mathcal{X}(\mathcal{P})$. For each $\alpha \in \mathcal{X}(\mathcal{P})$, we can therefore construct a univariate polynomial in Y without further computations:

$$g_\alpha(Y) = f_0(\alpha) + f_1(\alpha)Y + \dots + f_{d_Y}(\alpha)Y^{d_Y} \quad , f_i \in \mathbb{F}[X] \quad .$$

Again using univariate multipoint evaluation, we obtain $g_\alpha(\beta) = f(\alpha, \beta)$ for each $\beta \in \mathcal{Y}_\alpha(\mathcal{P})$.

For algorithm listing see Algorithm 1.

Replace := with arrow in algorithm listings.

Theorem III.1. *Algorithm 1 is correct. It has complexity*

$\mathcal{O}(d_Y M(d_X + n_X) \log(d_X + n_X) + n_X M(d_Y + \nu_Y) \log(d_Y + \nu_Y)) \subset \tilde{\mathcal{O}}(d_Y d_X + n_X(d_Y + \nu_Y))$
operations in \mathbb{F} , where $d_X = \deg_X(f)$, $n_X = n_X(\mathcal{P})$, and $\nu_Y = \nu_Y(\mathcal{P})$.

Algorithm 1: BivariateMPE: Bivariate multipoint evaluation

Input:

Bivariate polynomial $f = f_0(X) + f_1(X)Y + \dots + f_{d_Y}(X)Y^{d_Y} \in \mathbb{F}[X, Y]$.

Evaluation points $\mathcal{P} \subseteq \mathbb{F}^2$.

Output:

Evaluation values $\mathcal{F} = (f(\alpha, \beta))_{(\alpha, \beta) \in \mathcal{P}} \in \mathbb{F}^{\mathcal{P}}$.

```

1  $\mathcal{X} := \mathcal{X}(\mathcal{P})$ 
2  $\mathcal{Y}_\alpha := \mathcal{Y}_\alpha(\mathcal{P})$ 
3 for  $i = 1, \dots, d_Y$  do  $\mathcal{F}_i := \text{UnivariateMPE}(f_i, \mathcal{X}) \in \mathbb{F}^{\mathcal{X}}$ 
4 foreach  $\alpha \in \mathcal{X}$  do
5    $g_\alpha := \sum_{i=0}^{d_Y} \mathcal{F}_i[\alpha]Y^i \in \mathbb{F}[Y]$ 
6    $\mathcal{G}_\alpha := \text{UnivariateMPE}(g_\alpha, \mathcal{Y}_\alpha) \in \mathbb{F}^{\mathcal{Y}_\alpha}$ 
7 return  $\mathcal{F} := (\mathcal{G}_\alpha[\beta])_{(\alpha, \beta) \in \mathcal{P}} \in \mathbb{F}^{\mathcal{P}}$ 

```

Proof. Correctness follows from the fact that

$$\mathcal{F}[\alpha, \beta] = \mathcal{G}_\alpha[\beta] = g_\alpha(\beta) = \sum_{i=0}^{d_Y} \mathcal{F}_i[\alpha]\beta^i = \sum_{i=0}^{d_Y} f_i(\alpha)\beta^i = f(\alpha, \beta) .$$

For the complexity, Lines 1 and 2 both have cost $\mathcal{O}(n)$. Proposition II.6 implies that computing \mathcal{F}_i costs

$$\mathcal{O}(\mathbb{M}(\deg f_i + n_X) \log(\deg f_i + n_X)) \text{ for } i = 1, \dots, d_Y ,$$

thus the total cost for Line 3 becomes

$$\mathcal{O}(d_Y \mathbb{M}(d_X + n_X) \log(d_X + n_X)).$$

Line 5 costs no operations in \mathbb{F} . From Proposition II.6 it follows that computing each \mathcal{G}_α costs

$$\mathcal{O}(\mathbb{M}(d_Y + |\mathcal{Y}_\alpha|) \log(d_Y + |\mathcal{Y}_\alpha|)) \text{ for } \alpha \in \mathcal{X},$$

thus the total cost of Line 6 becomes

$$\mathcal{O}(n_X \mathbb{M}(d_Y + \nu_Y) \log(d_Y + \nu_Y)) .$$

Line 7 costs no operations in \mathbb{F} , and so the total cost of computing \mathcal{F} becomes as in the theorem. □

Remark III.2. If $\mathcal{P} \in \mathbb{F}^2$ is a semi-grid, and $f \in \mathbb{F}[X, Y]$ is a dense polynomial satisfying either $n_X \in \mathcal{O}(\deg_X f)$ or $\deg_Y(f) \in \mathcal{O}(\nu_Y)$, then Algorithm 1 has quasi-linear complexity in the input size $|\mathcal{P}| + \deg_X f \deg_Y f$.

Remark III.3. Even if we use classical polynomial multiplication, with $M(n) = \mathcal{O}(n^2)$, and if we assume $d_X \in \Theta(n_X)$ and $d_Y \in \Theta(\nu_Y)$, then the cost of Algorithm 1 is $\tilde{\mathcal{O}}(d_X d_Y n_X + d_Y n)$ which is for most point sets much better than the naive approach of point-by-point evaluation costing $\mathcal{O}(d_X d_Y n)$.

B. Fast encoding

It is straightforward to use Algorithm 1 to achieve fast encoding; details can be found in Algorithm 2.

Algorithm 2: Encode

Input: A C_{ab} code $\mathcal{C}_H(\mathcal{P}, m) \subseteq \mathbb{F}^n$ with $\mathcal{P} = \{P_1, \dots, P_n\}$ being finite rational places and dimension k . Message $\mathbf{m} \in \mathbb{F}^k$.

Output: Codeword $\mathbf{c} = \phi(\mathbf{m})$, where $\phi : \mathbb{F}^k \rightarrow \mathcal{C}_H(\mathcal{P}, m)$ is the encoding map defined Section II-B.

- 1 $f := \varphi(\mathbf{m}) \in \mathbb{F}[x, y]$ in standard form, where φ is as in Section II-B
 - 2 $\mathcal{F} := \text{BivariateMPE}(f, \mathcal{P}) \in \mathbb{F}^{\mathcal{P}}$, where f is lifted to $\mathbb{F}[X, Y]$
 - 3 **return** $(\mathcal{F}[\hat{P}_1], \dots, \mathcal{F}[\hat{P}_n]) \in \mathbb{F}^n$
-

Theorem III.4. Algorithm 2 is correct. It uses at most $\mathcal{O}(M(m + an_X) \log(m + an_X)) \subset \tilde{\mathcal{O}}(m + an_X)$ operations in \mathbb{F} , where $n_X = n_X(\mathcal{P})$.

Proof. Correctness follows trivially from Theorem III.1.

For complexity let $d_X = \deg_X(f)$, $d_Y = \deg_Y(f)$ and $\nu_Y = \nu_Y(\mathcal{P})$. Since f is in standard form we have that $d_Y < a$. Furthermore, since $f \in \mathcal{L}(mP_\infty)$ we know that $ad_X + bd_Y \leq m$. It follows that $d_X d_Y < d_X a \leq m$. Since for each value of X in $\mathcal{X}(\mathcal{P})$, there can be at most a solutions in Y to the C_{ab} curve equation, we know $\nu_Y < a$. It follows from Theorem III.1 that

the cost of Line 2 is

$$\begin{aligned} & \mathcal{O}(d_Y \mathbf{M}(d_X + n_X) \log(d_X + n_X) + n_X \mathbf{M}(d_Y + \nu_Y) \log(d_Y + \nu_Y)) \\ & \subset \mathcal{O}(\mathbf{M}(m + an_X) \log(m + an_X)) . \end{aligned}$$

□

As can be seen, the complexity of this algorithm depends on parameters of the C_{ab} curve compared to the code length as well as the layout of the evaluation points: more specifically on how an_X compares with the code length n . In Section V-A we will revisit the complexity for codes over C_{ab} curves that lie on semi-grids as well as C_{ab} curves which have many points. In the worst case, the following corollary bounds the complexity in terms of the length of the code under very mild assumptions on the C_{ab} curve. Note that this cost is still much better than encoding using a matrix-vector product in $\mathcal{O}(n^2)$ time.

Corollary III.5. *In the context of Algorithm 2, let q be the cardinality of \mathbb{F} and assume $n \geq q$. Assume further that the genus g of the C_{ab} curve satisfies $g \leq n$. Then the complexity of Algorithm 2 is $\tilde{\mathcal{O}}(q\sqrt{n}) \subset \tilde{\mathcal{O}}(n^{3/2})$.*

Proof. There can at most be q different X -coordinates in \mathcal{P} , so $n_X \leq q$. Assuming w.l.o.g that $a < b$ we get $n \geq g = \frac{1}{2}(a-1)(b-1) \geq \frac{1}{2}(a-1)^2$, and hence $a \leq \sqrt{2n} + 1 \in \mathcal{O}(\sqrt{n})$. Lastly, $m \leq n + 2g - 1 \in \mathcal{O}(n)$. The result follows from Theorem III.4. □

IV. A FAST UNENCODING ALGORITHM

We now consider the problem of *unencoding*: we are given a codeword $\mathbf{c} \in \mathcal{C}_H(\mathcal{P}, m)$ and we wish to find the message $\mathbf{m} = \phi^{-1}(\mathbf{c}) \in \mathbb{F}^k$, where ϕ is the encoding map defined in Section II-B. Following the discussion there, we factor ϕ as $\phi = \text{ev}_{\mathcal{P}} \circ \varphi$, where $\varphi : \mathbb{F}^k \mapsto \mathcal{L}(mP_{\infty})$ which is a linear map that sends unit vectors to monomials in

$$\hat{B} \subseteq B := \{x^i y^j \mid \deg_{a,b}(x^i y^j) \leq m \wedge j \leq a-1\} .$$

Recall that the evaluation map is injective on $\mathcal{L}(mP_{\infty})$ whenever $m < n$, so in this case $\hat{B} = B$. Otherwise, \hat{B} is chosen such that if $g \in \ker(\text{ev}_{\mathcal{P}})$, then the monomial of maximal (a, b) -weighted degree of g is *not* in \hat{B} .

