Partial Inverses mod $m(x)$ and Reverse Berlekamp–Massey Decoding

Jiun-Hung Yu, *Member, IEEE*, and Hans-Andrea Loeliger, *Fellow, IEEE*

***Abstract*— This semi-tutorial paper introduces the partial-inverse problem for polynomials and develops its application to decoding Reed–Solomon codes and some related codes. The most natural algorithm to solve the partial-inverse problem is very similar to, but more general than, the Berlekamp–Massey algorithm. Two additional algorithms are obtained as easy variations of the basic algorithm: the first variation is entirely new, while the second variation may be viewed as a version of the Euclidean algorithm. Decoding Reed–Solomon codes (and some related codes) can be reduced to the partial-inverse problem, both via the standard key equation and, more naturally, via an alternative key equation with a new converse. Shortened and singly-extended Reed–Solomon codes are automatically included. Using the properties of the partial-inverse problem, two further key equations with attractive properties are obtained. The paper also points out a variety of options for interpolation.**

***Index Terms*— Reed–Solomon codes, polynomial remainder codes, key equation, partial-inverse problem, partial-inverse algorithm, Euclidean algorithm, Berlekamp–Massey algorithm.**

I. Introduction

In this paper, we consider the following problem and its application to decoding Reed–Solomon codes and some related codes.

Partial-Inverse Problem in $\mathbf{F[x]/m(x)}$: Let $b(x)$ and $m(x)$ be nonzero polynomials over some field F, with $\deg b(x) < \deg m(x)$. For fixed $d \in \mathbb{Z}$ with $0 \le d \le \deg m(x)$, find a nonzero polynomial $\Lambda(x) \in F[x]$ of the smallest degree such that

$$\deg\Big(b(x)\Lambda(x) \bmod m(x)\Big) < d. \tag{1}$$

□

We will see that this problem has always a unique solution (up to a scale factor), and the solution satisfies

$$\deg \Lambda(x) \le \deg m(x) - d. \tag{2}$$

In the special case where $d = 1$ and $\gcd\big(b(x), m(x)\big) = 1$, the solution $\Lambda(x)$ is the inverse of $b(x)$ in $F[x]/m(x)$.

The partial-inverse problem is strongly reminiscent of various "key equations" in the literature of Reed–Solomon codes [2]–[8] and similar codes, and it is essentially equivalent to a modified Padé approximation problem due to McEliece and Shearer, which can be solved by the Euclidean algorithm [9], (see also [10, Sec. 5.7] and Appendix A). However, the exact problem statement is new.

This paper is semi-tutorial in the following sense: much of the material looks almost standard, but is not quite so; by developing the theory from a new starting point—the partial inverse problem—we gain generality and clarity, we unify results from different approaches in the literature, and we obtain many novelties in detail.

To set the stage, recall the two classical algorithms to decode Reed–Solomon codes and some related codes: the Berlekamp–Massey algorithm [3], [4] and the Euclidean algorithm (due to Sugiyama [5]). The Euclidean algorithm is considered to be more versatile (but perhaps slightly less efficient) than the Berlekamp–Massey algorithm; for example, the popular Shiozaki–Gao decoder [11], [12] is also based on the Euclidean algorithm, see also [13] and Section V-D. The Berlekamp–Massey algorithm and the Euclidean decoding algorithm are actually related: explicit (and nontrivial) translations were given in [14]–[17], and further connections were established in [18] and [19].

In order to clarify Berlekamp's algorithm [3], Massey [4] introduced the linear-feedback shift-register synthesis (LFSRS) problem: this is the problem that the Berlekamp–Massey algorithm solves intrinsically, and from which its application to decoding is developed most naturally. The LFSRS problem is similar, but not identical, to the partial-inverse problem for $m(x) = x^\nu$; in particular, the LFSRS problem does not always have a unique solution.

Likewise, in order to clarify Sugiyama's algorithm (the Euclidean algorithm), McEliece and Shearer [9] introduced a modified Padé approximation problem that is essentially equivalent to the partial-inverse problem (cf. Appendix A). Essentially the same problem is also discussed in [10, Sec. 5.7]. In this prior literature, the properties of the problem (as far as they are investigated) are all derived from the Euclidean algorithm.

In this paper, we develop another perspective. First, we investigate the partial-inverse problem without regard to any algorithm, and we find many new properties of it that will be helpful for decoding later on (cf. Sections II and III). In particular, we prove (2), we give a characterization of the solutions in terms of a gcd (Theorem 1), we discuss irrelevant coefficients in $b(x)$ and $m(x)$ (and how to get rid of them), and we show that the partial-inverse problem for general

Manuscript received September 23, 2015; revised July 17, 2016; accepted September 10, 2016. Date of publication September 26, 2016; date of current version November 18, 2016. This paper was presented in part at the 2013 IEEE International Symposium on Information Theory [1].

The authors are with the Department of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland. (e-mail: yu@isi.ee.ethz.ch; loeliger@isi.ee.ethz.ch).

Communicated by C. Xing, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2016.2613559

$m(x)$ can be transformed into a partial-inverse problem with $m(x) = x^\nu$.

Second, we find that the most natural algorithm for solving the partial-inverse problem is not the Euclidean algorithm, but a new algorithm that resembles the Berlekamp–Massey algorithm. In some special cases (including $m(x) = x^\nu - 1$, $m(x) = x^\nu$, and $m(x) = x^\nu - x$), the algorithm looks indeed very much like, and is as efficient as, the Berlekamp–Massey algorithm (but the input $b(x)$ is processed in the reverse order). We then note that this new algorithm is easily translated into two further algorithms, one of which is entirely new and the other may be viewed as a version of the Euclidean algorithm. We thus obtain a transparent unification of the Berlekamp–Massey approach with the Sugiyama approach, an additional new algorithm, and a derivation of the Euclidean algorithm that begins with the partial-inverse problem rather than with the computation of a gcd.

Third, we show that decoding Reed–Solomon codes (and some related codes) can be reduced to the partial-inverse problem, not only via the standard key equation, but also (and more naturally) via an alternative key equation with a new converse. Shortened and singly-extended Reed–Solomon codes (with an evaluation point at zero) are automatically included, and the approach is easily adapted to polynomial remainder codes [12], [20]–[23], which have not been amenable to Berlekamp–Massey decoding. Using our earlier results on the partial-inverse problem, we obtain two further key equations with attractive properties.

Fourth, we address the interpolation problem, i.e., the problem of recovering the codeword from the error locator polynomial. We give two general transform-domain interpolation formulas, one of which has implicitly been used in the Shiozaki–Gao decoder [11], [12] while the other appears to be entirely new. In addition, we adapt Horiguchi–Koetter interpolation [7], [24], which has been tied to Berlekamp–Massey decoding, to our more general setting.

In the appendices, we discuss extensions of the partial-inverse approach: first, to errors-and-erasures decoding, and second, to polynomial remainder codes (which have not been amenable to Berlekamp–Massey decoding).

We also note here that the partial-inverse approach can be extended to decode interleaved Reed–Solomon codes and similar codes beyond half the minimum distance [25], [26]. However, this extension is beyond the scope of the present paper and will be fully developed in a companion paper.

This paper is structured as follows. The most basic properties of the partial-inverse problem are given in Section II. Many additional new properties of the partial-inverse problem are discussed in Section III. The new basic partial-inverse algorithm and its two variations are given in Section IV. Decoding Reed–Solomon codes is addressed in Section V. The generalization to polynomial remainder codes is given in Section VI, and Section VII concludes the main part of the paper.

Additional pertinent material is given in the appendices. Appendix A addresses the relations between the partial-inverse problem and variations of Padé approximation problems. Appendix C shows how the standard key equation of Reed–Solomon codes can be formulated as a partial-inverse problem and solved by the algorithms of this paper. Appendix D addresses an extended partial-inverse problem where $\Lambda(x)$ is required to have a given factor. Errors-and-erasures decoding of Reed–Solomon codes and of polynomial remainder codes is addressed in Appendix E and in Appendix F, respectively.

The following notation will be used. The coefficient of x^ℓ of a polynomial $b(x) \in F[x]$ will be denoted by b_ℓ. The leading coefficient (i.e., the coefficient of $x^{\deg b(x)}$) of a nonzero polynomial $b(x)$ will be denoted by $\operatorname{lcf} b(x)$, and we also define $\operatorname{lcf}(0) \triangleq 0$. We will use "mod" both as in $r(x) = b(x) \bmod m(x)$ (the remainder of a division) and as in $b(x) \equiv r(x) \bmod m(x)$ (a congruence modulo $m(x)$). We will also use "div" for polynomial division: if

$$a(x) = q(x)m(x) + r(x) \tag{3}$$

with $\deg r(x) < \deg m(x)$, then $q(x) = a(x) \operatorname{div} m(x)$ and $r(x) = a(x) \bmod m(x)$.

The Hamming weight of $e \in F^n$ will be denoted by $\mathrm{w_H}(e)$. For $x \in \mathbb{R}$, $\lceil x \rceil$ is the smallest integer not smaller than x, and $\lfloor x \rfloor$ is the largest integer not larger than x.

II. Basic Properties of the Partial-Inverse Problem

The partial-inverse problem as defined in Section I has the following properties.

1) The stated assumptions imply $\deg m(x) \geq 1$.
2) For $d = \deg m(x)$, the problem is solved by $\Lambda(x) = 1$. Smaller values of d will normally require a polynomial $\Lambda(x)$ of higher degree.
3) In the special case where $d = 0$, the solution is

$$\Lambda(x) \triangleq m(x) / \gcd\big(b(x), m(x)\big). \tag{4}$$

Proposition 1: The partial-inverse problem has always a solution. □

Proof: The existence of a solution for $d = 0$ implies the existence of a solution for any $d \geq 0$. □

Proposition 2: The solution $\Lambda(x)$ of a partial-inverse problem is unique up to a scale factor in F. □

Proof: Let $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ be two solutions of the problem, which implies $\deg \Lambda^{(1)}(x) = \deg \Lambda^{(2)}(x) \geq 0$. Define

$$r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \bmod m(x) \tag{5}$$

$$r^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \bmod m(x) \tag{6}$$

and consider

$$\Lambda(x) \triangleq \Big(\operatorname{lcf} \Lambda^{(2)}(x)\Big)\Lambda^{(1)}(x) - \Big(\operatorname{lcf} \Lambda^{(1)}(x)\Big)\Lambda^{(2)}(x). \tag{7}$$

Then

$$r(x) \triangleq b(x)\Lambda(x) \bmod m(x) \tag{8}$$

$$= \Big(\operatorname{lcf} \Lambda^{(2)}(x)\Big)r^{(1)}(x) - \Big(\operatorname{lcf} \Lambda^{(1)}(x)\Big)r^{(2)}(x) \tag{9}$$

by the natural ring homomorphism $F[x] \to F[x]/m(x)$. Clearly, (9) implies that $\Lambda(x)$ also satisfies (1). But (7) implies $\deg \Lambda(x) < \deg \Lambda^{(1)}(x)$, which is a contradiction unless

$\Lambda(x) = 0$. Thus $\Lambda(x) = 0$, which means that $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ are equal up to a scale factor. □

Proposition 3 (Degree Bound): If $\Lambda(x)$ solves the partial-inverse problem, then

$$\deg \Lambda(x) \le \deg m(x) - d. \tag{10}$$

□

Proof: The case $d = \deg m(x)$ is obvious. Otherwise, let $\nu \triangleq \deg m(x) - d > 0$, and consider the mapping

$$F^{\nu+1} \to F^{\nu} \tag{11}$$

given by

$$(\Lambda_0, \ldots, \Lambda_\nu) \mapsto \Lambda(x) \triangleq \Lambda_0 + \Lambda_1 x + \ldots + \Lambda_\nu x^\nu \tag{12}$$

$$\mapsto r(x) \triangleq b(x)\Lambda(x) \bmod m(x) \tag{13}$$

$$\mapsto (r_0, \ldots, r_{\deg m(x)-1}) \tag{14}$$

$$\mapsto (r_d, \ldots, r_{\deg m(x)-1}). \tag{15}$$

Clearly, this mapping is linear (over F), and its kernel is nontrivial by (11). But any nonzero element in the kernel corresponds (by (12)) to a nonzero polynomial $\Lambda(x)$ that satisfies (1). □

A sort of converse to Proposition 3 is given as Theorem 1 below. Another converse is Lemma 4 in Appendix C.

III. More About the Partial-Inverse Problem

The partial-inverse problem is not exhausted by Propositions 2 and 3. In this section, we address a number of additional aspects of this problem, which the reader may prefer to skip on first reading this paper.

A. Minimal Partial Inverses

The following notion is closely related to the partial-inverse problem and will play a prominent role in the proof of the algorithm of Section IV.

Definition (Minimal Partial Inverse): For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, a nonzero polynomial $\Lambda(x) \in F[x]$ is a *minimal partial inverse of* $b(x)$ *mod* $m(x)$ if every nonzero $\Lambda^{(1)}(x) \in F[x]$ with

$$\deg\Big(b(x)\Lambda^{(1)}(x) \bmod m(x)\Big) \le \deg\Big(b(x)\Lambda(x) \bmod m(x)\Big) \tag{16}$$

satisfies $\deg \Lambda^{(1)}(x) \ge \deg \Lambda(x)$. □

Both $\Lambda(x) = 1$ and $\Lambda(x)$ as in (4) are minimal partial inverses of any $b(x)$.

Proposition 4 (Minimal Partial Inverses Solve Partial-Inverse Problems): The solution $\Lambda(x)$ of the partial-inverse problem is a minimal partial inverse of $b(x) \bmod m(x)$.

Conversely, $\Lambda(x)$ as in (4) solves the partial-inverse problem with $d = 0$; every other minimal partial inverse $\Lambda(x)$ of $b(x) \bmod m(x)$ solves the partial-inverse problem with

$$d = \deg\Big(b(x)\Lambda(x) \bmod m(x)\Big) + 1. \tag{17}$$

□

Proof: Let $\Lambda(x)$ be the solution of the partial-inverse problem. If $\Lambda(x)$ is not a minimal partial inverse, then there exists $\Lambda^{(1)}(x)$ that satisfies (16) but $\deg \Lambda^{(1)}(x) < \deg \Lambda(x)$, which is a contradiction. The converse is obvious. □

Proposition 5 (Minimal Partial Inverses of the Same Degree Are Unique): For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, let $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ be two minimal partial inverses of $b(x)$ with $\deg \Lambda^{(1)}(x) = \deg \Lambda^{(2)}(x)$. Then $\Lambda^{(1)}(x) = \alpha\Lambda^{(2)}(x)$ for some nonzero $\alpha \in F$. □

Proof: Define $r^{(1)}(x)$ and $r^{(2)}(x)$ as in (5) and (6), respectively, and assume (without loss of generality) that $\deg r^{(1)}(x) \ge \deg r^{(2)}(x)$. By Proposition 4, $\Lambda^{(1)}(x)$ solves the partial-inverse problem with

$$d = \deg\Big(b(x)\Lambda^{(1)}(x) \bmod m(x)\Big) + 1. \tag{18}$$

But $\Lambda^{(2)}(x)$ solves the same partial inverse problem. Thus $\Lambda^{(1)}(x) = \alpha\Lambda^{(2)}(x)$ by Proposition 2. □

For any $b(x)$ and any $m(x)$, $\Lambda(x) = 1$ is a minimal partial inverse of the smallest degree, and $\Lambda(x)$ as in (4) is a minimal partial inverse of the largest degree. The algorithms of Section IV can be used to compute *all* minimal partial inverses (for fixed $b(x)$ and $m(x)$) in a single run (cf. Theorem 4) .

