Some Cryptanalytic and Coding-Theoretic Applications of a Soft Stern Algorithm

Guo, Qian; Johansson, Thomas; Mårtensson, Erik; Stankovski, Paul

Published in:
Advances in Mathematics of Communications

DOI:
10.3934/amc.2019035

2019

Document Version:
Version created as part of publication process; publisher's layout; not normally made publicly available

Manuscript submitted to
AIMS' Journals
Volume **X**, Number **0X**, XX **201X**

doi:10.3934/xx.xx.xx.xx

pp. **X–XX**

SOME CRYPTANALYTIC AND CODING-THEORETIC APPLICATIONS OF A SOFT STERN ALGORITHM

Qian Guo

Selmer Center, Department of Informatics, University of Bergen
Postboks 7803, N-5020 Bergen, Norway

Thomas Johansson, Erik Mårtensson* and Paul Stankovski Wagner

Department of Electrical and Information Technology, Lund University
Box 118, SE-22100 Lund, Sweden

Abstract. Using the class of information set decoding algorithms is the best known way of decoding general codes, i.e. codes that admit no special structure, in the Hamming metric. The Stern algorithm is the origin of the most efficient algorithms in this class. We consider the same decoding problem but for a channel with soft information. We give a version of the Stern algorithm for a channel with soft information that includes some novel steps of ordering vectors in lists, based on reliability values. We demonstrate how the algorithm constitutes an improvement in some cryptographic and coding theoretic applications. We also indicate how to extend the algorithm to include multiple iterations and soft output values.

1. **Introduction.** For a general code with no special structure used for communication on the binary symmetric channel (BSC), the maximum-likelihood decoding problem (with some assumptions) is NP-hard. Still, decoding random linear codes is a central problem for many applications in cryptography, for example code-based crypto. Information set decoding (ISD) algorithms are the most promising candidates for solving instances of this problem.

The performance of these algorithms determines the security and hence the necessary parameters for many cryptosystems. The development of ISD algorithms include the Prange algorithm [23], the Lee-Brickell algorithm [18], the Stern algorithm [25], Canteaut-Chabaud [8], Ball-Collision Decoding [6], Finiasz-Sendrier [13], BJMM [3] and the recent improvement from May and Ozerov [19]. The Stern algorithm is the starting point for the most efficient algorithms in this class as it introduced a collision step that significantly decreased the complexity.

In this paper, we consider the decoding problem for a general code with no special structure used for communication on the Additive White Gaussian Noise (AWGN)

2010 *Mathematics Subject Classification.* Primary: 94A60; Secondary: 68P30.

Key words and phrases. Soft decoding, Stern algorithm, Side-channel attacks, Information set decoding, Soft Stern algorithm.

Part of the material in this paper was presented at the 2017 IEEE International Symposium on Information Theory (ISIT 2017), Aachen, Germany, June 25-30, 2017.

This work was supported in part by the Swedish Research Council (Grant No. 2015-04528). The first author was also supported in part by the Norwegian Research Council (Grants No. 247742/070).

* Corresponding author.

channel using the Euclidean metric. This is motivated by the fact that we have seen some recent applications for such decoding algorithms in coding theory and cryptography. One such application is the recently proposed version of the McEliece Public Key Cryptosystem (PKC) using soft information [2]. Another is the use of such algorithms in side-channel cryptanalysis, see, e.g., [22]. A third one is a new hybrid decoding of low-density parity-check (LDPC) codes in space telecommand links [1].

The soft decoding problem has been studied extensively in coding and communication, see, e.g., [10, 17], but mostly for special codes allowing efficient decoding. The study of general codes has been less intense. Early work by Chase [9] was followed by some work in the communication direction and a highly cited paper is [14]. More recently, fast soft-decision decoding of linear codes was considered in [1, 11, 26, 28] and by Wu and Hadjicostis in [30]. The same problem considered in the context of side-channel cryptanalysis in cryptology can be found in [4, 5, 12].