Our strategy will be to first use an efficient recursive strategy to find an $\hat{f} \in \mathbb{F}[X, Y]$ such that $\hat{f}(x(P_i), y(P_i)) = c_i$ for $i = 1, \dots, n$ while $\deg_Y \hat{f} < a$. In general, $\hat{f}(x, y) \neq f$ since

it will have incorrect monomial support. However, $f(X, Y) \in \hat{f} + \mathcal{G}$, where \mathcal{G} is the ideal of polynomials in $\mathbb{F}[X, Y]$ vanishing at the points P_1, \dots, P_n . By the choice of \hat{B} , we will see that we can recover $f(X, Y)$ by reducing \hat{f} modulo a suitable Gröbner basis G of \mathcal{G} using a fast multivariate division algorithm.

We will see in Section V-A that if the rational points of the C_{ab} curve form a semi-grid, e.g. the Hermitian curve, then the second step can be omitted because $\hat{f} = f$ holds whenever $m < n$.

A. Bivariate polynomial interpolation

1) *Finding a structured interpolation polynomial:* Suppose that we are given a set of n points $\mathcal{P} \subseteq \mathbb{F}^2$ and a corresponding collection of values $\mathcal{F} \in \mathbb{F}^{\mathcal{P}}$. The *interpolation problem* consists of finding a polynomial $f \in \mathbb{F}[X, Y]$ such that $f(\alpha, \beta) = \mathcal{F}[\alpha, \beta]$ for all $(\alpha, \beta) \in \mathcal{P}$. Since there are many such polynomials, one usually imposes constraints on the set of monomials $X^i Y^j$ that may appear with non-zero coefficient in f . Given the points \mathcal{P} and the allowed monomial support, it is not immediately obvious whether all possible interpolation constraints given by such \mathcal{F} can be satisfied. This depends on whether we can select a subset \mathcal{M} of n monomials $X^i Y^j$ satisfying the constraints and such that the corresponding vectors $\mathbf{v}_{i,j} := (\alpha^i \beta^j)_{(\alpha, \beta) \in \mathcal{P}} \in \mathbb{F}^{\mathcal{P}}$ are linearly independent over \mathbb{F} .

Whenever this is the case, we could therefore interpolate any values \mathcal{F} by linear algebra in $O(n^2)$ time using some precomputation: fix some order on \mathcal{P} and on the subset of monomials \mathcal{M} , allowing us to represent \mathcal{F} as a vector $\mathbf{y} \in \mathbb{F}^n$ as well as ordering the vectors $\mathbf{v}_{\alpha, \beta}$ as rows of a matrix $V \in \mathbb{F}^{n \times n}$. Then f as a vector of monomials becomes simply the vector–matrix product $\mathbf{y}V^{-1}$, once we have computed and stored V^{-1} .

To obtain a faster algorithm, we will use the following explicit equation which finds an interpolating function $\hat{f} \in \mathbb{F}[X, Y]$, but where we have much less control over the monomial support of the polynomial:

Lemma IV.1. *Given a point set $\mathcal{P} \in \mathbb{F}^2$ and interpolation values $\mathcal{F} \in \mathbb{F}^{\mathcal{P}}$, then $\hat{f} \in \mathbb{F}[X, Y]$ given by*

$$\hat{f} = \sum_{\alpha \in \mathcal{X}} \prod_{\alpha' \in \mathcal{X} \setminus \{\alpha\}} \frac{X - \alpha'}{\alpha - \alpha'} \sum_{\beta \in \mathcal{Y}_\alpha} \mathcal{F}[\alpha, \beta] \prod_{\beta' \in \mathcal{Y}_\alpha \setminus \{\beta\}} \frac{Y - \beta'}{\beta - \beta'}, \quad (\text{IV.1})$$

satisfies $f(\alpha, \beta) = \mathcal{F}[\alpha, \beta]$ for all $(\alpha, \beta) \in \mathcal{P}$.

Proof. Let $(\alpha, \beta) \in \mathcal{P}$. Then the only nonzero term in the first sum is the one corresponding to α , while the only nonzero term in the second sum is the one corresponding to β , thus

$$\hat{f}(\alpha, \beta) = \prod_{\alpha' \in \mathcal{X} \setminus \{\alpha\}} \frac{\alpha - \alpha'}{\alpha - \alpha'} \mathcal{F}[\alpha, \beta] \prod_{\beta' \in \mathcal{Y}_\alpha \setminus \{\beta\}} \frac{\beta - \beta'}{\beta - \beta'} = \mathcal{F}[\alpha, \beta].$$

□

Our strategy to compute \hat{f} in an efficient manner can be viewed in the following way: we start by computing the polynomials $\hat{f}_\alpha := \hat{f}(\alpha, Y) \in \mathbb{F}[Y]$ for every $\alpha \in \mathcal{X}$ using univariate interpolation. We then reinterpret our interpolation problem as being univariate over $(\mathbb{F}[Y])[X]$, i.e. we seek $\hat{f}(X)$ having coefficients in $\mathbb{F}[Y]$ and such that $\hat{f}(\alpha) = \hat{f}_\alpha$. However, to be more clear, and for a slightly better complexity (on the level of logarithms), we make this latter interpolation explicit.

Before we put these steps together to compute \hat{f} in Algorithm 4, we therefore first consider the following sub-problem: Given any subset $\mathcal{S} \in \mathbb{F}$ and a table of univariate polynomials $\mathcal{V} \in \mathbb{F}[Y]^\mathcal{S}$ indexed by \mathcal{S} , compute the following bivariate polynomial:

$$h(X, Y) = \sum_{\alpha \in \mathcal{S}} \mathcal{V}[\alpha] \prod_{\alpha' \in \mathcal{S} \setminus \{\alpha\}} (X - \alpha') \in \mathbb{F}[X, Y]. \quad (\text{IV.2})$$

For the $(\mathbb{F}[Y])[X]$ interpolation, we will follow an approach of univariate interpolation closely mimicking that of [36, Chapter 10.2]. Firstly, we arrange the X -coordinates of the interpolation points in a balanced tree:

Definition IV.2. Let $\mathcal{S} \subseteq \mathbb{F}$. A *balanced partition tree* of \mathcal{S} is a binary tree which has subsets of \mathcal{S} as nodes, and satisfies the following:

- 1) \mathcal{S} is the root node.
- 2) A leaf node is a singleton set $\{\alpha\} \subseteq \mathcal{S}$.
- 3) An internal node \mathcal{N} is the disjoint union of its two children $\mathcal{N}_1, \mathcal{N}_2$ and they satisfy $||\mathcal{N}_1| - |\mathcal{N}_2|| \leq 1$.

If \mathcal{T} is a balanced partition tree and $\mathcal{N} \subseteq \mathcal{S}$, we will write $\mathcal{N} \in \mathcal{T}$ if \mathcal{N} is a node of \mathcal{T} , and denote by $\mathcal{T}[\mathcal{N}]$ the set of its two child nodes.

Lemma IV.3 (Lemma 10.4 [36]). *There exists an algorithm TreeVanish_X which inputs a balanced partition tree \mathcal{T} of some $\mathcal{S} \subseteq \mathbb{F}$ and outputs the lookup table*

$$\text{TreeVanish}_X(\mathcal{T}) := \left(\prod_{\alpha \in \mathcal{N}} (X - \alpha) \right)_{\mathcal{N} \in \mathcal{T}} \in \mathbb{F}[X]^{\mathcal{T}}.$$

The algorithm uses at most $\mathcal{O}(M(k) \log(k)) \subset \tilde{\mathcal{O}}(k)$ operations in \mathbb{F} .

Algorithm 3: Combine: $\mathbb{F}[Y]$ -linear combination of vanishing polynomials

Input: Points $\mathcal{S} \subseteq \mathbb{F}$, non-empty.

Lookup table $\mathcal{V} \in \mathbb{F}[Y]^{\mathcal{S}}$.

A balanced partition tree \mathcal{T} with $\mathcal{S} \in \mathcal{T}$.

$\mathcal{U} = \text{TreeVanish}_X(\mathcal{T}) \in \mathbb{F}[X]^{\mathcal{T}}$.