Theorem 1 (Degree Bound With a Converse): For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, a nonzero $\Lambda(x) \in F[x]$ is a minimal partial inverse of $b(x)$ if and only if both

$$\deg \Lambda(x) + \deg\Big(b(x)\Lambda(x) \bmod m(x)\Big) < \deg m(x) \tag{19}$$

and

$$\gcd\big(\Lambda(x), q(x)\big) = 1, \tag{20}$$

where $q(x) \triangleq b(x)\Lambda(x) \operatorname{div} m(x)$. □

In the special case where $q(x) = 0$, (20) requires $\Lambda(x)$ to be a nonzero constant.

Proof of Theorem 1: Let

$$r(x) \triangleq b(x)\Lambda(x) \bmod m(x) \tag{21}$$

$$= b(x)\Lambda(x) - q(x)m(x). \tag{22}$$

For the direct part, assume that $\Lambda(x)$ is a minimal partial inverse of $b(x)$. Then (19) is immediate from Propositions 4 and 3. As for (20), assume that $\gcd\big(\Lambda(x), q(x)\big) = g(x)$ with $\deg g(x) > 0$. From (22), we then have

$$r(x) = g(x)\big(\Lambda'(x)b(x) - q'(x)m(x)\big) \tag{23}$$

with $\Lambda'(x) \triangleq \Lambda(x)/g(x)$ and $q'(x) \triangleq q(x)/g(x)$. It follows that $g(x)$ divides $r(x)$, and thus

$$r'(x) = \Lambda'(x)b(x) - q'(x)m(x) \tag{24}$$

$$= \Lambda'(x)b(x) \bmod m(x) \tag{25}$$

with $r'(x) \triangleq r(x)/g(x)$. But $\deg \Lambda'(x) < \deg \Lambda(x)$ and $\deg r'(x) \le \deg r(x)$, which is impossible because $\Lambda(x)$ is a minimal partial inverse.

For the converse part, assume that some nonzero $\Lambda(x)$ satisfies both (19) and (20). Let $\Lambda^{(1)}(x)$ be any nonzero polynomial such that

$$r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \bmod m(x) \tag{26}$$

satisfies

$$\deg r^{(1)}(x) \le \deg r(x). \tag{27}$$

We have to prove that $\deg \Lambda^{(1)}(x) \ge \deg \Lambda(x)$. To this end, we now assume

$$\deg \Lambda^{(1)}(x) < \deg \Lambda(x) \tag{28}$$

and derive a contradiction. Let $q^{(1)}(x)$ be defined by

$$b(x)\Lambda^{(1)}(x) = q^{(1)}(x)m(x) + r^{(1)}(x) \tag{29}$$

and consider

$$b(x)\Lambda(x)\Lambda^{(1)}(x) = \Lambda^{(1)}(x)q(x)m(x) + \Lambda^{(1)}(x)r(x) \tag{30}$$
$$= \Lambda(x)q^{(1)}(x)m(x) + \Lambda(x)r^{(1)}(x) \tag{31}$$

from (22) and (29), respectively. But we also have

$$\deg \Lambda^{(1)}(x) + \deg r(x) < \deg \Lambda(x) + \deg r(x) \tag{32}$$
$$< \deg m(x), \tag{33}$$

where the second inequality is (19), as well as

$$\deg \Lambda(x) + \deg r^{(1)}(x) \le \deg \Lambda(x) + \deg r(x) \tag{34}$$
$$< \deg m(x). \tag{35}$$

With (33) and (35), we see that (30) and (31) imply both $\Lambda^{(1)}(x)r(x) = \Lambda(x)r^{(1)}(x)$ and

$$\Lambda^{(1)}(x)q(x) = \Lambda(x)q^{(1)}(x). \tag{36}$$

If $q(x) \neq 0$, then $\Lambda(x)$ divides $\Lambda^{(1)}(x)q(x)$, and (20) implies that $\Lambda(x)$ divides $\Lambda^{(1)}(x)$, which contradicts (28). If $q(x) = 0$, then (20) requires $\Lambda(x)$ to be a nonzero constant, which again contradicts (28). □

The following lemma is a (stronger) counterpart of [4, Th. 1], but the proof is entirely different.

Lemma 1 (Degree Change Lemma): For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, let $\Lambda^{(2)}(x)$ be a minimal partial inverse of $b(x)$ and let

$$r^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \bmod m(x). \tag{37}$$

Let $\Lambda^{(1)}(x) \in F[x]$ be a nonzero polynomial of the smallest degree such that

$$\deg\big(b(x)\Lambda^{(1)}(x) \bmod m(x)\big) < \deg r^{(2)}(x). \tag{38}$$

Then

$$\deg \Lambda^{(1)}(x) = \deg m(x) - \deg r^{(2)}(x). \tag{39}$$

□

Proof: From Proposition 3, we have

$$\deg \Lambda^{(1)}(x) \le \deg m(x) - \deg r^{(2)}(x). \tag{40}$$

For the converse, assume that $\Lambda^{(1)}(x)$ is a nonzero polynomial (of any degree) that satisfies (38), i.e., the degree of

$$r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \bmod m(x) \tag{41}$$

satisfies

$$\deg r^{(1)}(x) < \deg r^{(2)}(x). \tag{42}$$

Multiplying (37) by $\Lambda^{(1)}(x)$ and (41) by $\Lambda^{(2)}(x)$ yields

$$\Lambda^{(1)}(x)r^{(2)}(x) \equiv \Lambda^{(2)}(x)r^{(1)}(x) \mod m(x). \tag{43}$$

Note that

$$\deg \Lambda^{(1)}(x) \ge \deg \Lambda^{(2)}(x) \tag{44}$$

because $\Lambda^{(2)}(x)$ is a minimal partial inverse and (38). We thus have

$$\deg \Lambda^{(1)}(x) + \deg r^{(2)}(x) > \deg \Lambda^{(2)}(x) + \deg r^{(1)}(x). \tag{45}$$

If

$$\deg \Lambda^{(1)}(x) < \deg m(x) - \deg r^{(2)}(x), \tag{46}$$

then (43) reduces to

$$\Lambda^{(1)}(x)r^{(2)}(x) = \Lambda^{(2)}(x)r^{(1)}(x), \tag{47}$$

which contradicts (45). Thus (46) is not tenable and

$$\deg \Lambda^{(1)}(x) \ge \deg m(x) - \deg r^{(2)}(x). \tag{48}$$

□

Corollary 1: For $b(x)$, $m(x)$, $\Lambda^{(2)}(x)$, and $r^{(2)}(x)$ as in Lemma 1, let $\Lambda^{(1)}(x)$ be a nonzero polynomial (of any degree) such that (38) holds and

$$\deg \Lambda^{(1)}(x) = \deg m(x) - \deg r^{(2)}(x). \tag{49}$$

Then $\Lambda^{(1)}(x)$ is (also) a minimal partial inverse of $b(x)$. □

Lemma 1 and Corollary 1 will be used in Section IV to prove the correctness of the partial-inverse algorithm, and they will also be used in Appendix B.

B. Irrelevant Coefficients

Proposition 6 (Irrelevant Coefficients): In the partial-inverse problem, coefficients b_ℓ of $b(x)$ with

$$\ell < 2d - \deg m(x) \tag{50}$$

and coefficients m_s of $m(x)$ with

$$s \le 2d - \deg m(x) \tag{51}$$

have no effect on the solution $\Lambda(x)$. □

Proof: From (50) and (10), we obtain

$$\ell + \deg \Lambda(x) < d, \tag{52}$$

which proves the first claim. As for the second claim, we begin by writing

$$b(x)\Lambda(x) \bmod m(x) = b(x)\Lambda(x) - m(x)q(x) \tag{53}$$

for some $q(x) \in F[x]$ with

$$\deg q(x) < \deg \Lambda(x). \tag{54}$$

(If $q(x) \neq 0$, (54) follows from considering the leading coefficient of the right-hand side of (53) with $\deg b(x) < \deg m(x)$). From (51), (54), and (10), we then obtain

$$s + \deg q(x) < d. \tag{55}$$

The second claim then follows from (53) and (55). □

Irrelevant coefficients according to Proposition 6 may be set to zero without affecting the solution $\Lambda(x)$. In fact, such coefficients can be stripped off as follows.

Proposition 7 (Reduced Partial-Inverse Problem): Consider a partial-inverse problem with

$$s \triangleq 2d - \deg m(x) > 0. \tag{56}$$

Define the polynomials $\tilde{b}(x)$ and $\tilde{m}(x)$ with

$$\tilde{b}_\ell \triangleq b_{\ell+s} \tag{57}$$

and

$$\tilde{m}_\ell \triangleq m_{\ell+s} \tag{58}$$

for $\ell \geq 0$. Then the modified partial-inverse problem with $b(x)$, $m(x)$, and d replaced by $\tilde{b}(x)$, $\tilde{m}(x)$, and $\tilde{d} \triangleq d - s$, respectively, has the same solution $\Lambda(x)$ as the original partial-inverse problem. In addition, we have

$$b(x)\Lambda(x) \operatorname{div} m(x) = \tilde{b}(x)\Lambda(x) \operatorname{div} \tilde{m}(x). \tag{59}$$

□

Note that the reduced partial-inverse problem satisfies

$$\deg \tilde{m}(x) = 2\tilde{d}. \tag{60}$$

Proof of Proposition 7: Consider an auxiliary partial-inverse problem with $b(x)$ replaced by $x^s\tilde{b}(x)$ and $m(x)$ replaced by $x^s\tilde{m}(x)$ (and d unchanged). This auxiliary problem has the same solution as the original problem by Proposition 6. The equivalence of this auxiliary problem with the modified problem is obvious from (53). □

C. Monomialized Partial-Inverse Problem

The partial-inverse problem with general $m(x)$ (as stated in Section I) can be transformed into another partial-inverse problem where (1) is replaced by

$$\deg\left(\tilde{b}(x)\Lambda(x) \bmod x^{2\tau}\right) < \tau \tag{61}$$

with $\tau \triangleq \deg m(x) - d$ and with $\tilde{b}(x)$ as defined below. The precise statement is given as Theorem 2 below. Note that we need the additional condition

$$d < \deg m(x). \tag{62}$$

Note also that the transformed partial-inverse problem (61) is reduced in the sense of Proposition 7.

The polynomial $\tilde{b}(x)$ in (61) is defined as follows. Let

$$n \triangleq \deg m(x) \tag{63}$$

and thus $\tau = n - d > 0$, and define

$$\overline{b}(x) \triangleq x^{n-1}b(x^{-1}). \tag{64}$$

Moreover, define $\overline{m}(x) \triangleq x^n m(x^{-1})$, and let $w(x)$ be the inverse of $\overline{m}(x) \bmod x^{2\tau}$ in $F[x]/x^{2\tau}$; this inverse exists because $\overline{m}(0) \neq 0$, which implies that $\overline{m}(x)$ is relatively prime to $x^{2\tau}$.

Further, let

$$a(x) \triangleq \overline{b}(x)w(x) \tag{65}$$

and

$$s(x) \triangleq a(x) \bmod x^{2\tau} = \sum_{\ell=0}^{2\tau-1} a_\ell\, x^\ell \tag{66}$$

and finally

$$\tilde{b}(x) \triangleq x^{2\tau-1}s(x^{-1}). \tag{67}$$

Theorem 2 (Monomialized Partial-Inverse Problem): Consider the partial-inverse problem as stated in Section I with the additional condition (62), and let $\tau \triangleq \deg m(x) - d$. Then the modified partial-inverse problem where $b(x)$, $m(x)$, and d are replaced by $\tilde{b}(x)$ (as defined above), $x^{2\tau}$, and τ, respectively, has the same solution $\Lambda(x)$ as the original partial-inverse problem. In addition, we have

$$b(x)\Lambda(x) \operatorname{div} m(x) = \tilde{b}(x)\Lambda(x) \operatorname{div} x^{2\tau}. \tag{68}$$

□

The computation of $\tilde{b}(x)$ requires the computation of the polynomial $w(x)$ (= the inverse of $\overline{m}(x) \bmod x^{2\tau}$ in $F[x]/x^{2\tau}$, as defined above), which can be computed either by the extended Euclidean algorithm or by the algorithms of Section IV.

Proof of Theorem 2: Consider the original partial-inverse problem and let $\Lambda(x)$ be its solution (which is unique up to a nonzero scale factor). Let

$$r(x) \triangleq b(x)\Lambda(x) \bmod m(x), \tag{69}$$

where $\deg r(x) < d$. We then write

$$r(x) = b(x)\Lambda(x) - q(x)m(x) \tag{70}$$

for some (unique) $q(x)$ with

$$\deg q(x) < \deg \Lambda(x) \leq \tau, \tag{71}$$

where the second inequality follows from Proposition 3. Note also that $\gcd\big(\Lambda(x), q(x)\big) = 1$ by Theorem 1 and Proposition 4. Now let

$$\overline{\Lambda}(x) \triangleq x^\tau \Lambda(x^{-1}) \tag{72}$$

$$\overline{q}(x) \triangleq x^{\tau-1} q(x^{-1}) \tag{73}$$

$$\overline{r}(x) \triangleq x^{d-1} r(x^{-1}). \tag{74}$$

Noting that $2\tau + d - 1 = n + \tau - 1$, reversing (70) yields first

$$x^{2\tau+d-1}r(x^{-1}) = x^{n+\tau-1}\big(b(x^{-1})\Lambda(x^{-1}) - q(x^{-1})m(x^{-1})\big) \tag{75}$$

and then

$$x^{2\tau}\overline{r}(x) = \overline{b}(x)\overline{\Lambda}(x) - \overline{q}(x)\overline{m}(x). \tag{76}$$

We thus obtain

$$\overline{b}(x)\overline{\Lambda}(x) \equiv \overline{q}(x)\overline{m}(x) \mod x^{2\tau} \tag{77}$$

and further

$$w(x)\overline{b}(x)\overline{\Lambda}(x) \equiv \overline{q}(x) \mod x^{2\tau} \tag{78}$$

where $w(x)$ (as defined above) is the inverse of $\overline{m}(x) \bmod x^{2\tau}$. With (66), we then have

$$s(x)\overline{\Lambda}(x) \equiv \overline{q}(x) \mod x^{2\tau}. \tag{79}$$

Note that (71) implies both $\deg \overline{\Lambda}(x) \le \tau$ and $\deg \overline{q}(x) < \tau$. We now write (79) as

$$s(x)\overline{\Lambda}(x) = \overline{p}(x)x^{2\tau} + \overline{q}(x) \tag{80}$$

for some (unique) $\overline{p}(x)$ with $\deg \overline{p}(x) < \deg \overline{\Lambda}(x) \le \tau$, and let

$$p(x) \triangleq x^{\tau-1}\overline{p}(x^{-1}). \tag{81}$$

By substituting x^{-1} for x in (80) and multiplying both sides by $x^{3\tau-1}$, we obtain

$$\tilde{b}(x)\Lambda(x) = x^{2\tau}q(x) + p(x), \tag{82}$$

from which we have

$$\deg\Big(\tilde{b}(x)\Lambda(x) \bmod x^{2\tau}\Big) < \tau. \tag{83}$$

It then follows from Theorem 1 that $\Lambda(x)$ is a minimal partial inverse of $\tilde{b}(x)$ (with respect to $x^{2\tau}$).