In this paper, we give a version of the Stern algorithm for the decoding problem with soft information, named the soft Stern algorithm. The algorithm reuses some ideas from previous work, such as ordered statistics [14]. It uses the idea of sorting of error vectors in lists, based on reliability values [27], and presents a novel way of combining this idea with the structure of the Stern algorithm. This leads to better performance compared to previously suggested algorithms like the one in [22]. Initially we consider a one-pass algorithm that succeeds with some probability q. We can then repeat this one-pass algorithm to achieve a higher success probability, where the way it is repeated depends on the application. Later, we briefly consider extending the algorithm to also allow for multiple iterations.

Next, we demonstrate how this new algorithm can be used in cryptographic and coding theory applications. First, we present a very efficient attack on an idea of using soft information in McEliece-type cryptosystems presented at ISIT 2016 [2]. Not only do we severely break the proposed schemes, but our algorithm shows that the whole idea of using soft information in this way is not fruitful. Secondly, we show how our algorithm can be applied to side-channel attacks. The problem of soft decoding of general codes appears in side-channel attacks in both [21] and [22]. Using our algorithm, both of those attacks can be improved. Thirdly, we show how our algorithm can be used to improve the hybrid decoding of low-density parity-check (LDPC) codes [1]. Finally, we indicate that by using soft output, our algorithm can be applied to the problem of decoding product codes.

The remaining parts of the paper are organized as follows. In Section 2 we give some preliminaries on coding theory and the considered channel. Section 3 gives an overview of the new algorithm, and in Section 4 we give a complete example of the algorithm. In Section 5 we analyze its time complexity and give simulation results demonstrating the improvement compared to previously proposed algorithms. In Section 6 we indicate how to generalize our algorithm by allowing for multiple iterations and soft output. In Section 7 we cover different applications of the algorithm. Finally, Section 8 concludes the paper.

2. **Preliminaries.**

We present some basic concepts in coding theory. Let $\mathbb{F}_2$ denote the binary finite field, $|x|$ the absolute value of x for $x \in \mathbb{R}$, and $\ln(\cdot)$ the logarithm with base e. Let π be a permutation of $\{1, \ldots, n\}$ and π^{-1} be its inverse. For a matrix $\mathbf{G}$, we let $\pi(\mathbf{G})$ denote the matrix obtained from $\mathbf{G}$ by permuting its column indices according to π.

2.1. **Basics in Coding Theory.**

Definition 1. An $[n, k]$ binary linear code $\mathcal{C}$ is a k-dimensional vector subspace of $\mathbb{F}_2^n$. Its co-dimension is $r = n - k$, characterizing the redundancy of the code.

A generator matrix $\mathbf{G}$ of the linear code $\mathcal{C}$ is defined as a $k \times n$ matrix in $\mathbb{F}_2^{k\times n}$ whose rows form a basis of the code. Equivalently, the code can be defined by a matrix $\mathbf{H}$ in $\mathbb{F}_2^{r\times n}$ whose kernel is the code $\mathcal{C}$, called a parity-check matrix of $\mathcal{C}$. For a length n vector $\mathbf{v}$, the support $\mathrm{supp}(\mathbf{v})$ is defined as $\{i : v_i \neq 0, 1 \leq i \leq n\}$. The Hamming weight of $\mathbf{v}$ is $w_H(\mathbf{v}) = |\{i : v_i \neq 0, 1 \leq i \leq n\}|$ and the Hamming distance is $d_H(\mathbf{v}, \mathbf{v}') = w_H(\mathbf{v} + \mathbf{v}')$.