Output: $h(X, Y) \in \mathbb{F}[X, Y]$ as in (IV.2).

```

1 if  $\mathcal{S} = \{\alpha\}$  then
2   | return  $\mathcal{V}[\alpha] \in \mathbb{F}[Y]$ 
3 else
4   |  $\{\mathcal{S}_1, \mathcal{S}_2\} := \mathcal{T}[\mathcal{S}]$ 
5   | for  $k = 1, 2$  do
6     |  $\mathcal{V}_k := (\mathcal{V}[\alpha])_{\alpha \in \mathcal{S}_k} \in \mathbb{F}[Y]^{\mathcal{S}_k}$ 
7     |  $\hat{f}_k := \text{Combine}(\mathcal{S}_k, \mathcal{V}_k, \mathcal{T}, \mathcal{U}) \in \mathbb{F}[X, Y]$ 
8   | return  $\hat{f}_1 \mathcal{U}[\mathcal{S}_2] + \hat{f}_2 \mathcal{U}[\mathcal{S}_1] \in \mathbb{F}[X, Y]$ 

```

With these tools in hand, Algorithm 3 is an algorithm for computing $h(X, Y)$ as in (IV.2).

Theorem IV.4. *Algorithm 3 is correct. If $\deg_Y \mathcal{V}[\alpha] < d$ for all $\alpha \in \mathcal{S}$, then the algorithm has complexity $\mathcal{O}(dM(k) \log(k)) \subset \tilde{\mathcal{O}}(dk)$ operations in \mathbb{F} , where $k = |\mathcal{S}|$.*

Proof. We prove correctness by induction on $|\mathcal{S}|$. The base case of $\mathcal{S} = \{\alpha\}$ is trivial. For the induction step, the algorithm proceeds into the else-branch and we note that $|\mathcal{S}_1|, |\mathcal{S}_2| < |\mathcal{S}|$ and so the induction hypothesis applies to \hat{f}_k for $k = 1, 2$:

$$\hat{f}_k = \sum_{\alpha \in \mathcal{S}_k} \mathcal{V}_k[\alpha] \prod_{\alpha' \in \mathcal{S}_k \setminus \{\alpha\}} (X - \alpha').$$

Note that $\mathcal{V}_k[\alpha] = \mathcal{V}[\alpha]$ and $\mathcal{S}_1 \cup \mathcal{S}_2 = \mathcal{S}$, so we conclude that the polynomial returned by the algorithm is

$$\begin{aligned} & \hat{f}_1 \mathcal{U}[\mathcal{S}_2] + \hat{f}_2 \mathcal{U}[\mathcal{S}_1] \\ &= \sum_{\alpha \in \mathcal{S}_1} \mathcal{V}[\alpha] \prod_{\alpha' \in \mathcal{S} \setminus \{\alpha\}} (X - \alpha') + \sum_{\alpha \in \mathcal{S}_2} \mathcal{V}[\alpha] \prod_{\alpha' \in \mathcal{S} \setminus \{\alpha\}} (X - \alpha') \\ &= \sum_{\alpha \in \mathcal{S}} \mathcal{V}[\alpha] \prod_{\alpha' \in \mathcal{S} \setminus \{\alpha\}} (X - \alpha') . \end{aligned}$$

For complexity let $T(k)$ denote the cost of Combine for $|\mathcal{S}| = k$. At a recursive step we solve two subproblems of size $k/2$; and we compute the expression $\hat{f}_1 \mathcal{U}[\mathcal{S}_2] + \hat{f}_2 \mathcal{U}[\mathcal{S}_1]$. Since $\mathcal{U}[\mathcal{S}_k] \in \mathbb{F}[X]$ then each of the two products can be carried out using $\deg_Y \hat{f}_k + 1 \leq d$ multiplications in $\mathbb{F}[X]_{\leq k}$ and hence cost $\mathcal{O}(dM(k))$ each. The addition $\hat{f}_1 \mathcal{U}[\mathcal{S}_2] + \hat{f}_2 \mathcal{U}[\mathcal{S}_1]$ costs a further $\mathcal{O}(dk)$. In total, we get the following recurrence relation:

$$T(k) = 2T(k/2) + \mathcal{O}(dM(k)) .$$

This has the solution $T(k) = \mathcal{O}(dM(k) \log(k)) + \mathcal{O}(k)T(1)$. $T(1)$ is the base case of the algorithm, and costs $\mathcal{O}(d)$. □

We are now in position to assemble the steps outlined for computing the interpolation polynomial given by Lemma IV.1 following the steps outlined above by supplying Algorithm 3 with the correct input; this is described in Algorithm 4.

Theorem IV.5. *Algorithm 4 is correct. It has complexity*

$$\mathcal{O}(\nu_Y M(n_X) \log(n_X) + n_X M(\nu_Y) \log(\nu_Y)) \subseteq \tilde{\mathcal{O}}(n_X \nu_Y)$$

operations in \mathbb{F} , where $n_X = n_X(\mathcal{P})$ and $\nu_Y = \nu_Y(\mathcal{P})$.

Proof. Denote by $\tilde{f} \in \mathbb{F}[X, Y]$ the polynomial returned by the algorithm and \hat{f} the polynomial of Lemma IV.1, and we wish to prove $\tilde{f} = \hat{f}$. Note first that for any $\alpha \in \mathcal{X}$ then

$$\hat{f}(\alpha, Y) = \sum_{\beta \in \mathcal{Y}_\alpha} \mathcal{F}[\alpha, \beta] \prod_{\beta' \in \mathcal{Y}_\alpha \setminus \{\beta\}} \frac{y - \beta'}{\beta - \beta'} = \hat{f}_\alpha ,$$

where \hat{f}_α is as computed in Line 5. By the correctness of Combine, then

$$\tilde{f} = \sum_{\alpha \in \mathcal{X}} f_\alpha / \mathcal{R}[\alpha] \prod_{\alpha' \in \mathcal{X} \setminus \{\alpha\}} (X - \alpha') .$$

Algorithm 4: BivariateInterp: Bivariate interpolation

Input: Points $\mathcal{P} \subseteq \mathbb{F}^2$, non-empty. Lookup table of interpolation values $\mathcal{F} \in \mathbb{F}^{\mathcal{P}}$.

Output: The polynomial $\hat{f} \in \mathbb{F}[X, Y]$ given by Lemma IV.1.

```

1  $\mathcal{X} := \mathcal{X}(\mathcal{P}) \subseteq \mathbb{F}$ 
2 foreach  $\alpha \in \mathcal{X}$  do
3    $\mathcal{Y}_\alpha := \mathcal{Y}_\alpha(\mathcal{P}) \subseteq \mathbb{F}$ 
4    $\mathcal{F}_\alpha := (\mathcal{F}[\alpha, \beta])_{\beta \in \mathcal{Y}_\alpha} \in \mathbb{F}^{\mathcal{Y}_\alpha}$ 
5    $\hat{f}_\alpha := \text{UnivariateInterp}_Y(\mathcal{Y}_\alpha, \mathcal{F}_\alpha) \in \mathbb{F}[Y]$ 
6  $\mathcal{T} :=$  a balanced partition tree of  $\mathcal{X}$ 
7  $\mathcal{U} := \text{TreeVanish}_X(\mathcal{T}) \in \mathbb{F}[X]^{\mathcal{T}}$ 
8  $g :=$  formal derivative of  $\mathcal{U}[\mathcal{X}] \in \mathbb{F}[X]$ 
9  $\mathcal{R} := \text{UnivariateMPE}(g, \mathcal{X}) \in \mathbb{F}^{\mathcal{X}}$ 
10  $\mathcal{V} := (\hat{f}_\alpha / \mathcal{R}[\alpha])_{\alpha \in \mathcal{X}} \in \mathbb{F}[Y]^{\mathcal{X}}$ 
11 return Combine( $\mathcal{X}, \mathcal{V}, \mathcal{T}, \mathcal{U}$ )

```

Since $g = \sum_{\alpha \in \mathcal{X}} \prod_{\alpha' \in \mathcal{X} \setminus \{\alpha\}} (X - \alpha')$ we have that

$$\mathcal{R}[\alpha] = \prod_{\alpha' \in \mathcal{X} \setminus \{\alpha\}} (\alpha - \alpha') \in \mathbb{F}, \quad \alpha \in \mathcal{X}.$$

It follows that $\tilde{f} = \hat{f}$.

For complexity we observe that computing \mathcal{T} in Line 6 cost $\mathcal{O}(n_X \log(n_X))$, and \mathcal{U} in Line 7 costs $\mathcal{O}(M(n_X) \log(n_X))$ by Lemma IV.3. Computing g in Line 8 costs $\mathcal{O}(n_X)$ and \mathcal{R} in Line 9 costs $\mathcal{O}(M(n_X) \log(n_X))$ by Proposition II.6. The total cost of computing all the f_α for $\alpha \in \mathcal{X}$ in Line 5 is $\mathcal{O}(n_X M(\nu_Y) \log(\nu_Y))$ by Proposition II.7. Since $\mathcal{R}[\alpha] \in \mathbb{F}$, the division \mathcal{V} in Line 10 costs $\mathcal{O}(n_X \nu_Y)$ operations, one for each of the $\deg \hat{f}_\alpha < \nu_Y$ coefficients in each of the n_X polynomials \hat{f}_α . Finally Line 11 costs $\mathcal{O}(\nu_Y M(n_X) \log(n_X))$ by Theorem IV.4. The total cost becomes as in the theorem. \square

Remark IV.6. If \mathcal{P} is a semi-grid then Algorithm 4 has quasi-linear complexity in the input size $|\mathcal{P}|$. Furthermore, if $\mathbb{F} = \mathbb{F}_q$, then the output polynomial \hat{f} has $\deg_X \hat{f} < n_X$ and $\deg_Y \hat{f} < \nu_Y$, so the $q^{n_X \nu_Y} = q^n$ different polynomials satisfying such degree restrictions must be in bijection with the q^n different choices of the values \mathcal{F} . Hence, Algorithm 4 returns the unique polynomial

\hat{f} satisfying these degree bounds and which interpolate the values. This is in contrast to the case where \mathcal{P} is not a semi-grid, as we will discuss in the following section.