We still have to show that $\Lambda(x)$ is the solution of the partial-inverse problem (61). Let $\Lambda^{(1)}(x)$ be any nonzero polynomial such that

$$r^{(1)}(x) \triangleq \tilde{b}(x)\Lambda^{(1)}(x) \bmod x^{2\tau} \tag{84}$$

satisfies $\deg r^{(1)}(x) < \tau$. Multiplying $p(x) = \tilde{b}(x)\Lambda(x) \bmod x^{2\tau}$ by $\Lambda^{(1)}(x)$ and (84) by $\Lambda(x)$ yields

$$p(x)\Lambda^{(1)}(x) \equiv r^{(1)}(x)\Lambda(x) \mod x^{2\tau}. \tag{85}$$

Note that $\deg r^{(1)}(x)\Lambda(x) < 2\tau$ by (71). If $\deg \Lambda^{(1)}(x) < \deg \Lambda(x)$, then (85) becomes

$$p(x)\Lambda^{(1)}(x) = r^{(1)}(x)\Lambda(x) \tag{86}$$

and $\deg r^{(1)}(x) < \deg p(x)$, contradicting the fact that $\Lambda(x)$ is a minimal partial inverse of $\tilde{b}(x)$.

Finally, we note that (68) is obvious from (82). □

IV. Partial-Inverse Algorithms

We now consider algorithms to solve the partial-inverse problem as stated in Section I. We give both a (new) basic algorithm, which resembles the Berlekamp–Massey algorithm, and two variations of it; the first variation is entirely new and the second variation may be viewed as a version of the Euclidean algorithm.

A. Basic Algorithm (Reverse Berlekamp–Massey Algorithm)

The basic algorithm is stated as *Algorithm 1* in the framed box. Note that lines 14–16 simply swap $\Lambda^{(1)}(x)$ with $\Lambda^{(2)}(x)$, d_1 with d_2, and κ_1 with κ_2. The only actual computations are in lines 7 and 8.

The heart of the algorithm is line 7, which is explained by the following lemma.

Lemma 2 (Remainder Decreasing Lemma): Let $m(x)$ be a polynomial over F with $\deg m(x) \ge 1$. For further polynomials $b(x), \Lambda^{(1)}(x), \Lambda^{(2)}(x) \in F[x]$, let

$$r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \bmod m(x), \tag{87}$$
$$r^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \bmod m(x), \tag{88}$$

Algorithm 1: Basic Partial-Inverse Algorithm
(reverse Berlekamp–Massey algorithm)
Input: $b(x)$, $m(x)$, and d as in the problem statement.
Output: $\Lambda(x)$ as in the problem statement.

```
1   if deg b(x) < d begin
2       return Λ(x) := 1
3   end
4   Λ^(1)(x) := 0,  d_1 := deg m(x),  κ_1 := lcf m(x)
5   Λ^(2)(x) := 1,  d_2 := deg b(x),  κ_2 := lcf b(x)
6   loop begin
7       Λ^(1)(x) := κ_2 Λ^(1)(x) − κ_1 x^(d_1−d_2) Λ^(2)(x)
8       d_1 := deg(b(x)Λ^(1)(x) mod m(x))
9       if d_1 < d begin
10          return Λ(x) := Λ^(1)(x)
11      end
12      κ_1 := lcf(b(x)Λ^(1)(x) mod m(x))
13      if d_1 < d_2 begin
14          (Λ^(1)(x), Λ^(2)(x)) := (Λ^(2)(x), Λ^(1)(x))
15          (d_1, d_2) := (d_2, d_1)
16          (κ_1, κ_2) := (κ_2, κ_1)
17      end
18  end
```

See also the refinements (Algorithms 1.A and 1.B) below.

Algorithm 1.A: Lines 8–12 of Algorithm 1 can be implemented as follows:

```
21      repeat
22          d_1 := d_1 − 1
23          if d_1 < d begin
24              return Λ(x) := Λ^(1)(x)
25          end
26          κ_1 := coefficient of x^(d_1) in
                       b(x)Λ^(1)(x) mod m(x)
27      until κ_1 ≠ 0
```

Algorithm 1.B: In the special case where $m(x) = x^{\nu}$, line 26 of Algorithm 1.A amounts to

31 $\quad \kappa_1 := b_{d_1}\Lambda_0^{(1)} + b_{d_1-1}\Lambda_1^{(1)} + \ldots + b_{d_1-\tau}\Lambda_\tau^{(1)}$

with $\tau \triangleq \deg \Lambda^{(1)}(x)$ and where $b_\ell \triangleq 0$ for $\ell < 0$.
In another special case where $m(x) = x^n - 1$, line 26 becomes

51 $\quad \kappa_1 := b_{d_1}\Lambda_0^{(1)} + b_{[d_1-1]}\Lambda_1^{(1)} + \ldots + b_{[d_1-\tau]}\Lambda_\tau^{(1)}$

with $b_{[\ell]} \triangleq b_{\ell \bmod n}$.

$d_1 \triangleq \deg r^{(1)}(x)$, $\kappa_1 \triangleq \operatorname{lcf} r^{(1)}(x)$, $d_2 \triangleq \deg r^{(2)}(x)$, $\kappa_2 \triangleq \operatorname{lcf} r^{(2)}(x)$, and assume $d_1 \ge d_2 \ge 0$. Then

$$\Lambda(x) \triangleq \kappa_2\Lambda^{(1)}(x) - \kappa_1 x^{d_1-d_2}\Lambda^{(2)}(x) \tag{89}$$

satisfies

$$\deg\Big(b(x)\Lambda(x) \bmod m(x)\Big) < d_1. \tag{90}$$

□

Proof: From (89), we obtain

$$r(x) \triangleq b(x)\Lambda(x) \bmod m(x) \tag{91}$$

$$= \kappa_2 r^{(1)}(x) - \kappa_1 x^{d_1-d_2} r^{(2)}(x) \tag{92}$$

by the natural ring homomorphism $F[x] \to F[x]/m(x)$. It is then obvious from (92) that $\deg r(x) < \deg r^{(1)}(x) = d_1$. □

In consequence, the value of d_1 is reduced in every execution of line 8 (cf. Section IV-F).

Note that lines 8 and 12 do not require the computation of the entire polynomial $b(x)\Lambda^{(1)}(x) \bmod m(x)$. Indeed, lines 8–12 can be replaced by Algorithm 1.A (see box).

The specialization to $m(x) = x^v$ and to $m(x) = x^n - 1$ (as in (109), see Section V) is given in Algorithm 1.B (see box). In these special cases, Algorithm 1 looks very much like, and is as efficient as, the Berlekamp–Massey algorithm [4] (but the polynomial $b(x)$ is processed in the reverse order).

Theorem 3 (Partial-Inverse Algorithm): Algorithm 1 returns the solution of the partial inverse problem. □

The proof will be given in Section IV-F. More generally, we have

Theorem 4 (All Minimal Partial Inverses): For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, run the algorithm with $d = 0$. Then $\Lambda^{(1)}(x)$ between lines 13 and 14 runs through all minimal partial inverses of $b(x)$ of degree larger than zero except the final output $\Lambda(x)$ as in (4). □

The proof will be given in Section IV-F. (In consequence, the algorithm can also be used to compute all Padé approximants of $b(x)$, cf. Appendix A.)

The complexity of the algorithm is determined by

Theorem 5 (Complexity): The number N_{it} of executions of line 26 in Algorithm 1.A is bounded by

$$N_{\text{it}} \le \deg m(x) - d + \deg \Lambda(x) \tag{93}$$

$$\le 2(\deg m(x) - d), \tag{94}$$

where $\Lambda(x)$ is the solution of the partial-inverse problem. □

The bound (93) is proved in Section IV-F. The bound (94) then follows from Proposition 3.

In the special case of Algorithm 1.B, we thus obtain the complexity $\mathcal{O}\big((v-d)^2\big)$. In the general case, the same complexity is obtained with the algorithm of the next section.

B. Quotient Saving Algorithm

Algorithm 2 (see box) is a variation of Algorithm 1 that achieves a generalization of Algorithm 1.B to general $m(x)$. To this end, we store and update also the quotients $Q^{(1)}(x)$ and $Q^{(2)}(x)$ defined by

$$b(x)\Lambda^{(\ell)}(x) = Q^{(\ell)}(x)m(x) + r^{(\ell)}(x) \tag{95}$$

with $r^{(\ell)}(x) \triangleq b(x)\Lambda^{(\ell)}(x) \bmod m(x)$. The coefficient of x^{d_1} of $r^{(1)}(x)$ (line 26) can then be computed as

$$\kappa_1 := \sum_{\ell=0}^{\tau} b_{d_1-\ell}\Lambda_\ell^{(1)} - \sum_{\ell=0}^{v} m_{d_1-\ell} Q_\ell^{(1)} \tag{96}$$

Algorithm 2: Quotient Saving Partial-Inverse Algorithm

Input: $b(x)$, $m(x)$, and d as in the problem statement.
Output: $\Lambda(x)$ as in the problem statement.

```
1   if deg b(x) < d begin
2       return Λ(x) := 1
3   end
4   Λ^(1)(x) := 0,  d_1 := deg m(x),  κ_1 := lcf m(x)
5   Λ^(2)(x) := 1,  d_2 := deg b(x),  κ_2 := lcf b(x)
6   Q^(1)(x) := −1,  Q^(2)(x) := 0
7   loop begin
8       Λ^(1)(x) := κ_2 Λ^(1)(x) − κ_1 x^(d_1−d_2) Λ^(2)(x)
9       Q^(1)(x) := κ_2 Q^(1)(x) − κ_1 x^(d_1−d_2) Q^(2)(x)
        ---------------------------------------------------
10      repeat
11          d_1 := d_1 − 1
12          if d_1 < d begin
13              return Λ(x) := Λ^(1)(x)
14          end
15          κ_1 := Σ_{ℓ=0}^{τ} b_{d_1−ℓ} Λ_ℓ^(1) − Σ_{ℓ=0}^{v} m_{d_1−ℓ} Q_ℓ^(1)
16      until κ_1 ≠ 0
        ---------------------------------------------------
17      if d_1 < d_2 begin
18          (Λ^(1)(x), Λ^(2)(x)) := (Λ^(2)(x), Λ^(1)(x))
19          (Q^(1)(x), Q^(2)(x)) := (Q^(2)(x), Q^(1)(x))
20          (d_1, d_2) := (d_2, d_1)
21          (κ_1, κ_2) := (κ_2, κ_1)
22      end
23  end
```

with $\tau \triangleq \deg \Lambda^{(1)}(x)$ and $v \triangleq \deg Q^{(1)}(x)$, and where both $b_\ell \triangleq 0$ and $m_\ell \triangleq 0$ for $\ell < 0$.

All other quantities in the algorithm remain unchanged. From (95), we have

$$\deg Q^{(\ell)}(x) < \deg \Lambda^{(\ell)}(x). \tag{97}$$

The complexity of the algorithm is $\mathcal{O}\big((v-d)^2\big)$ with $v \triangleq \deg m(x)$.

C. Remainder Saving Algorithm

Another variation of the basic partial-inverse algorithm is Algorithm 3 (see box), where we store and update the remainders $r^{(1)}(x)$ and $r^{(2)}(x)$ in (95). In consequence, the computation of line 26 is unnecessary. All other quantities in the algorithm remain unchanged. The complexity of the algorithm is $\mathcal{O}\big(v(v-d)\big)$ with $v \triangleq \deg m(x)$.

D. Recovering the Euclidean Algorithm

Algorithm 3 may be viewed as a version of the extended Euclidean algorithm. To see this, the extended Euclidean algorithm is stated as Algorithm 4 (see box). Lines 8 and 9 of Algorithm 3 may be viewed as implementating the polynomial operations in lines 4–6 of Algorithm 4 (up to a scale factor).

As stated, Algorithm 4 solves the partial-inverse problem. The standard gcd algorithm is recovered by running the

Algorithm 3: Remainder Saving Partial-Inverse Alg.
(a variation of a gcd algorithm)

Input: $b(x)$, $m(x)$, and d as in the problem statement.
Output: $\Lambda(x)$ as in the problem statement.

```
1   if deg b(x) < d begin
2       return Λ(x) := 1
3   end
4   Λ^(1)(x) := 0,  d_1 := deg m(x),  κ_1 := lcf m(x)
5   Λ^(2)(x) := 1,  d_2 := deg b(x),  κ_2 := lcf b(x)
6   r^(1)(x) := m(x),  r^(2)(x) := b(x)
7   loop begin
8       Λ^(1)(x) := κ_2 Λ^(1)(x) − κ_1 x^(d_1−d_2) Λ^(2)(x)
9       r^(1)(x) := κ_2 r^(1)(x) − κ_1 x^(d_1−d_2) r^(2)(x)
        ---------------------------------------------------
10      d_1 := deg r^(1)(x)
11      if d_1 < d begin
12          return Λ(x) := Λ^(1)(x)
13      end
14      κ_1 := lcf r^(1)(x)
        ---------------------------------------------------
15      if d_1 < d_2 begin
16          (Λ^(1)(x), Λ^(2)(x)) := (Λ^(2)(x), Λ^(1)(x))
17          (r^(1)(x), r^(2)(x)) := (r^(2)(x), r^(1)(x))
18          (d_1, d_2) := (d_2, d_1)
19          (κ_1, κ_2) := (κ_2, κ_1)
20      end
21  end
```

Algorithm 4: Extended Euclidean Algorithm
(with notation corresponding to Algorithm 3)

Input: $b(x)$, $m(x)$, and d as in the problem statement.
Output: $\Lambda(x)$ as in the problem statement.

```
1   r^(1)(x) = m(x), Λ^(1)(x) = 0, Q^(1)(x) = −1
2   r^(2)(x) = b(x), Λ^(2)(x) = 1, Q^(2)(x) = 0
3   while deg r^(2)(x) ≥ d begin
4       q(x) := r^(1)(x) div r^(2)(x)
5       r^(1)(x) := r^(1)(x) − q(x) r^(2)(x)
6       Λ^(1)(x) := Λ^(1)(x) − q(x) Λ^(2)(x)
7       Q^(1)(x) := Q^(1)(x) − q(x) Q^(2)(x)
        ---------------------------------------
8       (r^(1), r^(2)) := (r^(2), r^(1))
9       (Λ^(1), Λ^(2)) := (Λ^(2), Λ^(1))
10      (Q^(1), Q^(2)) := (Q^(2), Q^(1))
11  end
12  return Λ(x) := Λ^(2)(x)
```

algorithm with $d = 0$; when the algorithm stops, we have

$$r^{(1)}(x) = \gcd\big(m(x), b(x)\big). \tag{98}$$

The polynomials $Q^{(1)}(x)$ and $Q^{(2)}(x)$ in Algorithm 4 are not actually needed for the partial-inverse problem (but they do correspond to the polynomials $Q^{(1)}(x)$ and $Q^{(2)}(x)$ in Algorithm 2).

Note that, in contrast to Algorithm 4, Algorithm 3 requires no division (neither polynomial nor scalar).

As mentioned in the Introduction, McEliece and Shearer [9] showed that the Euclidean algorithm (Algorithm 4) solves a modified Padé approximation problem that is equivalent to the partial-inverse problem, see also [10, Sec. 5.7] and Appendix A. With hindsight, Algorithms 1–2 can thus be derived also from the Euclidean algorithm.