Remark IV.7. If \mathcal{P} is very far from being a semi-grid, the performance of the algorithm can sometimes be improved by a proper blocking-strategy in use of Combine. For example, suppose that $\mathcal{X} = \{\alpha_1, \dots, \alpha_k\}$, and furthermore assume that $\mathcal{Y}_{\alpha_1} = \dots, \mathcal{Y}_{\alpha_{k-1}} = 1$ while $\mathcal{Y}_{\alpha_k} = k$, so that $n = 2k - 1 \in \Theta(k)$. The cost of Algorithm 4 will therefore be $\tilde{O}(k^2)$. However, in this case we can split the points \mathcal{P} into $\mathcal{P}_1 := \{(\alpha, \beta) \in \mathcal{P} \mid \alpha \neq \alpha_k\}$ and $\mathcal{P}_2 := \{(\alpha, \beta) \in \mathcal{P} \mid \alpha = \alpha_k\}$, and the interpolation values into $\mathcal{F}_s := (\mathcal{F}[\alpha, \beta] / \gamma_s)_{(\alpha, \beta) \in \mathcal{P}_s} \in \mathbb{F}^{\mathcal{P}_s}$ for $s = 1, 2$, where $\gamma_s := \prod_{\alpha' \in \mathcal{X}(\mathcal{P}) \setminus \mathcal{X}(\mathcal{P}_s)} (\alpha - \alpha')$ have been precomputed. We can then compute $\hat{f}_s := \text{BivariateInterp}(\mathcal{P}_s, \mathcal{F}_s)$ and obtain the desired interpolating polynomial from Lemma IV.1 as follows:

$$\hat{f} = \sum_{s=1,2} \hat{f}_s \prod_{\alpha \in \mathcal{X}(\mathcal{P}) \setminus \mathcal{X}(\mathcal{P}_s)} (X - \alpha_i).$$

Not counting precomputation, the total cost of this approach becomes $\tilde{O}(k) = \tilde{O}(n)$.

As mentioned in the beginning of this section, the output of Algorithm 4 might not be equal to the original message. In order to obtain the correct message we need to reduce the output further.

2) *Reducing the support:* A priori there is no reason to expect that the computed polynomial \hat{f} , given by Lemma IV.1 is the smallest polynomial interpolating the points according to (a, b) -weighted degree. Moreover, as discussed in Section II-B, when $m \geq n$ then we are looking for a message polynomial with specific monomial support, and not just one of minimal (a, b) -weighted degree. In this section we show how we can use \hat{f} as an initial approximation and from this compute f .

In the following, we will use Gröbner bases and assume that the reader is familiar with the basic notions; see e.g. [6]. Generally, “smallness” in a multivariate polynomial setting is given by specifying a monomial order \leq . For our application, we will order by (a, b) -weighted degree, i.e. the order $\leq_{a,b}$ with $a, b \in \mathbb{Z}_{>0}$ such that:

$$X^{i_1} Y^{j_1} \leq_{a,b} X^{i_2} Y^{j_2} \iff ai_1 + bj_1 < ai_2 + bj_2 \vee (ai_1 + bj_1 = ai_2 + bj_2 \wedge i_1 \geq i_2) .$$

Given a point set $\mathcal{P} \in \mathbb{F}^2$ and an order $\leq_{a,b}$, we will call a monomial $X^i Y^j$ “reducible” if there is a polynomial $g \in \mathbb{F}[X, Y]$ with $\text{LM}_{\leq_{a,b}}(g) = X^i Y^j$ and such that $g(P) = 0$ for all $P \in \mathcal{P}$. We would now like to address the following problem:

Problem IV.8. Given a point set $\mathcal{P} \subseteq \mathbb{F}^2$, a polynomial $\hat{f} \in \mathbb{F}[X, Y]$ with $\deg_Y \hat{f} < \nu_Y := \nu_Y(\mathcal{P})$, and a monomial order $\leq_{a,b}$ on $\mathbb{F}[X, Y]$, compute a polynomial $f \in \mathbb{F}[X, Y]$ according to $\leq_{a,b}$ satisfying

$$f(P) = \hat{f}(P) \text{ for } P \in \mathcal{P} ,$$

and such that no monomial in the support of f is reducible.

Note that the polynomial $f - \hat{f}$ vanishes at all points \mathcal{P} . It is therefore natural to investigate the ideal

$$\mathcal{G} = \{g \mid g(P) = 0 \text{ for } P \in \mathcal{P}\} \subseteq \mathbb{F}[X, Y] . \quad (\text{IV.3})$$

In fact we have the following lemma, which follows immediately from standard properties of Gröbner bases, see e.g. [6, Theorem 3.3]:

Lemma IV.9. In the context of Problem IV.8, let G be a Gröbner basis of \mathcal{G} given in (IV.3) according to $\leq_{a,b}$. Then f is the unique remainder of dividing \hat{f} with G using the multivariate division algorithm, i.e. $f = \hat{f} \text{ rem } G$.

Proposition IV.10. There is an algorithm *Reduce* which inputs a polynomial $\hat{f} \in \mathbb{F}[X, Y]$ and the reduced Gröbner basis $G \subset \mathbb{F}[X, Y]$ of \mathcal{G} from (IV.3) according to $\leq_{a,b}$, and which outputs a solution $f \in \mathbb{F}[X, Y]$ to Problem IV.8. The complexity of the algorithm is:

$$\mathcal{O}(tM(\deg_X(\hat{f})\nu_Y) \log(\deg_X(\hat{f})\nu_Y)) \subset \tilde{\mathcal{O}}(t \deg_X(\hat{f})\nu_Y) ,$$

where $t = |G|$.

Proof. The approach is to compute $\hat{f} \text{ rem } G$ using the fast multivariate division algorithm of van der Hoeven [35], and we simply account for the preconditions and size estimates of Theorem 4 of that paper. Let $G = [G_1, \dots, G_n]$. The division algorithm of van der Hoeven proceeds in a manner analogously to the classical multivariate division algorithm: in each iteration the current remainder R is divided by one of the elements of the basis G_i whose leading monomial according to $\leq_{a,b}$ divides some term of R , and this is then cancelled by scaling and subtracting G_i . This computes $f, Q_1, \dots, Q_t \in \mathbb{F}[X, Y]$ such that

$$\hat{f} = Q_1 G_1 + \dots + Q_t G_t + f ,$$

and which satisfies the following:

- 1) $Q_i G_i \leq_{a,b} \hat{f}$ for $i = 1, \dots, t$; and
- 2) no monomial of f is divisible by $\text{LM}_{\leq_{a,b}}(G_i)$ for any i .

This algorithm requires that the divisors form an “auto-reduced” set, which is a weaker requirement than being a Gröbner basis. The cost of the algorithm is given as

$$\mathcal{O}(M(r) \log(r) + M(s_1) \log(s_1) + \dots + M(s_t) \log(s_t)) .$$

Here s_i denotes the generic size of $\text{supp}(Q_i G_i)$ for $i = 1, \dots, t$, i.e. the cardinality of the set

$$\{(i_Q + i_G, j_Q + j_G) \mid x^{i_Q} y^{j_Q} \in \text{supp } Q_i, x^{i_G} y^{j_G} \in \text{supp } G_i\} ,$$

and r denotes the generic size of $\text{supp } f$. We need to bound the s_i and r . We mentioned above that $Q_i G_i \leq_{a,b} \hat{f}$, and this also implies that $f \leq_{a,b} \hat{f}$. This means the support of all $Q_i G_i$ and of f are within the set

$$\{x^u y^v \mid au + bv \leq \deg_{a,b} \hat{f} \leq a \deg_X(\hat{f}) + b \deg_Y(\hat{f})\} ,$$

which has cardinality less than $\deg_X(\hat{f}) \deg_Y(\hat{f})$. Hence we can see $r, s_1, \dots, s_t \leq \deg_X(\hat{f}) \deg_Y(\hat{f})$, and the complexity estimate follows. □

The last result of this section is to bound the size t of the reduced Gröbner basis of \mathcal{G} for the cases of relevance to unencoding C_{ab} codes:

Lemma IV.11. *Let $\mathcal{P} \subset \mathbb{F}^2$ and let \mathcal{G} be given by (IV.3). Assume \mathcal{G} contains an element g such that $\text{LM}_{\leq_{a,b}}(g) = Y^a$ for some $a \in \mathbb{Z}_{\geq 1}$. Then the reduced Gröbner basis G of \mathcal{G} under monomial order $\leq_{a,b}$ has at most $a + 1$ elements.*