E. An Example

We illustrate Algorithms 1–3 with an example from [9]. Let $F_3 \triangleq \{0, 1, -1\}$ be the field of integers modulo 3, $b(x) \triangleq -x^6 + x^5 - x^3 + x^2 + x + 1$, and $m(x) \triangleq x^7$. For simplicity, we set $d = 6$.

We first use Algorithm 1, with lines 8–12 as in Algorithm 1.A. In line 4, we have $\Lambda^{(1)}(x) = 0$, $d_1 = 7$, and $\kappa_1 = 1$; in line 5, we have $\Lambda^{(2)}(x) = 1$, $d_2 = 6$, and $\kappa_2 = -1$.

1) In the first loop iteration, we obtain $\Lambda^{(1)}(x) = -x$ after line 7, $d_1 = 6$ after line 22, and $\kappa_1 = -b_5 = -1$ after line 26. Note that we compute line 26 via line 31; note also that $\Lambda^{(2)}(x)$, d_2, and κ_2 remain unchanged, i.e., $\Lambda^{(2)}(x) = 1$, $d_2 = 6$, and $\kappa_2 = -1$.
2) In the second loop iteration, we obtain $\Lambda^{(1)}(x) = x + 1$ after line 7, and $d_1 = 5$ after line 22. The condition in line 23 then holds, and Algorithm 1 returns $\Lambda(x) = x+1$ in line 24.

We next use Algorithm 2. Both lines 4 and 5 are the same as in Algorithm 1; in line 6, we have $Q^{(1)}(x) = -1$ and $Q^{(2)}(x) = 0$.

1) In the first loop iteration, we have $\Lambda^{(1)}(x) = -x$ after line 8, $Q^{(1)}(x) = 1$ after line 9, $d_1 = 6$ after line 11, and $\kappa_1 = -b_5 - m_6 = -b_5 = -1$ after line 15. Note that $\Lambda^{(2)}(x)$, d_2, κ_2, and $Q^{(2)}(x)$ remain unchanged, i.e., $\Lambda^{(2)}(x) = 1$, $d_2 = 6$, and $\kappa_2 = -1$, and $Q^{(2)}(x) = 0$.
2) In the second loop iteration, we obtain $\Lambda^{(1)}(x) = x + 1$ after line 8, $Q^{(1)}(x) = -1$ after line 9, $d_1 = 5$ after line 11. The condition in line 12 then holds, and Algorithm 2 returns $\Lambda(x) = x + 1$ in line 13.

Finally, we use Algorithm 3. Both lines 4 and 5 are the same as in Algorithm 1; in line 6, we have $r^{(1)}(x) = x^7$ and $r^{(2)}(x) = -x^6 + x^5 - x^3 + x^2 + x + 1$.

1) In the first loop iteration, we have $\Lambda^{(1)}(x) = -x$ after line 8, $r^{(1)}(x) = -x^6 + x^4 - x^3 - x^2 - x$ after line 9, $d_1 = 6$ after line 10, and $\kappa_1 = -1$ after line 14. Note that $\Lambda^{(2)}(x)$, d_2, κ_2, and $r^{(2)}(x)$ remain unchanged, i.e., $\Lambda^{(2)}(x) = 1$, $d_2 = 6$, and $\kappa_2 = -1$, and $r^{(2)}(x) = -x^6 + x^5 - x^3 + x^2 + x + 1$.
2) In the second loop iteration, we obtain $\Lambda^{(1)}(x) = x + 1$ after line 8, $r^{(1)}(x) = x^5 - x^4 - x^2 - x + 1$ after line 9, $d_1 = 5$ after line 10. The condition in line 11 then holds, and Algorithm 3 returns $\Lambda(x) = x + 1$ in line 12.

F. Proofs

We now proceed to prove Theorems 3–5. (Algorithms 2 and 3 are easy modifications of Algorithm 1 and do not require an extra proof.)

Proof of Theorems 3 and 4: We begin by restating Algorithm 1 with added assertions as Algorithm 5 (see box).

Algorithm 5: Basic Partial-Inverse Algorithm Restated

1 **if** $\deg b(x) < d$ **begin**
2 **return** $\Lambda(x) := 1$
3 **end**
4 $\Lambda^{(1)}(x) := 0,\ d_1 := \deg m(x),\ \kappa_1 := \mathrm{lcf}\, m(x)$
5 $\Lambda^{(2)}(x) := 1,\ d_2 := \deg b(x),\ \kappa_2 := \mathrm{lcf}\, b(x)$
6 **loop begin**

Assertions:
$d_1 > d_2 \geq d$ (A.1)
$\deg \Lambda^{(2)}(x) = \deg m(x) - d_1$ (A.2)
$> \deg \Lambda^{(1)}(x)$ (A.3)
$\Lambda^{(2)}(x)$ is a minimal partial inverse (A.4)

7 **repeat**
8 $\Lambda^{(1)}(x) := \kappa_2 \Lambda^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Lambda^{(2)}(x)$

Assertions:
$\deg(b(x)\Lambda^{(1)}(x) \bmod m(x)) < d_1$ (A.5)
$\deg \Lambda^{(1)}(x) = \deg m(x) - d_2$ (A.6)
$> \deg \Lambda^{(2)}(x)$ (A.7)

9 $d_1 := \deg\big(b(x)\Lambda^{(1)}(x) \bmod m(x)\big)$
10 **if** $d_1 < d$ **begin**

Assertion:
$\Lambda^{(1)}(x)$ is a min. partial inverse (A.8)

11 **return** $\Lambda(x) := \Lambda^{(1)}(x)$
12 **end**
13 $\kappa_1 := \mathrm{lcf}\,\big(b(x)\Lambda^{(1)}(x) \bmod m(x)\big)$
14 **until** $d_1 < d_2$

Assertion:
$\Lambda^{(1)}(x)$ is a minimal partial inverse (A.9)

15 $(\Lambda^{(1)}(x), \Lambda^{(2)}(x)) := (\Lambda^{(2)}(x), \Lambda^{(1)}(x))$
16 $(d_1, d_2) := (d_2, d_1)$
17 $(\kappa_1, \kappa_2) := (\kappa_2, \kappa_1)$
18 **end**

Note the added inner **repeat** loop (lines 7–14), which does not change the algorithm but helps with the proof.

Throughout the algorithm (except at the very beginning, before the first execution of lines 9 and 13), d_1, d_2, κ_1, and κ_2 are defined as in Lemma 2, i.e., $d_1 = \deg r^{(1)}(x)$, $\kappa_1 = \mathrm{lcf}\, r^{(1)}(x)$, $d_2 = \deg r^{(2)}(x)$, and $\kappa_2 = \mathrm{lcf}\, r^{(2)}(x)$ for $r^{(1)}(x)$ and $r^{(2)}(x)$ as in (87) and (88).

Assertions (A.1)–(A.4) are easily verified, both from the initialization and from (A.6), (A.7), and (A.9).

As for (A.5), after the very first execution of line 8, we still have $d_1 = \deg m(x)$ (from line 4), which makes (A.5) obvious. For all later executions of line 8, (A.5) follows from Lemma 2.

As for (A.6) and (A.7), we note that line 8 changes the degree of $\Lambda^{(1)}(x)$ as follows:

- Upon entering the **repeat** loop, line 8 increases the degree of $\Lambda^{(1)}$ to

$$\deg \Lambda^{(2)}(x) + d_1 - d_2 = \deg m(x) - d_2 \quad (99)$$
$$> \deg \Lambda^{(2)}(x), \quad (100)$$

  which follows from (A.1)–(A.3).

- Subsequent executions of line 8 without leaving the **repeat** loop (i.e., without executing lines 15–17) do not change the degree of $\Lambda^{(1)}(x)$. (This follows from the fact that d_1 is smaller than in the first execution while $\Lambda^{(2)}(x)$, d_2, and $\kappa_2 \neq 0$ remain unchanged.)

Assertion (A.9) follows from Corollary 1, which applies because $d_1 < d_2$ and (A.6). Because of (A.1), the same argument applies also to (A.8).

We have thus proved Theorem 3, and Theorem 4 is obvious from this discussion. □

Proof of Theorem 5: We refer to Algorithm 1 with lines 8–12 replaced by Algorithm 1.A, and we assume $\deg b(x) \geq d$.

For the proof, we augment the algorithm with the two additional variables $N_{\mathrm{it}}^{(1)}$ and $N_{\mathrm{it}}^{(2)}$, which are both initialized to zero and swapped whenever $\Lambda^{(1)}$ and $\Lambda^{(2)}$ are swapped. Moreover, $N_{\mathrm{it}}^{(1)}$ is incremented in every execution of line 26. When the algorithm stops, we have

$$N_{\mathrm{it}} = N_{\mathrm{it}}^{(1)} + N_{\mathrm{it}}^{(2)} \quad (101)$$

as well as

$$N_{\mathrm{it}}^{(1)} \leq \deg m(x) - d \quad (102)$$

and

$$N_{\mathrm{it}}^{(2)} \leq \deg m(x) - \tilde{d}_2 \quad (103)$$
$$= \deg \Lambda^{(1)}(x), \quad (104)$$

where $\tilde{d}_2$ is the value of d_1 before the last swap, and where the second step follows from Lemma 1. □

G. Concluding Remarks for Section IV

We have proposed three partial-inverse algorithms: a new basic version (the reverse Berlekamp–Massey algorithm) and two variations of it. The first variation (the quotient saving algorithm) is entirely new while the second variation (the remainder saving algorithm) may be viewed as a variation of the Euclidean gcd algorithm. The relative attractivity of these algorithms depends on the particulars of the application.

The easy transition between Algorithm 1 and Algorithm 3 is a new contribution to the substantial body of literature connecting the Euclidean algorithm with the Berlekamp–Massey algorithm [14]–[19].

Finally, we remark that any of the proposed algorithms can be made asymptotically faster by standard techniques for asymptotically fast polynomial multiplication and division as in [10]. However, such techniques are outside the scope of this paper.

V. Decoding Reed–Solomon Codes

Decoding Reed–Solomon codes (up to half the minimum distance) can be reduced in several ways to the partial-inverse problem of Section I. (The extension to decoding interleaved Reed–Solomon codes beyond half the minimum distance as in [25] and [26] is beyond the scope of this paper.) We will also point out several interpolation methods, some of which are (partly or entirely) new as well.

A. Reed–Solomon Codes

Let F be a finite field and let $\beta_0, \ldots, \beta_{n-1}$ be n different elements of F. Let

$$m(x) \triangleq \prod_{\ell=0}^{n-1} (x - \beta_\ell), \tag{105}$$

let $F[x]/m(x)$ be the ring of polynomials modulo $m(x)$, and let ψ be the evaluation mapping

$$\psi : F[x]/m(x) \to F^n : a(x) \mapsto \big(a(\beta_0), \ldots, a(\beta_{n-1})\big), \tag{106}$$

which is a ring isomorphism. A Reed–Solomon code with blocklength n and dimension k may be defined as

$$\{c = (c_0, \ldots, c_{n-1}) \in F^n : \deg \psi^{-1}(c) < k\}. \tag{107}$$

The standard definition of Reed–Solomon codes requires, in addition, that

$$\beta_\ell = \alpha^\ell \quad \text{for } \ell = 0, \ldots, n-1, \tag{108}$$

where $\alpha \in F$ is a primitive n-th root of unity. This additional condition implies

$$m(x) = x^n - 1 \tag{109}$$

and turns ψ into a discrete Fourier transform [6]. However, (108) and (109) will not be required below. In particular, the set $\{\beta_0, \ldots, \beta_{n-1}\}$ will be permitted to contain 0.

In general, the inverse mapping ψ^{-1} can be computed by Lagrange interpolation or according to the Chinese remainder theorem. (For the latter, see also Section VI and [22].)

B. Error Locator Polynomial and Interpolation

The standard decoding problem can be stated as follows. Let $y = (y_0, \ldots, y_{n-1}) \in F^n$ be the received word, which we wish to decompose into

$$y = c + e \tag{110}$$

where c is a codeword and where the Hamming weight of $e = (e_0, \ldots, e_{n-1}) \in F^n$ is as small as possible.

Let $C(x) \triangleq \psi^{-1}(c)$, and analogously $E(x) \triangleq \psi^{-1}(e)$ and $Y(x) \triangleq \psi^{-1}(y)$. Clearly, we have $\deg C(x) < k$ and $\deg E(x) < \deg m(x) = n$.

For any $e \in F^n$, we define the error locator polynomial

$$\Lambda_e(x) \triangleq \prod_{\substack{\ell \in \{0,\ldots,n-1\} \\ e_\ell \neq 0}} (x - \beta_\ell). \tag{111}$$

Clearly, $\deg \Lambda_e(x) = \mathrm{w_H}(e)$ and

$$E(x)\Lambda_e(x) = Q_e(x)m(x) \tag{112}$$

for some nonzero $Q_e(x) \in F[x]$.

(In the literature, the error locator polynomial is usually defined with $(x - \beta_\ell)$ in (111) replaced by $(1 - \beta_\ell x)$, which requires $\beta_\ell \neq 0$, cf. Appendix C.)

The heart of many decoding methods is an algorithm for computing an estimate of the error locator polynomial. If $\Lambda_e(x)$ is known, the polynomial $C(x)$ and/or the codeword c can be recovered in many different ways; in particular, we can use Propositions 8, 9, or 11 below. Proposition 11 is standard; the idea of Proposition 8 was implicitly used in [12] and [11], but explicitly stated only in [21] and [22]; Proposition 9 appears to be entirely new.

Proposition 8 (Multiply-Divide Interpolation): If $\Lambda(x) = \Gamma(x)\Lambda_e(x)$ for some nonzero $\Gamma(x) \in F[x]$ and $\deg \Lambda(x) \leq n - k$, then

$$Y(x)\Lambda(x) \bmod m(x) = C(x)\Lambda(x) \tag{113}$$

and thus

$$C(x) = \frac{Y(x)\Lambda(x) \bmod m(x)}{\Lambda(x)} \tag{114}$$

□

Proof: Assume that $\Lambda(x)$ has the stated properties. Then

$$\begin{aligned} &Y(x)\Lambda(x) \bmod m(x) \\ &\quad = C(x)\Lambda(x) \bmod m(x) + E(x)\Lambda(x) \bmod m(x) \quad (115) \\ &\quad = C(x)\Lambda(x), \quad (116) \end{aligned}$$

where the second term in (115) vanishes because of (112). □

In the special case where $m(x) = x^n - 1$, computing the numerator in (114) amounts to a cyclic convolution.

Proposition 9 (Div-Mod Interpolation): Let $\Lambda(x) = \Gamma(x)\Lambda_e(x)$ for some nonzero $\Gamma(x) \in F[x]$ such that $\Lambda(x)$ divides $m(x)$ and $\deg \Lambda(x) \leq n - k$. Then

$$C(x) = Y(x) \bmod \big(m(x)/\Lambda(x)\big). \tag{117}$$

□

Note that $\Lambda(x) = \gamma\, \Lambda_e(x)$ (with $\gamma \neq 0$) qualifies, but $\Lambda(x)$ is allowed to have additional single roots in the set $\{\beta_0, \ldots, \beta_{n-1}\}$.