Proof. Since G is a Gröbner basis, the assumption on the existence of $g \in \mathcal{G}$ implies that $G = [G_1, \dots, G_t]$ contains a polynomial, say G_1 , whose leading polynomial divides Y^a , i.e. $\text{LM}_{\leq_{a,b}}(G_1) = y^A$ for $A \leq a$. Since G is a reduced Gröbner basis, this implies that $\deg_Y(G_1) = A$ and $\deg_Y(G_i) < A$ for $i > 1$. Moreover, each $\deg_Y(\text{LM}_{\leq_{a,b}}(G_i))$ must be different, for if $\deg_Y(\text{LM}_{\leq_{a,b}}(G_i)) = \deg_Y(\text{LM}_{\leq_{a,b}}(G_j))$ for $i \neq j$, then either $\text{LM}_{\leq_{a,b}}(G_i) \mid \text{LM}_{\leq_{a,b}}(G_j)$ or $\text{LM}_{\leq_{a,b}}(G_j) \mid \text{LM}_{\leq_{a,b}}(G_i)$, either of which contradicts that G is reduced. This implies $t \leq A + 1 \leq a + 1$. □

We will consider the computation of the reduced Gröbner basis $G \subset \mathbb{F}[X, Y]$ as precomputation, but a small note on the complexity of this computation is in order. The ideal \mathcal{G} is well-studied, and the structure of a lex-ordered Gröbner basis with $x < y$ was already investigated by Lazard [18]. Later and more explicitly, \mathcal{G} appeared as a special case of the ideals studied in

soft-decoding of Reed–Solomon codes using the Kötter–Vardy decoding algorithm, see e.g. [15], [22]. The first reference gives an algorithm for computing a Gröbner basis of \mathcal{G} according to the lex-monomial order with complexity $O(n^2)$, though this can likely be accelerated to $\tilde{O}(n)$ using fast multiplication of univariate polynomials. Neither of the references [15], [22] are directly applicable to compute a Gröbner basis for $\leq_{a,b}$, however, since they only support monomial orders of the form $\leq_{1,b}$ ([15] also supports orders which do not translate into the form $\leq_{a,b}$ however). The order $\leq_{a,b}$ is equivalent to the order $\leq_{1,b/a}$, where we support a rational number as the weight of Y . Such rational weights are handled for a similar Gröbner basis computation in [28], and this would likely lead to an efficient algorithm for our case, though the details are beyond the scope of this paper.

A generic approach is to observe that \mathcal{G} is a zero-dimensional ideal which means that we can use the FGLM algorithm to transform a Gröbner basis from the lex-ordering to one for $\leq_{a,b}$ [7]. The X -degree of the lex-order Gröbner basis output by \mathcal{G} will be n_X , and then the FGLM algorithm has running time at most $\mathcal{O}(n_X^3 \nu_Y^3)$.

B. Fast unencoding

We will now apply results from Section IV-A to the unencoding problem. For a given codeword we compute the interpolating polynomial in Lemma IV.1, which we then reduce using techniques described in Section IV-A2. For algorithm listing see Algorithm 5.

Algorithm 5: Unencode: Unencoding of C_{ab} codes

Input: A C_{ab} code $\mathcal{C}_H(\mathcal{P}, m)$ with $\mathcal{P} = \{P_1, \dots, P_n\}$ being finite rational places. A reduced Gröbner basis of $G \subseteq \mathbb{F}[X, Y]$ of \mathcal{G} as defined in (IV.3) under monomial order $\leq_{a,b}$, where $a = \deg_Y(H)$ and $b = \deg_X(H)$.

Output: $f_H \in \mathcal{L}(mP_\infty)$ in standard form and monomial support in \hat{B} given in Section II-B, and s.t. $f(P_i) = c_i$ for $i = 1, \dots, n$.

- 1 $\mathcal{F} := (c_i)_{P_i \in \mathcal{P}} \in \mathbb{F}^{\mathcal{P}}$
 - 2 $\hat{f} := \text{BivariateInterp}(\mathcal{P}, \mathcal{F}) \in \mathbb{F}[X, Y]$
 - 3 $f := \text{Reduce}(\hat{f}, G) \in \mathbb{F}[X, Y]$
 - 4 **return** $f_H := f(x, y) \in \mathbb{F}[x, y]$
-

Theorem IV.12. *Algorithm 5 is correct. It uses at most*

$$\mathcal{O}(aM(n_X\nu_Y) \log(n_X\nu_Y)) \subset \tilde{\mathcal{O}}(an_X\nu_Y)$$

operations in \mathbb{F} .

Proof. By the correctness of Algorithm 3 then \hat{f} satisfies $\hat{f}(P_i) = c_i$ for $i = 1, \dots, n$ and by Proposition IV.10 then so does f and hence f_H . For the monomial support on f_H , note that $H \in \mathcal{G}$ since H vanishes at all of \mathcal{P} . Since $\text{LM}_{\leq a,b}(H) = Y^a$ then G contains an element $G_1 \in \mathbb{F}[X, Y]$ with $\text{LM}_{\leq a,b}(G_1) \mid Y^a$. Hence $\deg_Y(f) < \deg_Y(G_1) \leq a$ and so f_H is obtained in standard form from f using the natural inclusion of $\mathbb{F}[x, y]$ in $\mathbb{F}[X, Y]$, and the monomial support of f_H corresponds exactly to the monomial support of f . By Proposition IV.10, f contains no reducible monomials, which means that the support of f_H is in \hat{B} .

For complexity, Theorem IV.5 implies that Line 2 costs

$$\mathcal{O}(\nu_Y M(n_X) \log(n_X) + n_X M(\nu_Y) \log(\nu_Y)) .$$

By Lemma IV.1 then $\deg_X(\hat{f}) < n_X$. Since $H \in \mathcal{G}$ then by Lemma IV.11 G contains at most $a + 1$ elements. Therefore Line 3 costs

$$\mathcal{O}(aM(n_X\nu_Y) \log(n_X\nu_Y)) ,$$

by Proposition IV.10, and this dominates the total complexity. As mentioned Line 4 costs no operations in \mathbb{F} . □

Similar to the situation in Section III-B, the complexity of Algorithm 5 depends on the value of a compared to the code length as well as the layout of the evaluation points \mathcal{P} . We will return to analyse special cases in the following section, but the following corollary bounds the complexity in the worst case under very mild assumptions on the C_{ab} code. For codes with $n \approx q$, this cost is not asymptotically better than the naive unencoding using linear algebra, which has cost $\mathcal{O}(n^2)$ assuming some precomputation, but one can keep in mind that AG codes are mostly interesting for use in constructing codes which are markedly longer than the field size.

Corollary IV.13. *In the context of Algorithm 5, let q be the cardinality of \mathbb{F} and assume $n \geq q$. Assume further that the genus g of the C_{ab} curve satisfies $g \leq n$. Then the complexity of Algorithm 5 is $\tilde{\mathcal{O}}(qn) \subset \tilde{\mathcal{O}}(n^2)$.*

Proof. Assuming w.l.o.g. that $a < b$, we use the same upper bound $a \leq \sqrt{n}$ as in the proof of Corollary III.5, as well as the fact that $\nu_Y \leq a$ since for any X -coordinate, there can be at most a solutions in Y to the C_{ab} curve equation. \square

Remark IV.14. We discuss for which C_{ab} codes it could be faster to use the unencoding approach discussed in Section I-A using structured system solving, which costs $\tilde{\mathcal{O}}(a^{\omega-1}n)$. We ignoring hidden constants and log-terms, we see that whenever

$$n_X \nu_Y > n a^{\omega-2}, \quad (\text{IV.4})$$

structured system solving should be faster than our approach. The worst case for our algorithm is $\nu_Y = a$, so (IV.4) becomes $a^{3-\omega} > n/n_X$. Assuming $a < b$ then the genus $g \geq \frac{1}{2}(a-1)^2$ and C_{ab} codes of most interest to coding theory satisfies $g < n$. Asymptotically replacing $a-1$ by a , we conclude that for (IV.4) to hold, then at least $(n/n_X)^{2/(3-\omega)} < 2n$. Now $n_X \leq q$, so this is only possible if $n < 2q^{2/(\omega-1)}$. Taking the best known value for $\omega \approx 2.37286$ [20], we get $n < 2q^{1.46}$. However, in practice we use matrix multiplication algorithms with values of ω quite close to 3. For instance Strassen's multiplication algorithm has exponent $\hat{\omega} \approx 2.81$ [33], which would give the condition $n < q^{1.1}$, which can be considered a quite short C_{ab} code.

V. APPLICATIONS

In this section, we will apply Algorithm 1 to the encoding and unencoding for various AG codes coming from C_{ab} curves. As we will show, in many interesting cases we can encode and unencode faster than $\mathcal{O}(n^2)$, in some cases even in $\tilde{\mathcal{O}}(n)$.

A. Quasi-linear encoding and unencoding for C_{ab} curves on semi-grids

We already observed in Remark III.2 that our multipoint evaluation algorithm has very good complexity when the evaluation points lie on a semi-grid. In this section we therefore investigate certain C_{ab} curves having many points that lie on a semi-grid and the complexity of our algorithms for codes over such curves. Specifically, we will be interested in the following types of C_{ab} codes:

Definition V.1. A C_{ab} code $\mathcal{C}_H(\mathcal{P}, m)$ is called maximal semi-grid if \mathcal{P} is a semi-grid with $\nu_Y(\mathcal{P}) = a =: \deg_Y(H)$ and $m < n$.