Proof of Proposition 9: The polynomial

$$\tilde{m}(x) \triangleq m(x)/\Lambda(x) \tag{118}$$

is a product of linear factors $(x - \beta_\ell)$ (up to a nonzero scale factor) with $\beta_\ell \in \{\beta_0, \ldots, \beta_{n-1}\}$ such that $e_\ell = E(\beta_\ell) = 0$. Thus $\tilde{m}(x)$ divides $E(x)$. Moreover, it is obvious that $\deg \tilde{m}(x) \geq k > \deg C(x)$, and thus

$$\begin{aligned} Y(x) \bmod \tilde{m}(x) &= \big(C(x) + E(x)\big) \bmod \tilde{m}(x) \quad (119) \\ &= C(x). \quad (120) \end{aligned}$$

□

If we need to recover the actual codeword c (rather than just the polynomial $C(x)$), interpolation according to Propositions 8 or 9 requires the additional computation of $c = \psi\big(C(x)\big)$. Alternatively, it may be attractive to compute the error pattern e $(= y - c)$ by Forney's formula [6], [8] or by the Horiguchi–Koetter formula [7], [24]. The latter will be adapted to our setting in Section V-D.4. For the former, we need the polynomial $Q_e(x)$ from (112), or a generalization of it, which can be computed as follows.

Proposition 10 (Generalized Error Locator Equation): Let $\Lambda(x) = \Gamma(x)\Lambda_e(x)$ for some nonzero $\Gamma(x) \in F[x]$. If $\deg \Lambda(x) \leq n - k$, then

$$E(x)\Lambda(x) = Q(x)m(x) \tag{121}$$

with

$$Q(x) \triangleq Y(x)\Lambda(x) \operatorname{div} m(x). \tag{122}$$

□

Proof: From (122) and (113), we have

$$Y(x)\Lambda(x) = Q(x)m(x) + C(x)\Lambda(x) \tag{123}$$

and thus $Q(x)m(x) = E(x)\Lambda(x)$. □

The components of $e = (e_0, \ldots, e_{n-1})$ can be computed from (112) or from (121) using L'Hôpital's rule, which is known as Forney's Formula [6], [8]. Specifically:

Proposition 11 (Forney's Formula): Let $\Lambda(x) = \Gamma(x)\Lambda_e(x)$ for some nonzero $\Gamma(x) \in F[x]$. Assume $\deg \Lambda(x) \le n-k$ and assume that $\Lambda(x)$ has no repeated roots in the set $\{\beta_0, \ldots, \beta_{n-1}\}$. Then,

$$e_\ell \triangleq \begin{cases} 0 & \text{if } \Lambda(\beta_\ell) \neq 0 \\ \dfrac{Q(\beta_\ell)m'(\beta_\ell)}{\Lambda'(\beta_\ell)} & \text{if } \Lambda(\beta_\ell) = 0 \end{cases} \tag{124}$$

for $\ell = 0, 1, \ldots, n-1$, with $Q(x)$ as in (122), and where $\Lambda'(x)$ and $m'(x)$ denote the formal derivatives of $\Lambda(x)$ and $m(x)$, respectively. □

Note that (124) works for general $m(x)$, and $m'(\beta_\ell)$ can be pre-computed. For example, for $m(x) = x^{|F|} - x$, we have $m'(\beta_\ell) = -1$; for the standard case (108) with $m(x) = x^n - 1$, we have $m'(\beta_\ell) = (n \bmod p)\beta_\ell^{-1}$, where p is the characteristic of F.

The quantity $Q(\beta_\ell)$ in (124) can be obtained without explicit computation of $Q(x)$, cf. Theorem 9 below.

Any of the Propositions 8, 9, or 11 can be used for erasures-only decoding where $\Lambda(x)$ is given *a priori*. For this application, it is essential that $\Lambda(\beta_\ell) = 0$ does not imply $e_\ell \neq 0$, i.e., $\Lambda(x)$ may be a nontrivial multiple of $\Lambda_e(x)$.

We now turn to the standard decoding problem (110). Joint errors-and-erasures decoding will be addressed in Appendix E.

C. Key Equations

An estimate of the error locator polynomial can be found via the standard key equation, which will be addressed in Appendix C. However, in the context of this paper, it is more natural to begin with the following alternative key equation. The direct part is well known from [12], but the converse part is new.

Theorem 6 (Alternative Key Equation): If $\mathrm{w_H}(e) \le \frac{n-k}{2}$, then the error locator polynomial $\Lambda_e(x)$ satisfies

$$\deg\big(Y(x)\Lambda_e(x) \bmod m(x)\big) < k + \deg \Lambda_e(x) \tag{125}$$

$$\le n - (n-k)/2. \tag{126}$$

Conversely, for any y and $e \in F^n$ and $t \in \mathbb{R}$ with

$$\mathrm{w_H}(e) \le t \le (n-k)/2, \tag{127}$$

if some nonzero $\Lambda(x) \in F[x]$ with $\deg \Lambda(x) \le t$ satisfies

$$\deg\Big(Y(x)\Lambda(x) \bmod m(x)\Big) < n - t, \tag{128}$$

then $\Lambda(x)$ is a multiple of $\Lambda_e(x)$. □

The proof is given below. In particular, for $t = (n-k)/2$, we obtain the following corollary, which was implicitly assumed in [12], but does not seem to be actually proved in the literature.

Corollary 2: If $\mathrm{w_H}(e) \le \frac{n-k}{2}$, then $\Lambda_e(x)$ is the nonzero polynomial of the smallest degree (unique up to a scale factor) that satisfies

$$\deg\Big(Y(x)\Lambda(x) \bmod m(x)\Big) < \frac{n+k}{2} \tag{129}$$

□

Finding $\Lambda_e(x)$ from (129) is obviously a partial-inverse problem.

Proof of Theorem 6: (125) is immediate from (113), and (126) follows from $k + \mathrm{w_H}(e) \le k + \frac{n-k}{2} = \frac{n+k}{2}$.

As for the converse, assume (127), (128), and $\deg \Lambda(x) \le t$ and consider

$$Y(x)\Lambda(x) \bmod m(x) = C(x)\Lambda(x) + E(x)\Lambda(x) \bmod m(x). \tag{130}$$

Under the stated assumptions, the degree of the left-hand side of (130) is smaller than $n-t$ and also

$$\deg\big(C(x)\Lambda(x)\big) < k + t \le n - t. \tag{131}$$

It follows that

$$\deg\Big(E(x)\Lambda(x) \bmod m(x)\Big) < n - t. \tag{132}$$

Now write

$$E(x)\Lambda(x) = Q(x)m(x) + E(x)\Lambda(x) \bmod m(x) \tag{133}$$

according to the polynomial division theorem. But $E(x)$ (and thus $E(x)\Lambda(x)$) has at least $n - \mathrm{w_H}(e) \ge n - t$ zeros in the set $\{\beta_0, \beta_1, \ldots, \beta_{n-1}\}$. It follows that $E(x)\Lambda(x) \bmod m(x)$ has also at least $n-t$ zeros (in this set), which contradicts (132) unless

$$E(x)\Lambda(x) \bmod m(x) = 0. \tag{134}$$

But any nonzero polynomial $\Lambda(x)$ that satisfies (134) is a multiple of the error locator polynomial (111). □

We have established that finding the error locator polynomial is a partial-inverse problem. It then follows from Proposition 6 that coefficients Y_ℓ of $Y(x)$ with $\ell < k$ are irrelevant and can be set to zero; the remaining coefficients Y_ℓ are syndromes since $C_\ell = 0$ and $Y_\ell = E_\ell$ for $\ell \ge k$.

Theorem 7 (Reduced Key Equation): Let

$$\tilde{b}(x) \triangleq Y_k + Y_{k+1}x + \ldots + Y_{n-1}x^{n-k-1} \tag{135}$$

$$\tilde{m}(x) \triangleq m_k + m_{k+1}x + \ldots + m_n x^{n-k}. \tag{136}$$

If $\mathrm{w_H}(e) \le (n-k)/2$, then $\Lambda(x) = \Lambda_e(x)$ is a nonzero polynomial of the smallest degree (unique up to a scale factor) that satisfies

$$\deg\Big(\tilde{b}(x)\Lambda(x) \bmod \tilde{m}(x)\Big) < \frac{n-k}{2} \tag{137}$$

Moreover,

$$\tilde{b}(x)\Lambda_e(x) \operatorname{div} \tilde{m}(x) = Y(x)\Lambda_e(x) \operatorname{div} m(x). \tag{138}$$

□

Proof: The proof is immediate from Corollary 2 and Proposition 7 with $d = (n+k)/2$ and $s = k$. □

Eq. (138) can be used, e.g., in Forney's formula (124) with $Q(x)$ as in (138).

In the important special cases where $m(x) = x^n - 1$ (as in (109)) or $m(x) = x^n - x$ (for singly extended Reed–Solomon codes), we have $\tilde{m}(x) = x^{n-k}$, and (137) looks like a standard key equation. The same effect can be achieved for general $m(x)$ by the following theorem.

Theorem 8 (Monomialized Key Equation): Let $\tilde{b}(x)$ be the polynomial (67) for $b(x) = Y(x)$ and $\tau = (n-k)/2$. If $\mathrm{w_H}(e) \le \tau$, then $\Lambda(x) = \Lambda_e(x)$ is a nonzero polynomial of the smallest degree that satisfies

$$\deg\Big(\tilde{b}(x)\Lambda(x) \bmod x^{n-k}\Big) < \frac{n-k}{2}. \tag{139}$$

Moreover,

$$\tilde{b}(x)\Lambda_e(x) \operatorname{div} x^{n-k} = Y(x)\Lambda_e(x) \operatorname{div} m(x). \tag{140}$$

□

Proof: From Corollary 2 and Theorem 2 with $d = n - t$. □

Theorem 8 is similar in effect, but slightly more general than, the procedure described in [8, Sec. 6.3]. In particular, it works also for singly extended Reed–Solomon codes.

D. Decoding Algorithms

Determining the error locator $\Lambda_e(x)$ from any of the key equations (129), (137), or (139) is a partial-inverse problem. We thus arrive at the following general decoding procedure:

1) Compute $Y(x) = \psi^{-1}(y)$.
2) Solve any of the key equations (129), (137), or (139) by any of the algorithms of Section IV. If $\mathrm{w_H}(e) \le \frac{n-k}{2}$, then the polynomial $\Lambda(x)$ returned by the algorithm equals $\Lambda_e(x)$, up to a scale factor.
3) Complete decoding, e.g., by means of Propositions 8, 9 or 11, or by other means as described below.

Obviously, this general procedure encompasses a variety of specific algorithms, some of which will be discussed below.

In addition, there are many opportunities for detecting violations of the assumption $\mathrm{w_H}(e) \le (n-k)/2$. In particular, condition (125) can be checked. Alternatively, if (137) is used, then the condition

$$\deg\Big(\tilde{b}(x)\Lambda(x) \bmod \tilde{m}(x)\Big) < \deg\Lambda(x) \tag{141}$$

can be checked. If (114) is used for interpolation, it should be checked whether $\Lambda(x)$ indeed divides $Y(x)\Lambda(x) \bmod m(x)$; if Forney's formula (Proposition 11) is used, it should be checked whether $\mathrm{w_H}(e) = \deg\Lambda(x)$; etc.

In Step 2, the number of iterations in the partial-inverse algorithm is upper bounded by $(n-k)/2 + \mathrm{w_H}(e)$ by Theorem 5; this is smaller than the number of iterations in the Berlekamp–Massey algorithm, especially if $\mathrm{w_H}(e)$ is small.

We now briefly discuss a number of specific decoding algorithms, i.e., specific choices in Steps 2 and 3. First, we consider algorithms that recover the polynomial $C(x)$ or $E(x)$; then we consider algorithms that recover the codeword c (or the error pattern e) directly.

1) Shiozaki–Gao Decoding: Use Algorithm 3 to solve (129). Step 3 is then naturally carried out via

$$C(x) = r^{(1)}(x)/\Lambda(x), \tag{142}$$

which is immediate from $r^{(1)}(x) = Y(x)\Lambda(x) \bmod m(x)$ and Proposition 8. The resulting decoding algorithm essentially coincides with the (essentially identical) algorithms by Gao [11] and Shiozaki [12], except that [11] and [12] both use Algorithm 4 instead of Algorithm 3.

(Shiozaki [12] and Gao [11] propose the same algorithm, but very different proofs. Yet another proof of this algorithm is given in [22] and [13].)

2) Reverse Berlekamp–Massey Decoding for Cyclic Reed–Solomon Codes: Let $m(x) = x^n - 1$ and use Algorithm 1 (with the refinements in Algorithms 1.A and 1.B) to solve any of the key equations (129), (137), or (139).

Two natural methods for Step 3 are as follows. The first method is to use Proposition 8. Computing $r(x) \triangleq Y(x)\Lambda(x) \bmod (x^n - 1)$ (the numerator in (114)) amounts to the cyclic convolution

$$r_\ell = \sum_{i=0}^{\tau} Y_{[\ell-i]}\Lambda_i \tag{143}$$

with $\tau \triangleq \deg\Lambda(x)$. (The computation (143) may be viewed as continuing line 26 of Algorithm 1.A with frozen $\Lambda^{(1)}(x) = \Lambda(x)$.)

A second method for Step 3 begins with

$$\Lambda(x)E(x) \bmod (x^n - 1) = 0 \tag{144}$$

from (112). Note that $E_\ell = Y_\ell$ for $\ell \ge k$, and $E_{k-1}, \ldots, E_0$ can be computed from (144) by the recursion

$$E_{\ell-\tau} = -\frac{1}{\Lambda_\tau}\sum_{i=0}^{\tau-1} E_{\ell-i}\Lambda_i \tag{145}$$

for $\ell = k+\tau-1, k+\tau-2, \ldots, \tau$, where $\tau \triangleq \deg\Lambda(x)$. We then obtain $C(x) = Y(x) - E(x)$. (Analogous recursions for standard Berlekamp–Massey decoding are given, e.g., in [6] and [13].)

3) Using Algorithm 2: For general $m(x)$, it may be attractive to use Algorithm 2 to solve (137), and to ask the algorithm to return also $Q(x) \triangleq Q^{(1)}(x)$. Two natural methods for Step 3 are as follows. The first method is to use Proposition 8 and to compute the numerator $r(x)$ in (114) from

$$r(x) = Y(x)\Lambda(x) - Q(x)m(x). \tag{146}$$

(This computation may be viewed as continuing line 15 of Algorithm 2 with frozen $\Lambda^{(1)}(x)$ and $Q^{(1)}(x)$.)

A second method for Step 3 begins with

$$E(x)\Lambda(x) = Q(x)m(x) \tag{147}$$

from (138) and Proposition 10, from which we obtain the recursion

$$E_{\ell-\tau} = -\frac{1}{\Lambda_\tau}\left(\sum_{i=0}^{\tau-1} E_{\ell-i}\Lambda_i - \sum_{i=0}^{\nu} m_{\ell-i}Q_i\right) \tag{148}$$

for $\ell = k+\tau-1, k+\tau-2, \ldots, \tau$, where τ and ν are the degrees of $\Lambda(x)$ and $m(x)$, respectively.

4) Using Forney's Formula and Horiguchi–Koetter Interpolation: Assume now that we wish to recover not $C(x)$, but the codeword $c \in F^n$ (or, equivalently, the error pattern $e = y - c$). In this case, the following methods are often more attractive than first finding $C(x)$ and then computing $c = \psi\big(C(x)\big)$.

Recall that we can use any of the algorithms of Section IV to solve any of the key equations (129), (137), or (139). If Algorithm 2 is used, then the polynomial $Q(x)$ from (122) is already available (as $Q^{(1)}(x)$), so Forney's formula (124) can be applied directly. (This works for all three key equations due to (138) and (140).)

If Algorithms 1 or 3 are used, Forney's formula (124) can be used without explicitly computing $Q(x)$, as follows.

Theorem 9 (Generalized Horiguchi–Koetter Interpolation): If $\mathrm{w_H}(e) \le \frac{n-k}{2}$ and $\Lambda(\beta_\ell) = 0$, then

$$Q(\beta_\ell) = \frac{\gamma}{\Lambda^{(2)}(\beta_\ell)} \tag{149}$$

with

$$\gamma \triangleq \frac{\kappa_2 \cdot \operatorname{lcf} \Lambda(x)}{\operatorname{lcf} m(x)} \tag{150}$$

where $\Lambda(x) = \Lambda^{(1)}(x)$, $\Lambda^{(2)}(x)$, and κ_2 are obtained from the partial-inverse algorithm (when it terminates). □

Theorem 9 is a nontrivial adaptation and generalization (with an entirely different proof) of the original idea by Horiguchi, which is restricted to Berlekamp–Massey decoding [7], [24]; see also Appendix C.

Proof of Theorem 9: We assume $\mathrm{w_H}(e) \le (n-k)/2$, so that $\Lambda^{(1)}(x) = \Lambda(x) = \Lambda_e(x)$ (up to a scale factor) when the partial-inverse algorithm terminates. Also, both $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ are minimal partial inverses and

$$\deg \Lambda^{(1)}(x) = \deg m(x) - \deg\big(b(x)\Lambda^{(2)}(x) \bmod m(x)\big) \tag{151}$$

by Lemma 1. From Lemma 3 (eq. (191)) in Appendix B, we then have

$$\Lambda(x)Q^{(2)}(x) - \Lambda^{(2)}(x)Q(x) = -\gamma, \tag{152}$$

and the theorem follows. □

VI. Application to Polynomial Remainder Codes

Polynomial remainder codes [12], [20]–[22] are a class of codes that include Reed–Solomon codes as a special case. It is well known that polynomial remainder codes can be decoded by the Euclidean algorithm, but these codes have not been amenable to Berlekamp–Massey decoding. In this section, we briefly outline how decoding via the alternative key equation (Theorem 6 in Section V-C) generalizes to polynomial remainder codes, which can thus be decoded by all the algorithms of Section IV.

Let $m_0(x), \ldots, m_{n-1}(x) \in F[x]$ be relatively prime polynomials and let $m(x) \triangleq \prod_{\ell=0}^{n-1} m_\ell(x)$. Let $R_m \triangleq F[x]/m(x)$ denote the ring of polynomials modulo $m(x)$ and let $R_{m_\ell} \triangleq F[x]/m_\ell(x)$. The mapping (106) is generalized to the ring isomorphism

$$\begin{aligned}\psi : R_m &\to R_{m_0} \times \ldots \times R_{m_{n-1}} : \\ a(x) &\mapsto \psi(a) \triangleq \big(\psi_0(a), \ldots, \psi_{n-1}(a)\big)\end{aligned} \tag{153}$$

with $\psi_\ell(a) \triangleq a(x) \bmod m_\ell(x)$. Following [22], a polynomial remainder code may be defined as

$$\{c = (c_0, \ldots, c_{n-1}) \in R_{m_0} \times \ldots \times R_{m_{n-1}} : \deg \psi^{-1}(c) < K\} \tag{154}$$

where

$$K \triangleq \sum_{\ell=0}^{k-1} \deg m_\ell(x) \tag{155}$$

for some fixed k, $0 < k < n$. We also define

$$N \triangleq \deg m(x) = \sum_{\ell=0}^{n-1} \deg m_\ell(x). \tag{156}$$

As in Section V, let $y = c + e$ be the received word with codeword c and error e, and let $C(x) \triangleq \psi^{-1}(c)$, $E(x) \triangleq \psi^{-1}(e)$, and $Y(x) \triangleq \psi^{-1}(y)$. Clearly, $\deg C(x) < K$ and $\deg E(x) < N$.

For such codes, the error locator polynomial

$$\Lambda_e(x) \triangleq \prod_{\substack{\ell \in \{0,\ldots,n-1\} \\ e_\ell \neq 0}} m_\ell(x) \tag{157}$$

and the error factor polynomial [22]

$$\Lambda_E(x) \triangleq m(x)/\gcd\big(E(x), m(x)\big) \tag{158}$$

do not, in general, coincide. Note that $\Lambda_E(x)$ is a nonzero polynomial of the smallest degree such that

$$E(x)\Lambda_E(x) \bmod m(x) = 0. \tag{159}$$

If all moduli $m_\ell(x)$ are irreducible, then $\Lambda_E(x) = \gamma\,\Lambda_e(x)$ for some nonzero $\gamma \in F$. In any case, $\deg \Lambda_E(x) \le \deg \Lambda_e(x)$.

We then have the following generalizations of Propositions 8 and 9. (The former was first stated in [21] and [22].)

Proposition 12 (Multiply-Divide Interpolation): If $\Lambda(x)$ is a nonzero polynomial multiple of $\Lambda_E(x)$ with $\deg \Lambda(x) \le N - K$, then

$$C(x) = \frac{Y(x)\Lambda(x) \bmod m(x)}{\Lambda(x)} \tag{160}$$

□

The proof agrees with the proof of Proposition 8.

Proposition 13 (Div-Mod Interpolation): Let $\Lambda(x) = \Gamma(x)\Lambda_E(x)$ for some nonzero $\Gamma(x) \in F[x]$ such that $\Lambda(x)$ divides $m(x)$ and $\deg \Lambda(x) \le N - K$. Then

$$C(x) = Y(x) \bmod \big(m(x)/\Lambda(x)\big). \tag{161}$$

□

The proof is an obvious adaptation of the proof of Proposition 9.

We also have the following generalization of Theorem 6:

Theorem 10 (Key Equation for Polynomial Remainder Codes): For given y and e with $\deg \Lambda_E(x) \le t \le \frac{N-K}{2}$, assume that some nonzero polynomial $\Lambda(x)$ with $\deg \Lambda(x) \le t$ satisfies

$$\deg\big(Y(x)\Lambda(x) \bmod m(x)\big) < N - t. \tag{162}$$

Then $\Lambda(x)$ is a multiple of $\Lambda_E(x)$. Conversely, $\Lambda(x) = \Lambda_E(x)$ is a polynomial of the smallest degree that satisfies (162). □

It follows that the decoding procedure of Section V works also for polynomial remainder codes, except that n, k, and $\Lambda_e(x)$ are replaced by N, K, and $\Lambda_E(x)$, respectively.

Proof of Theorem 10: We have $Y(x) = \psi^{-1}(y) = C(x) + E(x)$ with $\deg C(x) < K$, and $\deg E(x) \ge N - \deg \Lambda_E(x) \ge N - t$ if $E(x) \ne 0$. Consider

$$Y(x)\Lambda(x) \bmod m(x) = C(x)\Lambda(x) + E(x)\Lambda(x) \bmod m(x). \tag{163}$$

Under the stated assumptions, the degree of the left-hand side of (163) is less than $N - t$, and also

$$\deg\Big(C(x)\Lambda(x)\Big) < K + t \le N - t. \tag{164}$$

It follows that

$$\deg\Big(E(x)\Lambda(x) \bmod m(x)\Big) < N - t. \tag{165}$$

Now write

$$E(x)\Lambda(x) = Q(x)m(x) + \tilde{E}(x) \tag{166}$$

with

$$\tilde{E}(x) \triangleq E(x)\Lambda(x) \bmod m(x) \tag{167}$$

according to the polynomial division theorem, and let

$$m_E(x) \triangleq m(x)/\Lambda_E(x) = \gcd\big(E(x), m(x)\big). \tag{168}$$

Since $\deg \Lambda_E(x) \le t$, we have $\deg m_E(x) \ge N - t$. Taking (166) modulo $m_E(x)$ yields

$$\tilde{E}(x) \bmod m_E(x) = 0 \tag{169}$$

since $E(x) \bmod m_E(x) = 0$. It follows that $\tilde{E}(x) = 0$ since $\deg \tilde{E}(x) < N - t \le \deg m_E(x)$. Finally, from $\tilde{E}(x) = 0$ and (167), we obtain

$$E(x)\Lambda(x) \bmod m(x) = 0. \tag{170}$$

But any nonzero polynomial $\Lambda(x)$ that satisfies (170) is a multiple of the error factor polynomial (158). □

VII. Conclusion

We have introduced the partial-inverse problem for polynomials mod $m(x)$, which is closely related to many classical concepts and algorithms in coding theory. In contrast to the LFSRS problem, which underlies the Berlekamp–Massey algorithm, the partial-inverse problem has always a unique solution (up to a scale factor) and comes with a nontrivial upper bound on the degree of the solution. We also derived many new properties of the partial-inverse problem that are useful for its application to decoding.

We presented several versions of a new algorithm for solving the partial-inverse problem, which can be applied to decoding Reed–Solomon codes and generalizations of Reed–Solomon codes (including polynomial remainder codes). The basic partial-inverse algorithm strongly resembles the Berlekamp–Massey algorithm, and an easy variation of it may be viewed as a version of the Euclidean algorithm. These algorithms can be used to solve the standard key equation (cf. Appendix C), but they are more naturally applied to an alternative key equation with a new converse. We also pointed out a variety of options for interpolation including adaptations of Horiguchi–Koetter interpolation.

In the appendices, we address Padé approximations, rational-function reconstruction, and joint errors-and-erasures decoding.

We hope to have demonstrated that the partial-inverse approach is attractive for classical algebraic decoding up to half the minimum distance. However, the partial-inverse approach is even more attractive for decoding interleaved Reed–Solomon codes (and related codes) beyond half the minimum distance [25], [26], which will be addressed in a companion paper.

Appendix A
Relations to Padé Approximations and Rational-Function Reconstruction

In this appendix, we discuss the exact relation between the partial-inverse problem (as defined in Section I) and some closely related problems in the literature including Padé approximation and rational-function reconstruction as in [10], and the McEliece–Shearer problem from [9].

A. McEliece–Shearer Problem

The following problem is paraphrased from [9], see also [10, Sec. 5.7]:

McEliece–Shearer Problem: Let $b(x)$ and $m(x)$ be nonzero polynomials over some field F, with $\deg b(x) < \deg m(x)$. For fixed $d \in \mathbb{Z}$ with $0 \le d \le \deg m(x)$, find a pair of polynomials $r(x)$ and $\Lambda(x) \ne 0$ such that

$$b(x)\Lambda(x) \equiv r(x) \mod m(x), \tag{171}$$

$$\deg r(x) < d, \tag{172}$$

$$\deg \Lambda(x) \le \deg m(x) - d. \tag{173}$$

□

In general, this problem may have multiple solutions. In any solution, $r(x) = b(x)\Lambda(x) \bmod m(x)$ is determined by $\Lambda(x)$.

Clearly, the solution $\Lambda(x)$ with the smallest degree is the solution of the corresponding partial-inverse problem, for which (173) is redundant by Proposition 3. McEliece and Shearer [9] showed that this particular solution can be computed by the Euclidean algorithm. Moreover, [9, Th. 1] implies the following fact:

Theorem 11 (All Solutions): The solutions of the McEliece–Shearer problem consist of the pairs

$$\Lambda(x) = a(x)\,\Lambda^{(0)}(x) \quad \text{and} \quad r(x) = a(x)\,r^{(0)}(x), \tag{174}$$

where $\Lambda^{(0)}(x)$ is the solution of the smallest degree (= the solution of the partial-inverse problem) and where $a(x)$ is any nonzero polynomial such that (172) and (173) hold. □

The proofs of this fact in [9] and [10, Lemma 5.15] use properties of the Euclidean algorithm. Below (in Section VII-C), we give a different proof that is not based on an algorithm.

B. Padé Approximation and Rational-Function Approximation

The rational-function approximation problem of [10, Sec. 5.7] is obtained from the McEliece–Shearer problem (as defined above) by adding the condition

$$\gcd\big(\Lambda(x), m(x)\big) = 1. \tag{175}$$

If (175) holds, $\Lambda(x)$ has an inverse in $F[x]/m(x)$, and (171) is equivalent to

$$b(x) \equiv \frac{r(x)}{\Lambda(x)} \mod m(x). \tag{176}$$

The Padé approximation problem is then obtained by specialization to $m(x) = x^{\nu}$ [10]. Note that, by Theorem 11, all polynomials $\Lambda(x)$ and $r(x)$ that satisfy (171)–(173) yield the same rational function $r(x)/\Lambda(x)$ in (176), and the rational-function approximation problem has a solution if and only if the solution of the partial-inverse problem satisfies (175).

A necessary (but not sufficient) condition for the existence of a solution is

$$\deg \gcd\big(b(x), m(x)\big) < d, \tag{177}$$

which can be seen as follows. Let $\Lambda(x)$ and $r(x)$ be a solution, i.e.,

$$b(x)\Lambda(x) = q(x)m(x) + r(x) \tag{178}$$

with $\deg r(x) < d$. Clearly, $\gcd\big(b(x), m(x)\big)$ divides $r(x)$; if $\deg \gcd\big(b(x), m(x)\big) \geq d$, then $r(x) = 0$, which contradicts (176), so (175) cannot hold.

C. Proof of Theorem 11

One direction is trivial: all polynomials (174) obviously satisfy (171)–(173).

For the other direction, let $\Lambda^{(0)}(x)$ (with $r^{(0)}(x) \triangleq b(x)\Lambda^{(0)}(x) \bmod m(x)$) be the solution of the partial inverse problem and let $\Lambda(x)$ (with $r(x) \triangleq b(x)\Lambda(x) \bmod m(x)$) be any other solution of (171)–(173). We thus have

$$b(x)\Lambda(x) = q(x)m(x) + r(x) \tag{179}$$

and

$$b(x)\Lambda^{(0)}(x) = q^{(0)}(x)m(x) + r^{(0)}(x) \tag{180}$$

with

$$\gcd\big(\Lambda^{(0)}(x), q^{(0)}(x)\big) = 1 \tag{181}$$

by Theorem 1. Clearly, we have

$$b(x)\Lambda(x)\Lambda^{(0)}(x) = \Lambda(x)q^{(0)}(x)m(x) + \Lambda(x)r^{(0)}(x) \tag{182}$$

$$= \Lambda^{(0)}(x)q(x)m(x) + \Lambda^{(0)}(x)r(x). \tag{183}$$

Since both $\deg\Lambda(x) + \deg r^{(0)}(x) < \deg m(x)$ and $\deg\Lambda^{(0)}(x) + \deg r(x) < \deg m(x)$, (182) and (183) imply both that

$$\Lambda(x)r^{(0)}(x) = \Lambda^{(0)}(x)r(x) \tag{184}$$

and

$$\Lambda(x)q^{(0)}(x) = \Lambda^{(0)}(x)q(x). \tag{185}$$

Thus $\Lambda^{(0)}(x)$ divides $\Lambda(x)q^{(0)}(x)$, and (181) implies that $\Lambda^{(0)}(x)$ divides $\Lambda(x)$. We have established that $\Lambda(x) = a(x)\Lambda^{(0)}(x)$ for some $a(x)$, and $r(x) = a(x)r^{(0)}(x)$ follows from (184).

APPENDIX B
HORIGUCHI PROPERTIES

The following (new) lemma is the key to our derivation of Horiguchi–Koetter interpolation in Section V-D and Appendix C.