Note the condition $m < n$ means that the encoding map is injective on $\mathcal{L}(mP_\infty)$ so the complications discussed in Section II-B do not apply.

Proposition V.2. *Let $\mathcal{C}_H(\mathcal{P}, m)$ be a maximal semi-grid C_{ab} code of length n . Then encoding using Algorithm 2 has complexity $\mathcal{O}(M(n) \log(n)) \in \tilde{\mathcal{O}}(n)$.*

Proof. Since $\mathcal{C}_H(\mathcal{P}, m)$ is maximal semi-grid then $an_X(\mathcal{P}) = \nu_Y(\mathcal{P})n_X(\mathcal{P}) = |\mathcal{P}| = n$, and further $m < n$. Hence, the result follows from Theorem III.4. \square

The cost of unencoding using Algorithm 5 is dominated by the call to Reduce in Line 3. The following proposition shows that for maximal semi-grid C_{ab} code, this step can be omitted. First a small lemma.

Lemma V.3. *Let $\mathcal{C}_H(\mathcal{P}, m) \subseteq \mathbb{F}^n$ be a maximal semi-grid C_{ab} code. Let $\mathcal{G}_H = \{g \in \mathbb{F}[x, y] \mid g(P) = 0 \text{ for all } P \in \mathcal{P}\}$. Then $\mathcal{G}_H = G \cdot \mathbb{F}[x, y]$, where $G = \prod_{\alpha \in \mathcal{X}(\mathcal{P})} (x - \alpha)$.*

Proof. Clearly $G \in \mathcal{G}_H$ so $G \cdot \mathbb{F}[x, y] \subseteq \mathcal{G}_H$. For the other inclusion, observe first that we could write $\mathcal{G}_H = \bigcup_{i \in \mathbb{Z}} \mathcal{L}(-D + iP_\infty)$, where $D = \sum_{P \in \mathcal{P}} P$. Note now that the pole order of $x - \alpha$ at P_∞ is a , and it has poles nowhere else. On the other hand, it has a zero at each of the points $\{(\alpha, \beta) \mid \beta \in \mathcal{Y}_\alpha(\mathcal{P})\}$ and there are $\nu_Y(\mathcal{P}) = a$ of them. Hence, the divisor of $x - \alpha$ is given exactly as

$$(x - \alpha) = \sum_{\beta \in \mathcal{Y}_\alpha} P_{(\alpha, \beta)} - aP_\infty ,$$

where $P_{\alpha, \beta} \in \mathcal{P}$ is the place corresponding to the point $(\alpha, \beta) \in \mathcal{P}$. It follows that $(G) = D - nP_\infty$. Therefore, for any $z \in \mathcal{G}_H$ we have $z/G \in \mathcal{L}(sP_\infty)$ for some $s \in \mathbb{Z}$, i.e. z/G has only poles at P_∞ and so $z/G \in \mathbb{F}[x, y]$. Hence $\mathcal{G}_H \subseteq G \cdot \mathbb{F}[x, y]$, and we conclude equality. \square

Proposition V.4. *Let $\mathcal{C}_H(\mathcal{P}, m) \subseteq \mathbb{F}^n$ be a maximal semi-grid C_{ab} code. Let $f \in \mathcal{L}(mP_\infty)$ and let $\hat{f} \in \mathbb{F}[X, Y]$ be given by Lemma IV.1 where $\mathcal{F}[P] = f(P)$ for all $P \in \mathcal{P}$. Then $f = \hat{f}(x, y)$.*

Proof. We write ν_Y instead of $\nu_Y(\mathcal{P})$ in the following, and similarly for $\mathcal{X}, \mathcal{Y}_\alpha, n_X$, etc. Note first that $\hat{f} \in \mathbb{F}[X, Y]$ has $\deg_Y \hat{f} < \nu_Y = a$ so $\hat{f}(x, y)$ is obtained in standard form by the natural inclusion of $\mathbb{F}[x, y]$ in $\mathbb{F}[X, Y]$.

Let $G(x)$ and \mathcal{G}_H be given as in Lemma V.3. Note that $f - \hat{f}(x, y) \in \mathcal{G}_H = G \cdot \mathbb{F}[x, y]$. Write $f - \hat{f}(x, y) = \tilde{f}_0(x) + \tilde{f}_1(x)y + \dots + \tilde{f}_{a-1}(x)y^{a-1}$ in standard form with $\tilde{f}_i \in \mathbb{F}[x]$. Since $G(x)$ is univariate in x , then $G(x)$ must divide every $\tilde{f}_i(x)$. Hence, if $f - \hat{f}(x, y) \neq 0$, we must have

$$\begin{aligned}
 \deg_x G &\leq \deg_x(\tilde{f}_i) \text{ for some } i \in \{0, \dots, a-1\}. \text{ But} \\
 &\leq \max(\deg_{a,b}(f)/a, \deg_X(\hat{f})) \\
 &\leq \max(m/a, n_X - 1) \\
 &< n_X = \deg_x G,
 \end{aligned}$$

where the last inequality follows from \mathcal{P} being a semi-grid and so $n = |\mathcal{P}| = \nu_Y n_X = a n_X$. This is a contradiction. □

Proposition V.5. *Let $\mathcal{C}_H(\mathcal{P}, m)$ be a maximal semi-grid C_{ab} code of length n . There is an algorithm for unencoding $\mathcal{C}_H(\mathcal{P}, m)$ with complexity $\mathcal{O}(\nu_Y M(n_X) \log(n_X) + n_X M(n_Y) \log(n_Y)) \in \tilde{\mathcal{O}}(n)$.*

Proof. The algorithm is simply Algorithm 5 with the following two changes:

- 1) We return \hat{f} in Line 2 and skip Line 3.
- 2) We do not take the Gröbner basis $G \in \mathbb{F}[X, Y]$ as input.

That this algorithm is correct follows from Proposition V.4 since the code is maximal semi-grid. The big- \mathcal{O} complexity is exactly that of Theorem IV.5 and the relaxation follows from $n_X(\mathcal{P})\nu_Y(\mathcal{P}) = n$. □

We stress that in contrast to the general unencoding algorithm, Algorithm 5, the unencoding algorithm for maximal semi-grid codes requires no precomputation.

We proceed by showing that several interesting classes of C_{ab} curves admit long maximal semi-grid codes.

1) *The Hermitian curve:* The Hermitian curve is defined for fields $\mathbb{F} = \mathbb{F}_{q^2}$ for some prime power q using the defining polynomial

$$H_{\mathcal{H}}(X, Y) = Y^q + Y - X^{q+1}. \tag{V.1}$$

It is easily checked that it is a C_{ab} curve, with $a = q$ and $b = q + 1$. We say that a C_{ab} code is a *Hermitian code* if the curve equation that is used is Equation (V.1).

The Hermitian curve and its function field are well known and have been studied extensively in the literature, see e.g. [32, Lemma 6.4.4]. For instance, using the trace and norm maps of the extension $\mathbb{F}_{q^2} : \mathbb{F}_q$, it is easy to show that for every $\alpha \in \mathbb{F}_{q^2}$ there exist precisely q distinct elements $\beta \in \mathbb{F}_{q^2}$ such that $H_{\mathcal{H}}(\alpha, \beta) = 0$. In other words, the set of rational points $\mathcal{P}_{\mathcal{H}}$ on $H_{\mathcal{H}}$ form a semi-grid with $\mathcal{X}(\mathcal{P}) = \mathbb{F}_{q^2}$ and $\nu_Y(\mathcal{P}) = q = a$, i.e. a total of q^3 points.

Therefore, the Hermitian code $\mathcal{C}_{H_{\mathcal{H}}}(\mathcal{P}_{\mathcal{H}}, m)$ with $m < n = q^3$ is maximal semi-grid since $\mathcal{P}_{\mathcal{H}}$ is a semi-grid with $\nu_Y(\mathcal{P}) = \deg_Y(H)$. We immediately get the following corollary of Proposition V.2 and Proposition V.5:

Corollary V.6. *Let $\mathcal{C}_{H_{\mathcal{H}}}(\mathcal{P}_{\mathcal{H}}, m) \in \mathbb{F}_{q^2}^n$ be an Hermitian code with $m < n = |\mathcal{P}_{\mathcal{H}}|$, and $\mathcal{P}_{\mathcal{H}}$ the set of all rational points on the $H_{\mathcal{H}}$ as given in (V.1). Then encoding using Algorithm 1 uses*

$$\mathcal{O}(M(m+n)\log(m+n)) \subset \tilde{\mathcal{O}}(n)$$

operations in \mathbb{F}_{q^2} . Unencoding using the algorithm of Proposition V.5 uses

$$\mathcal{O}(q(M(q^2))\log(q)) \subset \tilde{\mathcal{O}}(n)$$

operations in \mathbb{F}_{q^2} .

Note that encoding and unencoding also has quasi-linear complexity for any shorter Hermitian code, where we use any sub semi-grid of the points $\mathcal{P} \subset \mathcal{P}_{\mathcal{H}}$ as long as $\nu_Y(\mathcal{P}) = \nu_Y(\mathcal{P}_{\mathcal{H}}) = a$. This corresponds to selecting a number $n_X \leq q^2$ of X -coordinates, and for each choosing all the q points in $\mathcal{P}_{\mathcal{H}}$ having this X -coordinate.