Lemma 3: For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, let both $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ be minimal partial inverses of $b(x)$ and let

$$r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \bmod m(x) \tag{186}$$

$$q^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \operatorname{div} m(x) \tag{187}$$

$$r^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \bmod m(x) \tag{188}$$

$$q^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \operatorname{div} m(x). \tag{189}$$

Assume further that

$$\deg\Lambda^{(1)}(x) = \deg m(x) - \deg r^{(2)}(x). \tag{190}$$

Then we have both

$$\Lambda^{(1)}(x)q^{(2)}(x) - \Lambda^{(2)}(x)q^{(1)}(x) = -\gamma \tag{191}$$

and

$$\Lambda^{(1)}(x)r^{(2)}(x) - \Lambda^{(2)}(x)r^{(1)}(x) = \gamma\, m(x), \tag{192}$$

where the nonzero constant $\gamma \in F$ is given by

$$\gamma \triangleq \frac{\operatorname{lcf}\Lambda^{(1)}(x) \cdot \operatorname{lcf} r^{(2)}(x)}{\operatorname{lcf} m(x)} \tag{193}$$

□

The meaning of (190) derives from Lemma 1: $\Lambda^{(1)}(x)$ is the minimal partial inverse of the smallest degree such that

$$\deg r^{(1)} < \deg r^{(2)}. \tag{194}$$

In particular, we have

$$\deg\Lambda^{(1)} > \deg\Lambda^{(2)}. \tag{195}$$

Eq. (191) is used in Section V-D; (192) is used in Appendix C.

Proof of Lemma 3: We begin by writing

$$b(x)\Lambda^{(1)}(x) = q^{(1)}(x)m(x) + r^{(1)}(x) \tag{196}$$

and

$$b(x)\Lambda^{(2)}(x) = q^{(2)}(x)m(x) + r^{(2)}(x), \tag{197}$$

from which we obtain

$$\begin{aligned} & b(x)\Lambda^{(1)}(x)\Lambda^{(2)}(x) \\ & \quad = \Lambda^{(1)}(x)q^{(2)}(x)m(x) + \Lambda^{(1)}(x)r^{(2)}(x) \qquad (198) \\ & \quad = \Lambda^{(2)}(x)q^{(1)}(x)m(x) + \Lambda^{(2)}(x)r^{(1)}(x), \qquad (199) \end{aligned}$$

and further

$$\begin{aligned} & \Big(\Lambda^{(1)}(x)q^{(2)}(x) - \Lambda^{(2)}(x)q^{(1)}(x)\Big)\, m(x) \\ & \quad = \Lambda^{(2)}(x)r^{(1)}(x) - \Lambda^{(1)}(x)r^{(2)}(x). \qquad (200) \end{aligned}$$

We now claim that the degree of the right side of (200) equals $\deg m(x)$. Indeed, from (194) and (195), we have

$$\deg \Lambda^{(2)}(x) + \deg r^{(1)}(x) < \deg \Lambda^{(1)}(x) + \deg r^{(2)}(x), \quad (201)$$

and from (190), we have

$$\deg \Lambda^{(1)}(x) + \deg r^{(2)}(x) = \deg m(x). \quad (202)$$

Considering now the left side of (200), it is clear that

$$\alpha \triangleq \Lambda^{(1)}(x)q^{(2)}(x) - \Lambda^{(2)}(x)q^{(1)}(x) \quad (203)$$

is a nonzero constant, which is determined by

$$\alpha \operatorname{lcf} m(x) = -\operatorname{lcf} \Lambda^{(1)}(x) \cdot \operatorname{lcf} r^{(2)}(x), \quad (204)$$

and (191) follows. Eq. (192) then follows from (200). □

Appendix C
Decoding Reed–Solomon Codes via the Standard Key Equation

As mentioned, the definition of the error locator polynomial in Section V is not quite standard, and neither are the key equations (129), (137), and (139). We now show that the standard key equation (with the standard error locator polynomial) is also a partial inverse problem and can thus be solved by any of the algorithms of Section IV. In addition, we adapt Horiguchi–Koetter interpolation [7], [24] for use with reverse Berlekamp–Massey decoding (i.e., Algorithm 1 of Section IV).

A. Standard Key Equation as a Partial-Inverse Problem

The standard definition of the error locator polynomial is not (111), but

$$\Lambda(x) \triangleq \prod_{\substack{\ell \in \{0,\ldots,n-1\} \\ e_\ell \neq 0}} (1 - \beta_\ell x), \quad (205)$$

which requires $0 \notin \{\beta_0, \ldots, \beta_{n-1}\}$. Different versions of the key equation exist in the literature [3]–[8], but they are equivalent to the standard version [27]

$$S(x)\Lambda(x) \equiv \Gamma(x) \mod x^{n-k}, \quad (206)$$

where n and k are the blocklength and the dimension of the code, respectively, and where $S(x)$ is a (given) syndrome polynomial with $\deg S(x) < n-k$. The desired solution (under the assumption $\mathrm{w_H}(e) \leq (n-k)/2$) is a pair $\Gamma(x)$ and $\Lambda(x) \neq 0$ such that

$$\deg \Gamma(x) < \deg \Lambda(x) \leq (n-k)/2 \quad (207)$$

and

$$\gcd\big(\Lambda(x), \Gamma(x)\big) = 1. \quad (208)$$

The polynomial $\Gamma(x)$ in (205) is commonly called the error evaluator polynomial.

Proposition 14: Any solution $\Lambda(x)$ of (206)–(208) solves the partial-inverse problem (as defined in Section I) with $b(x) = S(x)$, $m(x) = x^{n-k}$ and $d = \lfloor \frac{n-k}{2} \rfloor$. □

In consequence, the standard key equation can be solved by the algorithms of Section IV. The proof of Proposition 14 follows from the following lemma.

Lemma 4: Let $b(x)$ and $m(x)$ be nonzero polynomials in $F[x]$ with $\deg b(x) < \deg m(x)$ and with $\deg m(x) \geq 2d$ for some positive $d \in \mathbb{Z}$. If some nonzero $\Lambda(x) \in F[x]$ with

$$r(x) \triangleq b(x)\Lambda(x) \bmod m(x) \quad (209)$$

satisfies

$$\deg \Lambda(x) \leq d, \quad (210)$$

$$\deg r(x) < d, \quad (211)$$

and

$$\gcd\big(\Lambda(x), r(x)\big) = 1, \quad (212)$$

then $\Lambda(x)$ is a solution of the partial-inverse problem

$$\deg\big(b(x)\Lambda(x) \bmod m(x)\big) < d. \quad (213)$$

□

Proof: Assume that $\Lambda(x)$ is not a solution of the partial inverse problem (213), i.e., there exists some nonzero $\Lambda^{(1)}(x)$ with $\deg \Lambda^{(1)}(x) < \deg \Lambda(x)$ such that

$$r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \bmod m(x) \quad (214)$$

satisfies $\deg r^{(1)}(x) < d$. Multiplying (209) by $\Lambda^{(1)}(x)$ and (214) by $\Lambda(x)$ yields

$$r^{(1)}(x)\Lambda(x) \equiv r(x)\Lambda^{(1)}(x) \mod m(x), \quad (215)$$

from which we obtain

$$r^{(1)}(x)\Lambda(x) = r(x)\Lambda^{(1)}(x) \quad (216)$$

by (210) and (211). It follows that $\Lambda(x)$ divides $\Lambda^{(1)}(x)$ because of (212). But this is impossible since $\deg \Lambda^{(1)}(x) < \deg \Lambda(x)$. □

B. Forney's Formula and Horiguchi–Koetter Interpolation

According to Forney's Formula [6, Sec. 7.4]) and [8, Sec. 6.3], the error pattern satisfies

$$e_\ell \triangleq \begin{cases} -\dfrac{\beta_\ell \cdot \Gamma(\beta_\ell^{-1})}{\Lambda'(\beta_\ell^{-1})} & \text{if } \Lambda(\beta_\ell^{-1}) = 0 \\ 0 & \text{if } \Lambda(\beta_\ell^{-1}) \neq 0 \end{cases} \quad (217)$$

for $\ell = 0, 1, \ldots, n-1$, where $\Lambda'(x)$ denotes the formal derivative of $\Lambda(x)$.

If the key equation (206)–(208) is solved by the Euclidean algorithm (or by Algorithm 3 of Section IV), then both $\Lambda(x)$ and $\Gamma(x)$ are available and (217) can be used directly.

On the other hand, if the key equation (206)–(208) is solved by the Berlekamp–Massey algorithm or by Algorithm 1 of Section IV, then only $\Lambda(x)$ is immediately available. In this case, we can compute $\Gamma(x)$ from

$$\Gamma(x) = S(x)\Lambda(x) \bmod x^{n-k}; \tag{218}$$

alternatively, we can use Horiguchi–Koetter interpolation as follows. For standard Berlekamp–Massey decoding, Horiguchi–Koetter interpolation is described in [24] and [7]. For reverse Berlekamp–Massey decoding, i.e., if Algorithm 1 of Section IV is used to solve the standard key equation according to Proposition 14, we have

Theorem 12 (Horiguchi–Koetter Interpolation for Reverse Berlekamp–Massey Decoding): If $\mathrm{w_H}(e) \leq \frac{n-k}{2}$ and $\Lambda(\beta_\ell^{-1}) = 0$, then

$$\Gamma(\beta_\ell^{-1}) = -\frac{\gamma \cdot \beta_\ell^{-(n-k)}}{\Lambda^{(2)}(\beta_\ell^{-1})} \tag{219}$$

with

$$\gamma \triangleq \frac{\kappa_2 \cdot \operatorname{lcf} \Lambda(x)}{\operatorname{lcf} m(x)} \tag{220}$$

where $\Lambda(x) = \Lambda^{(1)}(x)$, $\Lambda^{(2)}(x)$, and κ_2 are obtained from the partial-inverse algorithm (when it terminates). □

Proof: Assuming $\mathrm{w_H}(e) \leq (n-k)/2$, we have $\Lambda^{(1)}(x) = \Lambda(x) = \Lambda_e(x)$, up to a scale factor, when the algorithm terminates. Note that both $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ are minimal partial inverses by Theorem 4. From Lemma 1, we have

$$\deg \Lambda^{(1)}(x) = \deg m(x) - \deg r^{(2)}(x) \tag{221}$$

with $r^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \bmod m(x)$. The theorem then follows from Lemma 3, eq. (192), in Appendix B with $\Lambda^{(1)}(x) = \Lambda(x)$, $r^{(1)}(x) = \Gamma(x)$, and $m(x) = x^{n-k}$. □

Appendix D
Solving the Partial-Inverse Problem With a Mandatory Factor

In preparation for errors-and-erasures decoding, consider the partial-inverse problem of Section I with the additional condition that $\Lambda(x)$ is required to be a (nonzero) multiple of some given nonzero polynomial $\Gamma(x) \in F[x]$ i.e.,

$$\Lambda(x) = \Gamma(x)\tilde{\Lambda}(x); \tag{222}$$

in other words, (1) is replaced by

$$\deg\big(b(x)\Gamma(x)\tilde{\Lambda}(x) \bmod m(x)\big) < d. \tag{223}$$

This problem can be solved as follows. First, we note that (223) is equivalent to

$$\deg\Big(\tilde{b}(x)\tilde{\Lambda}(x) \bmod m(x)\Big) < d \tag{224}$$

with

$$\tilde{b}(x) \triangleq b(x)\Gamma(x) \bmod m(x). \tag{225}$$

We thus have again a standard partial-inverse problem (as in Section I) with $b(x)$ and $\Lambda(x)$ replaced by $\tilde{b}(x)$ and $\tilde{\Lambda}(x)$, respectively. In particular, the degree bound (10) applies to $\tilde{\Lambda}(x)$, from which we obtain

$$\deg \Lambda(x) \leq \deg m(x) - d + \deg \Gamma(x). \tag{226}$$

Note that, in general, the degree of the right-hand side of (226) may exceed the degree of $m(x)$.

It is thus obvious that $\Lambda(x)$ can be computed as follows:

1) Compute $\tilde{b}(x)$ according to (225).
2) Run any of the algorithms of Section IV (with $\tilde{b}(x)$ instead of $b(x)$) to obtain $\tilde{\Lambda}(x)$.
3) Compute $\Lambda(x)$ from (222).

Algorithm 6 (see box) is a variation of this procedure where $\Lambda(x)$ is computed directly; $\tilde{\Lambda}(x)$ does not appear explicitly and only the leading term of $\tilde{b}(x)$ is required. Algorithm 6 is an easy modification of Algorithm 1 (with input $\tilde{b}(x)$) based on (222) and the relation

$$\tilde{b}(x)\tilde{\Lambda}(x) \equiv b(x)\Lambda(x) \mod m(x). \tag{227}$$

The translation of Algorithm 6 to corresponding versions of Algorithms 2 and 3 (Quotient Saving and Remainder Saving, respectively), is obvious.

Algorithm 6: Mandatory-Factor Partial-Inverse Algorithm

Input: $b(x)$, $m(x)$, d, $\Gamma(x)$, and leading term of $\tilde{b}(x)$ (225).
Output: $\Lambda(x)$.

```
1   if deg b̃(x) < d begin
2       return Λ(x) := Γ(x)
3   end
4   Λ^(1)(x) := 0,  d1 := deg m(x),  κ1 := lcf m(x)
5   Λ^(2)(x) := Γ(x),  d2 := deg b̃(x),  κ2 := lcf b̃(x)
6   loop begin
7       Λ^(1)(x) := κ2 Λ^(1)(x) − κ1 x^(d1−d2) Λ^(2)(x)
8       d1 := deg(b(x)Λ^(1)(x) mod m(x))
9       if d1 < d begin
10          return Λ(x) := Λ^(1)(x)
11      end
12      κ1 := lcf(b(x)Λ^(1)(x) mod m(x))
13      if d1 < d2 begin
14          (Λ^(1)(x), Λ^(2)(x)) := (Λ^(2)(x), Λ^(1)(x))
15          (d1, d2) := (d2, d1)
16          (κ1, κ2) := (κ2, κ1)
17      end
18  end
```

Appendix E
Joint Errors-and-Erasures Decoding of Reed–Solomon Codes

The partial-inverse approach to decoding Reed–Solomon codes is easily extended to errors-and-erasures decoding. Consider a Reed–Solomon code as in Section V-A, but without requiring (108) and (109). Let $y = (y_0, \ldots, y_{n-1}) \in F^n$ be the received word and let $\mathcal{Z} \subset \{0, \ldots, n-1\}$ be the set of

erased positions, which is known to the decoder: for $\ell \in \mathcal{Z}$, the decoder assumes that the symbol y_ℓ is useless and can be ignored.

Let c be the transmitted codeword and let $y = c + e + z$ with error $e \in F^n$ defined by

$$e_\ell \stackrel{\triangle}{=} \begin{cases} y_\ell - c_\ell, & \ell \notin \mathcal{Z} \\ 0, & \ell \in \mathcal{Z} \end{cases} \tag{228}$$

for $\ell \in \{0, \ldots, n-1\}$, and with $z \in F^n$ defined by

$$z_\ell \stackrel{\triangle}{=} \begin{cases} y_\ell - c_\ell, & \ell \in \mathcal{Z} \\ 0, & \ell \notin \mathcal{Z}. \end{cases} \tag{229}$$

Let $\mathcal{E} \subset \{0, \ldots, n-1\}$ be the positions of the (non-erased) errors, i.e., $\ell \in \mathcal{E}$ iff $e_\ell \neq 0$. (Note that $\mathcal{Z} \cap \mathcal{E} = \emptyset$ by this definition.) We then have both the error locator polynomial

$$\Lambda_e(x) \stackrel{\triangle}{=} \prod_{\ell \in \mathcal{E}} (x - \beta_\ell) \tag{230}$$

and the erasure locator polynomial

$$\Lambda_z(x) \stackrel{\triangle}{=} \prod_{\ell \in \mathcal{Z}} (x - \beta_\ell), \tag{231}$$

and the latter is known to the decoder.