2) *Norm-trace and other Hermitian-like curves:* It is not hard to find other examples of C_{ab} curves which admit large maximal semi-grid codes. In this subsection we give examples of curves from the literature which have this property.

Let q be a prime power and $r \in \mathbb{Z}_{\geq 2}$. Further let e be a positive integer dividing the integer $(q^r - 1)/(q - 1)$ and define

$$H_{\mathcal{N},e}(X, Y) = X^{q^{r-1}} + \dots + X^q + X - Y^e \in \mathbb{F}_{q^r}[X, Y], \quad (\text{V.2})$$

This is a C_{ab} polynomial with $a = e$ and $b = q^{r-1}$.

We first look at the case $e = (q^r - 1)/(q - 1)$ which gives rise to the norm-trace curves studied in [11]. For $r = 2$ they simplify to the Hermitian curve. Similarly to the Hermitian curve, one obtains that the set of points $\mathcal{P}_{\mathcal{N}}$ of $H_{\mathcal{N}}(X, Y)$ form a semi-grid with $n_X(\mathcal{P}_{\mathcal{H}}) = q^r$ and $\nu_Y(\mathcal{P}_{\mathcal{H}}) = a$ and therefore $|\mathcal{P}_{\mathcal{N}}| = q^{2r-1}$. Corollary V.6 generalizes directly and shows that one-point norm-trace codes can be encoded and unencoded in quasi-linear time:

Corollary V.7. *Let q be a prime power, $r \in \mathbb{Z}_{\geq 2}$, and $e = (q^r - 1)/(q - 1)$. Further let $H_{\mathcal{N}}(X, Y)$ be as in (V.2), and $\mathcal{P}_{\mathcal{N}}$ the set of rational points on $H_{\mathcal{N}}$. Let $\mathcal{C}_{H_{\mathcal{N}}}(\mathcal{P}_{\mathcal{N}}, m)$ be the corresponding C_{ab} code for some $m < n = |\mathcal{P}_{\mathcal{N}}| = q^{2r-1}$. Then encoding using Algorithm 1 uses*

$$\mathcal{O}(M(m+n)\log(m+n)) \subset \tilde{\mathcal{O}}(n)$$

operations in \mathbb{F}_{q^r} . Unencoding using the algorithm of Proposition V.5 uses

$$\mathcal{O}(q^{r-1}(\mathbf{M}(q^r)) \log(q^r)) \subset \tilde{\mathcal{O}}(n)$$

operations in \mathbb{F}_{q^r} .

If $e < (q^r - 1)/(q - 1)$, the equation $H_{\mathcal{N},e}(\alpha, \beta) = 0$ has $q^{r-1} + e(q^r - q^{r-1})$ solutions in $\mathbb{F}_{q^r}^2$. The small term q^{r-1} comes from the solutions where $\beta = 0$ and $\alpha^{q^{r-1}} + \dots + \alpha = 0$. The remaining $e(q^r - q^{r-1})$ points again form a semi-grid $\mathcal{P}_{\mathcal{N},e}$ with $\nu_Y(\mathcal{P}_{\mathcal{N},e}) = e = a$ and $n_X = q^r - q^{r-1}$, and can therefore be used to construct long codes with efficient encoding and decoding. A special case of these curves, where r is even and e divides $q^{r/2} + 1$ was considered in [26]. We obtain the following.

Corollary V.8. *Let q be a prime power, $r \in \mathbb{Z}_{\geq 2}$, and e an integer dividing $(q^r - 1)/(q - 1)$, but not equal to it. Further let $H_{\mathcal{N},e}$ be given by (V.2), and $\mathcal{P}_{\mathcal{N},e}$ the set of rational points on $H_{\mathcal{N},e}$. Let $\mathcal{C}_{H_{\mathcal{N},e}}(\mathcal{P}_{\mathcal{N},e}, m)$ be the corresponding C_{ab} code for some $m < n = |\mathcal{P}_{\mathcal{N},e}| = e(q^r - q^{r-1})$. Then encoding using Algorithm 1 uses*

$$\mathcal{O}(\mathbf{M}(m + n) \log(m + n)) \subset \tilde{\mathcal{O}}(n)$$

operations in \mathbb{F}_{q^r} . Unencoding using the algorithm of Proposition V.5 uses

$$\mathcal{O}(e(\mathbf{M}(q^r)) \log(q^r)) \subset \tilde{\mathcal{O}}(n)$$

operations in \mathbb{F}_{q^r} .

B. Fast encoding for good families of C_{ab} curves

We will now investigate the complexity of Algorithm 1 for families of curves that have many points. More precisely, we will consider curves for which the number of points is asymptotically close to the *Hasse–Weil bound*:

Theorem V.9 (Hasse–Weil bound, [32, Theorem 5.2.3]). *If N is the number of rational places of an algebraic function field F over \mathbb{F}_q , then*

$$N \leq 2g\sqrt{q} + (q + 1),$$

where g is the genus of F .

For C_{ab} curves we know $g = \frac{1}{2}(a-1)(b-1)$, so the Hasse–Weil bound upper-bounds the length n of a C_{ab} code $\mathcal{C}(H, m)$ over \mathbb{F}_q as follows:

$$n \leq \text{HW}(H) := (a-1)(b-1)\sqrt{q} + q .$$

Observe that this is one less than the bound on the number of rational places of the function field corresponding to the C_{ab} curve, since the rational place at infinity is not included as an evaluation point. The Hermitian curve is an example of a curve which attains the Hasse–Weil bound.

Lemma V.10. *Let $\mathcal{C}_H(\mathcal{P}, m)$ be a C_{ab} code over \mathbb{F}_q , with $a = \deg_Y(H)$, $b = \deg_X(H)$ and $a < b$. Let n be the length of the code, and assume that $n \geq q$ as well as $n \geq c \cdot \text{HW}(H)$ for some constant $c \in (0, 1]$. Then the following upper bounds hold:*

$$\begin{aligned} a &< \sqrt{\frac{n}{c\sqrt{q}}} + 1 ; \\ qa &< \frac{n^{5/4}}{\sqrt{c}} + n ; \\ qa^2 &< \frac{n^{3/2}}{c} + \frac{2n^{5/4}}{\sqrt{c}} + 2n . \end{aligned}$$

Proof. Since $n \geq c \cdot \text{HW}(H) \geq c(a-1)(b-1)\sqrt{q}$ and $a < b$ we have that

$$n > c(a-1)^2\sqrt{q} \iff \frac{n}{c\sqrt{q}} > (a-1)^2 ,$$

which gives the first bound. Since $q \leq n$ we then also get

$$qa < q^{3/4}\sqrt{\frac{n}{c}} + q \leq \frac{n^{5/4}}{c^{1/2}} + n .$$

Lastly, $qa^2 < q((a-1)^2 + 2a)$ and so

$$qa^2 < \frac{n\sqrt{q}}{c} + 2qa \leq \frac{n^{3/2}}{c} + 2qa ,$$

and the last bound follows by inserting our earlier bound for qa . □

In the following, we will discuss the asymptotic complexity of encoding and unencoding for infinite families of C_{ab} codes. Note that for this to make any sense, the length of the codes must go to infinity and therefore the size of the fields over which the codes are defined must also go to infinity. In the remainder of the section, when we introduce an infinite family of C_{ab} curves

$\Gamma = \{\mathcal{C}_{H_i}(\mathcal{P}_i, m_i)\}_{i \in \mathbb{Z}_{\geq 1}}$, we also implicitly introduce the related variables: q_i is the prime power such that H_i and the code $\mathcal{C}_{H_i}(\mathcal{P}_i, m_i)$ is defined over \mathbb{F}_{q_i} ; $a_i := \deg_Y(H_i)$ and $b_i := \deg_X(H_i)$ and we assume $a_i < b_i$; and n_i is the length of the code for each i .

For an infinite sequence of real numbers $\mathbf{c} = (c_1, c_2, \dots) \in (0, 1]^\infty$, we say that the code family Γ is *asymptotically \mathbf{c} -good* if $n_i \geq c_i \text{HW}(H_i)$ for all $i = 1, 2, \dots$.

Theorem V.11. *Let $\Gamma = \{\mathcal{C}_{H_i}(\mathcal{P}_i, m_i)\}_{i \in \mathbb{Z}_{\geq 1}}$ be an infinite family of C_{ab} codes with related variables q_i, a_i, b_i, n_i , with $a_i < b_i$, which is asymptotically \mathbf{c} -good for $\mathbf{c} = (c_1, c_2, \dots)$. Then the asymptotic complexity of encoding $\mathcal{C}_{H_i}(\mathcal{P}_i, m_i)$ for $i \rightarrow \infty$ using Algorithm 2 is*

$$\mathcal{O}(M(m_i + n_i^{5/4}/\sqrt{c_i}) \log(m_i + n_i^{5/4}/\sqrt{c_i})) \subset \tilde{\mathcal{O}}(m_i + n_i^{5/4}/\sqrt{c_i})$$

operations in \mathbb{F}_{q_i} . The asymptotic complexity of unencoding $\mathcal{C}(H_i, m_i)$ for $i \rightarrow \infty$ using Algorithm 5 is

$$\mathcal{O}(M(n_i^{3/2}/c_i) \log(n_i^{3/2}/c_i)) \subset \tilde{\mathcal{O}}(n_i^{3/2}/c_i)$$

operations in \mathbb{F}_{q_i} .

Proof. Theorem III.4 gives the asymptotic cost of encoding $\mathcal{C}(H_i, m_i)$ as

$$\mathcal{O}(M(m_i + a_i n_{X,i}) \log(m_i + a_i n_{X,i})) ,$$

operations in \mathbb{F}_{q_i} , where $n_{X,i}$ is the number of distinct X -coordinates in the evaluation points used in $\mathcal{C}_{H_i}(\mathcal{P}_i, m_i)$. Since $n_{X,i} \leq q_i$ we can use the bound on $a_i q_i$ given by Lemma V.10. In the big-Oh notation, the lower-order terms can be ignored, and this gives the estimate of encoding.

For unencoding, the cost is given by Theorem IV.12 as

$$\mathcal{O}(M(a_i^2 n_{X,i}) \log(a_i^2 n_{X,i})) .$$

We use $a_i^2 n_{X,i} \leq a_i^2 q_i$ which is then bounded by Lemma V.10. Note that since $c_i \in (0, 1]$ then $\sqrt{c_i} > c_i$ and so we always have $n^{3/2}/c_i \geq n^{5/4}/\sqrt{c_i}$, so we need only keep the term $n^{3/2}/c$ in the asymptotic estimate. \square

Let us discuss some consequences of this result. Consider first that all $c_i = c$ for some fixed constant $0 < c \leq 1$, i.e. that all the curves of the codes in Γ are a factor c from Hasse–Weil: then we can encode using $\tilde{\mathcal{O}}(n_i^{5/4})$ operations in \mathbb{F}_{q_i} , or $\tilde{\mathcal{O}}(n_i^{5/4} \log(q_i))$ bit-operations, which is significantly better than the naive approach of roughly $\mathcal{O}(n_i m_i)$ operations in \mathbb{F}_{q_i} . Though the constant c disappears in the asymptotic estimate, Theorem V.11 describes by the dependency

on $1/\sqrt{c}$ how the encoding algorithm fares on asymptotically worse families compared to asymptotically better families. For instance, if $c = 1/100$ and Γ consists of C_{ab} codes over curves achieving only 1% of the Hasse–Weil bound, the running time of the algorithm will be roughly 10 times slower pr. encoded symbol compared to running the algorithm on a family which attains the Hasse–Weil bound.

Theorem V.11 is useful also for families of curves which get farther and farther away from the Hasse–Weil bound. Indeed as long as $1/\sqrt{c_i}$ grows slower than $n_i^{3/4}$, i.e. c_i stays above $n_i^{-9/16}$ times a constant, we still get an improvement over the naive encoding algorithm.

Remark V.12. *An alternative unencoding approach of structured system solving described in Section I-A has a cost of $\tilde{O}(a^{\omega-1}n)$. It does not seem easy to completely fairly compare this cost with that of Theorem V.11, but we can apply a similar over-bounding strategy and get a single exponent for n : By Lemma V.10 then $a_i < n_i^{1/2}q_i^{-1/4}c^{-1/2}$. Note that $q_i^2 \geq n_i$ so we get $a_i < n_i^{3/8}c_i^{-1/2}$ and hence if we replace a by this bound in the cost of the structured system solving we get $\tilde{O}(n_i^{1+3/8(\omega-1)})$ for $i \rightarrow \infty$. This is roughly $\tilde{O}(n_i^{1.515})$ if we use the best known value for $\omega \approx 2.37286$ [20]. For the more practical matrix multiplication algorithms of Strassen with $\hat{\omega} \approx 2.81$ [33], we get $\tilde{O}(n_i^{1.68})$, and simply replacing ω by 3 yields $\tilde{O}(n_i^{1.75})$.*

REFERENCES

- [1] P. Beelen and K. Brander. Efficient list decoding of a class of algebraic-geometry codes. *Advances in Mathematics of Communications*, 4(4):485–518, Nov. 2010.
- [2] P. Beelen and R. Pellikaan. The newton polygon of plane curves with many rational points. *Designs, Codes and Cryptography*, 21(1):41–67, Oct 2000.
- [3] A. Bostan, C.-P. Jeannerod, C. Moulleron, and E. Schost. On matrices with displacement structure: generalized operators and faster algorithms. *arXiv:1703.03734 [cs]*, Mar. 2017. arXiv: 1703.03734.
- [4] A. Bostan, C.-P. Jeannerod, and E. Schost. Solving structured linear systems with large displacement rank. *Theoretical Computer Science*, 407(1–3):155–181, Nov. 2008.
- [5] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, July 1991.
- [6] D. A. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra (Undergraduate Texts in Mathematics)*. Springer, 3rd edition edition, 2007.
- [7] J. C. 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, Oct. 1993.
- [8] G.-L. Feng and T. Rao. Simple approach for construction of algebraic-geometric codes from affine plane curves. *Information Theory, IEEE Transactions on*, 40:1003 – 1012, 08 1994.
- [9] S. Gao. Absolute irreducibility of polynomials via newton polytopes. *Journal of Algebra*, 237(2):501 – 520, 2001.

- [10] A. Garcia and H. Stichtenoth. A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vladut bound. *Inventiones Mathematicae*, 121(1):211–222, Dec. 1995.
- [11] O. Geil. On codes from norm–trace curves. *Finite Fields and Their Applications*, 9(3):351 – 371, 2003.
- [12] V. D. Goppa. Algebraico-Geometric Codes. *Mathematics of the USSR-Izvestiya*, 21(1):75, 1983.
- [13] D. Harvey, J. V. D. Hoeven, and G. Lecerf. Faster polynomial multiplication over finite fields. *Journal of the ACM*, 63(6), Feb. 2017.
- [14] T. Høholdt, J. Lint, and R. Pellikaan. Algebraic geometry of codes, handbook of coding theory. *Amsterdam*, pages 871–961, 01 1998.
- [15] C.-P. Jeannerod, V. Neiger, E. Schost, and G. Villard. Computing minimal interpolation bases. *Journal of Symbolic Computation*, 83:272–314, Nov. 2017.
- [16] J. Justesen. On the complexity of decoding Reed-Solomon codes (Corresp.). *IEEE Transactions on Information Theory*, 22(2):237–238, Mar. 1976.
- [17] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–155, Oct 2008.
- [18] D. Lazard. Ideal bases and primary decomposition: case of two variables. *Journal of Symbolic Computation*, 1(3):261–270, 1985.
- [19] F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 514–523, Oct 2012.
- [20] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014.
- [21] K. Lee, M. Bras-Amoros, and M. O’Sullivan. Unique Decoding of General AG Codes. *IEEE Transactions on Information Theory*, 60(4):2038–2053, Apr. 2014.
- [22] K. Lee and M. E. O’Sullivan. An Interpolation Algorithm Using Gröbner Bases for Soft-Decision Decoding of Reed-Solomon Codes. In *IEEE International Symposium on Information Theory*, pages 2032–2036, 2006.
- [23] K. Lee and M. E. O’Sullivan. List decoding of Hermitian codes using Gröbner bases. *Journal of Symbolic Computation*, 44(12):1662–1675, 2009.
- [24] S. Miura. Algebraic geometric codes on certain plane curves. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 76(12):1–13, 11 1993.
- [25] S. Miura and N. Kamiya. Geometric-goppa codes on some maximal curves and their minimum distance. *Proceedings of 1993 IEEE Information Theory Workshop*, pages 85–86, 06 1993.
- [26] C. Munuera, A. Ulveda, and F. Torres. Castle curves and codes. *Advances in Mathematics of Communications*, 3, 11 2009.
- [27] A. K. Narayanan and M. Weidner. Nearly linear time encodable codes beating the Gilbert-Varshamov bound. Dec. 2017.
- [28] J. Nielsen and P. Beelen. Sub-Quadratic Decoding of One-Point Hermitian Codes. *IEEE Transactions on Information Theory*, 61(6):3225–3240, June 2015.
- [29] M. Nüsken and M. Ziegler. Fast multipoint evaluation of bivariate polynomials. In S. Albers and T. Radzik, editors, *Algorithms – ESA 2004*, pages 544–555, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [30] V. Y. Pan. Simple Multivariate Polynomial Multiplication. *Journal of Symbolic Computation*, 18(3):183–186, Sept. 1994.
- [31] S. Sakata, H. E. Jensen, and T. Høholdt. Generalized Berlekamp-Massey Decoding of Algebraic-Geometric Codes up to Half the Feng–Rao Bound. *IEEE Transactions on Information Theory*, 41(6):1762–1768, 1995.
- [32] H. Stichtenoth. *Algebraic Function Fields and Codes*. Springer, 2nd edition, 2009.
- [33] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

- [34] M. A. Tsfasman, S. G. Vladut, and T. Zink. Modular curves, Shimura curves, and Goppa codes, better than Varshamov-Gilbert bound. *Mathematische Nachrichten*, 109(1):21–28, 1982.
- [35] J. Van Der Hoeven. On the complexity of multivariate polynomial division. In *Special Sessions in Applications of Computer Algebra*, pages 447–458. Springer, 2015.
- [36] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2012.