As in Section V-B, let $Y(x) \stackrel{\triangle}{=} \psi^{-1}(y)$, and analogously $C(x) \stackrel{\triangle}{=} \psi^{-1}(c)$ and $E(x) \stackrel{\triangle}{=} \psi^{-1}(e)$ and $Z(x) \stackrel{\triangle}{=} \psi^{-1}(z)$. We then have both

$$E(x)\Lambda_e(x) \bmod m(x) = 0 \tag{232}$$

and

$$Z(x)\Lambda_z(x) \bmod m(x) = 0. \tag{233}$$

We now propose three different methods for errors-and-erasures decoding, all of which will correct any combination of errors and erasures with

$$2|\mathcal{E}| + |\mathcal{Z}| \leq n - k. \tag{234}$$

Methods I and II below are similar to methods described in [6], [28] and [29], which use either the Berlekamp–Massey algorithm or the Euclidean algorithm. Method III does not seem to have a counterpart in the literature prior to [23].

A. Method I

Assume $|\mathcal{Z}| \leq n - k$ and let

$$\hat{Y}(x) \stackrel{\triangle}{=} Y(x)\Lambda_z(x) \bmod m(x) \tag{235}$$

$$= \hat{C}(x) + \hat{E}(x) \tag{236}$$

with $\hat{C}(x) \stackrel{\triangle}{=} C(x)\Lambda_z(x)$ and

$$\hat{E}(x) \stackrel{\triangle}{=} E(x)\Lambda_z(x) \bmod m(x). \tag{237}$$

With $\hat{k} \stackrel{\triangle}{=} k + |\mathcal{Z}|$, we have $\deg \hat{C}(x) < \hat{k}$. We then have the following generalizations of Theorem 6 and Corollary 2.

Theorem 13: If $\mathrm{w_H}(e) \leq \frac{n-\hat{k}}{2}$, then

$$\deg\big(\hat{Y}(x)\Lambda_e(x) \bmod m(x)\big) < \hat{k} + \deg \Lambda_e(x) \tag{238}$$

$$\leq n - (n - \hat{k})/2. \tag{239}$$

Conversely, for $t \in \mathbb{R}$ with

$$\mathrm{w_H}(e) \leq t \leq (n - \hat{k})/2, \tag{240}$$

if some nonzero $\Lambda(x) \in F[x]$ with $\deg \Lambda(x) \leq t$ satisfies

$$\deg\Big(\hat{Y}(x)\Lambda(x) \bmod m(x)\Big) < n - t, \tag{241}$$

then $\Lambda(x)$ is a multiple of $\Lambda_e(x)$. □

Corollary 3: If $\mathrm{w_H}(e) \leq \frac{n-\hat{k}}{2}$, then $\Lambda_e(x)$ is the nonzero polynomial of the smallest degree (unique up to a scale factor) that satisfies (241) with $t \stackrel{\triangle}{=} (n - \hat{k})/2$. □

The proof of Theorem 13 is essentially identical to the proof of Theorem 6 and is omitted.

We can thus decode essentially as in Section V-D: compute $\hat{Y}(x)$ as in (235) and run any of the algorithms of Section IV with $b(x) = \hat{Y}(x)$ and $d = \hat{k} + \lceil \frac{n-\hat{k}}{2} \rceil$. Complete decoding, e.g., by (114) with $\Lambda(x) = \Lambda_e(x)\Lambda_z(x)$.

B. Method II

This method uses Algorithm 6 of Appendix D and does not require the computation of (235). We begin by noting that (241) can be written as

$$\hat{Y}(x)\Lambda(x) \bmod m(x) = Y(x)\big(\Lambda_z(x)\Lambda(x)\big) \bmod m(x), \tag{242}$$

from which we obtain the following variation of Corollary 3.

Corollary 4: If $\mathrm{w_H}(e) \leq \frac{n-\hat{k}}{2}$, then $\Lambda(x) = \Lambda_z(x)\Lambda_e(x)$ is the nonzero polynomial of the smallest degree (unique up to a scale factor) that satisfies

$$\deg\Big(Y(x)\Lambda(x) \bmod m(x)\Big) < \frac{n + \hat{k}}{2} \tag{243}$$

and is a multiple of $\Lambda_z(x)$. □

We can thus run Algorithm 6 (with $b(x) = Y(x)$ and $\Gamma(x) = \Lambda_z(x)$) to compute $\Lambda(x) = \Lambda_z(x)\Lambda_e(x)$. Decoding can be completed as above, e.g., by (114).

C. Method III

This method is particularly obvious in the setting of this paper. Erased symbols may be viewed as missing evaluation points, which amounts to omitting the corresponding factors from (105). Let

$$\tilde{m}(x) \stackrel{\triangle}{=} m(x)/\Lambda_z(x), \tag{244}$$

which has degree $\tilde{n} \stackrel{\triangle}{=} n - |\mathcal{Z}|$, and let

$$\tilde{Y}(x) \stackrel{\triangle}{=} Y(x) \bmod \tilde{m}(x). \tag{245}$$

We can then run any of the algorithms of Section IV with $\tilde{Y}(x)$, $\tilde{m}(x)$ and $d = \lceil \frac{\tilde{n}+k}{2} \rceil$ as input. If $\mathrm{w_H}(e) \leq (\tilde{n} - k)/2$, the partial-inverse algorithm will return $\Lambda(x) = \gamma\,\Lambda_e(x)$ for some nonzero $\gamma \in F$.

Decoding can be completed by

$$C(x) = \frac{\tilde{Y}(x)\Lambda(x) \bmod \tilde{m}(x)}{\Lambda(x)} \tag{246}$$

according to Proposition 8 or by

$$C(x) = \tilde{Y}(x) \bmod \big(\tilde{m}(x)/\Lambda(x)\big) \tag{247}$$

according to Proposition 9.

APPENDIX F
JOINT ERRORS-AND-ERASURES DECODING OF POLYNOMIAL REMAINDER CODES

Errors-and-erasures decoding as in Appendix E can be generalized to polynomial remainder codes as in Section VI. We focus here on the generalization of Method III of Appendix E; the corresponding generalization of Method I and of Method II is straightforward (see also [23]).

As in Appendix E, let $\mathcal{Z} \subset \{0, \ldots, n-1\}$ be the set of erased positions. We then define the erasure locator polynomial as

$$\Lambda_z(x) \stackrel{\triangle}{=} \prod_{\ell \in \mathcal{Z}} m_\ell(x). \tag{248}$$

We further define

$$\tilde{m}(x) \stackrel{\triangle}{=} m(x)/\Lambda_z(x), \tag{249}$$

which has degree $\tilde{N} \stackrel{\triangle}{=} N - \deg \Lambda_z(x)$. We also define the modified error factor polynomial

$$\tilde{\Lambda}_E(x) \stackrel{\triangle}{=} \tilde{m}(x)/\gcd\big(E(x), \tilde{m}(x)\big), \tag{250}$$

which is a nonzero polynomial of the smallest degree that satisfies

$$E(x)\tilde{\Lambda}_E(x) \bmod \tilde{m}(x) = 0. \tag{251}$$

Note that $\tilde{\Lambda}_E(x)$ divides (158), and thus

$$\deg \tilde{\Lambda}_E(x) \le \deg \Lambda_E(x). \tag{252}$$

Now let

$$\tilde{Y}(x) \stackrel{\triangle}{=} Y(x) \bmod \tilde{m}(x). \tag{253}$$

We then have the analog of Theorem 10.

Theorem 14: For given y and e with $\deg \tilde{\Lambda}_E(x) \le t \le \frac{\tilde{N}-K}{2}$, assume that some nonzero polynomial $\Lambda(x)$ with $\deg \Lambda(x) \le t$ satisfies

$$\deg\big(\tilde{Y}(x)\Lambda(x) \bmod \tilde{m}(x)\big) < \tilde{N} - t. \tag{254}$$

Then $\Lambda(x)$ is a multiple of $\tilde{\Lambda}_E(x)$. Conversely, $\Lambda(x) = \tilde{\Lambda}_E(x)$ is a polynomial of the smallest degree that satisfies (254). □

The proof is easily adapted from the proof of Theorem 10.

We can thus run any of the algorithms of Section IV with $\tilde{Y}(x)$, $\tilde{m}(x)$ and $d = \lceil \frac{\tilde{N}+k}{2} \rceil$ as input. If

$$\deg \tilde{\Lambda}_E(x) \le (\tilde{N} - K)/2 \tag{255}$$

or, equivalently, if

$$2 \deg \tilde{\Lambda}_E(x) + \deg \Lambda_z(x) < N - K, \tag{256}$$

then the partial-inverse algorithm will return $\Lambda(x) = \gamma\, \Lambda_E(x)$ for some nonzero $\gamma \in F$. Decoding can be completed, e.g., by (246) or by (247).

REFERENCES

[1] J.-H. Yu and H.-A. Loeliger, "Reverse Berlekamp-Massey decoding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 1212–1216.

[2] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Oct. 1962.

[3] E. R. Berlekamp, *Algebraic Coding Theory*. New York, NY, USA: McGraw-Hill, 1968.

[4] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory*, vol. 15, no. 1, pp. 122–127, May 1969.

[5] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Inf. Control*, vol. 27, no. 1, pp. 87–99, Jan. 1975.

[6] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[7] R. E. Blahut, *Algebraic Codes on Lines, Planes, and Curves*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[8] R. M. Roth, *Introduction to Coding Theory*. New York, NY, USA: Cambridge Univ. Press, 2006.

[9] R. J. McEliece and J. B. Shearer, "A property of Euclid's algorithm and an application to Padé approximation," *J. Appl. Math.*, vol. 34, pp. 611–615, Jun. 1978.

[10] J. Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed. Cambridge, U.K.: Cambridge Univ, Press, 2013.

[11] S. Gao, "A new algorithm for decoding Reed–Solomon codes," in *Communications, Information and Network Security*, vol. 712. V. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, Eds. Norwell, MA, USA: Kluwer, 2003, pp. 55–68.

[12] A. Shiozaki, "Decoding of redundant residue polynomial codes using Euclid's algorithm," *IEEE Trans. Inf. Theory*, vol. 34, no. 5, pp. 1351–1354, Sep. 1988.

[13] M. Bossert and S. Bezzateev, "A unified view on known algebraic decoding algorithms and new decoding concepts," *IEEE Trans. Inf. Theory*, vol. 59, no. 11, pp. 7320–7336, Nov. 2013.

[14] J. L. Dornstetter, "On the equivalence between Berlekamp's and Euclid's algorithms (Corresp.)," *IEEE Trans. Inf. Theory*, vol. 33, no. 3, pp. 428–431, May 1987.

[15] A. E. Heydtmann and J. M. Jensen, "On the equivalence of the Berlekamp–Massey and the Euclidean algorithms for decoding," *IEEE Trans. Inf. Theory*, vol. 46, no. 7, pp. 2614–2624, Nov. 2000.

[16] M. Bras-Amorós and M. E. O'Sullivan, "From the Euclidean algorithm for solving a key equation for dual Reed–Solomon codes to the Berlekamp–Massey algorithm," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes* (Lecture Notes in Computer Science), vol. 5527, M. Bras-Amorós and Høholdt, Eds. New York, NY, USA: Springer, Jun. 2009, pp. 32–42.

[17] T. D. Mateer, "On the equivalence of the Berlekamp–Massey and the Euclidean algorithms for algebraic decoding," in *Proc. 12th Can. Workshop Inf. Theory*, Kelowna, BC, Canada, May 2011, pp. 139–142.

[18] L. R. Welch and R. A. Scholtz, "Continued fractions and Berlekamp's algorithm," *IEEE Trans. Inf. Theory*, vol. 25, no. 1, pp. 19–27, Jan. 1979.

[19] U. Cheng, "On the continued fraction and Berlekamp's algorithm (Corresp.)," *IEEE Trans. Inf. Theory*, vol. 30, no. 3, pp. 541–544, May 1984.

[20] J. J. Stone, "Multiple-burst error correction with the chinese remainder theorem," *J. Soc. Ind. Appl. Math.*, vol. 11, no. 1, pp. 74–81, Mar. 1963.

[21] J.-H. Yu and H.-A. Loeliger, "On irreducible polynomial remainder codes," in *Proc. IEEE Int. Symp. Inf. Theory* Saint Petersburg, Russia, Jul./Aug. 2011, pp. 1190–1194.

[22] J.-H. Yu and H.-A. Loeliger. (Jan. 2012). "On polynomial remainder codes." [Online]. Avaliable: http://arxiv.org/abs/1201.1812

[23] J.-H. Yu. (Feb. 2012). "On the joint error-and-erasure decoding for irreducible polynomial remainder codes." [Online]. Avaliable: http://arxiv.org/abs/1202.5413

[24] T. Horiguchi, "High-speed decoding of BCH codes using a new error-evaluation algorithm," *Electron. Commun. Jpn.*, vol. 72, no. 12, pp. 63–72, 1989.

[25] J.-H. Yu and H.-A. Loeliger, "An algorithm for simultaneous partial inverses," in *Proc. 52nd Annu. Allerton Conf. Commun. Control, Comput.* Monticello, IL, USA, Oct. 2014, pp. 928–935.

[26] J.-H. Yu and H.-A. Loeliger, "Decoding of interleaved Reed-Solomon codes via simultaneous partial inverses," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)* Hong Kong, Jun. 2015, pp. 2396–2400.

[27] P. Fitzpatrick, "On the key equation," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1290–1302, Sep. 1995.

[28] T. K. Truong, I. S. Hsu, W. L. Eastman, and I. S. Reed, "Simplified procedure for correcting both errors and erasures of Reed-Solomon code using Euclidean algorithm," *IEE Proc. E, Comput. Digit. Techn.*, vol. 135, no. 6, pp. 318–324, Nov. 1988.

[29] T.-C. Lin, P.-D. Chen, and T.-K. Truong, "Simplified procedure for decoding nonsystematic Reed-Solomon codes over gf(2^m) using euclid's algorithm and the fast Fourier transform," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1588–1592, Jun. 2009.

Jiun-Hung Yu (S'10–M'14) was born in Nantou, Taiwan, in 1979. He received the M.S. degree in communication engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2003, and the Ph.D. degree in electrical engineering from ETH Zurich, in 2014. From 2003 to 2008, he was with Realtek Semiconductor Cooperation, Hsinchu. Since 2008, he has been with the Signal and Information Processing Laboratory of ETH Zurich. His research interests include communication theory, error-correcting codes, and statistical signal processing.

Hans-Andrea Loeliger (S'85–M'92–SM'03–F'04) has been a full professor at ETH Zurich since 2000. He received both a diploma in electrical engineering and a PhD (in 1992) from ETH Zurich. From 1992 to 1995, he was with Linkoping University, Sweden. From 1995 to 2000, he was a full-time technical consultant and coowner of a consulting company. His research interests lie in the broad areas of signal processing, statistical models, information theory, error correcting codes, communications, system theory, and electronics.