Suppose an $[n, k]$ binary linear code $\mathcal{C}$ with generator matrix $\mathbf{G}$ is used for transmission on the AWGN channel. Let $\mathbf{c} = (c_1, c_2, \cdots, c_n)$ be a codeword to be transmitted. In Binary Phase-Shift Keying (BPSK) transmission, the codeword $\mathbf{c}$ is mapped to a bipolar sequence $\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, \cdots, \hat{c}_n)$, where $\hat{c}_i \in \mathbb{R}$ through $\hat{c}_i = (-1)^{c_i}$, for $1 \leq i \leq n$. For any binary vector $\mathbf{x}$, we use the notation $\hat{\mathbf{x}}$ to denote the result after applying the above mapping.

After transmission, where AWGN noise is added, the received vector is denoted $\mathbf{r} = (r_1, r_2, \cdots, r_n)$, $r_i \in \mathbb{R}$ for $1 \leq i \leq n$, where $r_i = \hat{c}_i + w_i$ and w_i are iid Gaussian random variables with zero mean and standard deviation σ. Since the values are floating-point, we say that we have soft information. If the noise would be binary, we would have worked with hard information. If we would translate each value of the $\mathbf{r}$ vector to its most probably binary value in $\mathbf{c}$, we we would make a so called hard decision.

In the continuation, when discussing the reliability value of a position we refer to r_i. When discussing how reliable a position is we refer to the absolute value of r_i.

Our soft-decision decoding problem is now the following: Find the most likely codeword being transmitted when receiving $\mathbf{r}$. We consider maximum-likelihood decoding (MLD). It is well known that the MLD metric becomes the squared Euclidean distance and that the codeword $\mathbf{c}$ closest to a received vector $\mathbf{r}$ is the one that minimizes the distance $D(\hat{\mathbf{c}}, \mathbf{r}) = \sum_{i=1}^{n}(r_i - \hat{c}_i)^2$ (see, e.g., [29]).

For binary codes, it is common to use the log likelihood ratio (LLR), which is defined as

$$L_i = \ln\left[\frac{p(r_i|c_i = 0)}{p(r_i|c_i = 1)}\right],$$

where $p(r_i|c_i)$ is the pdf of r_i conditioned on c_i. After some calculations one can rewrite this for the AWGN channel as

$$L_i = \frac{2r_i}{\sigma^2}.$$

We point out that we actually only need soft information in LLR form and the algorithm to be proposed works for any noise distribution, not just AWGN.

Finally, we introduce a class of codes for later use.

Definition 2. A low density parity-check (LDPC) code is a linear code admitting a sparse parity-check matrix, while a moderate density parity-check (MDPC) code is a linear code with a denser but still sparse parity-check matrix.

In previous work, the Hamming weight of the row vector is usually employed to characterize its sparsity; LDPC codes have small constant row weights, MDPC codes have row weights $O(\sqrt{n \log n})$. These classes of codes are of interest since they

Algorithm 1 The Stern algorithm

Input: Generator matrix $\mathbf{G}$, parameters p, l

1. Choose a column permutation π and form $\pi(\mathbf{G})$, meaning that the columns in $\mathbf{G}$ are permuted according to π.
2. Bring the generator matrix $\pi(\mathbf{G})$ to systematic form:
$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J})\,.$$
3. Let $\mathbf{z}$ run through all weight p vectors of length $k/2$. Store all vectors $\mathbf{x} = (\mathbf{z}, \mathbf{0})\mathbf{G}^*$ in a sorted list $\mathcal{L}_1$, sorted according to $\phi(\mathbf{x})$. Then construct a list $\mathcal{L}_2$ sorted according to $\phi(\mathbf{x})$, containing all vectors $\mathbf{x} = (\mathbf{0}, \mathbf{z})\mathbf{G}^*$ where $\mathbf{z}$ runs through all weight p vectors. Add all pairs of vectors $\mathbf{x} \in \mathcal{L}_1$ and $\mathbf{x}' \in \mathcal{L}_2$ for which $\phi(\mathbf{x}) = \phi(\mathbf{x}')$ and put in a list $\mathcal{L}_3$.
4. For each $\mathbf{x} \in \mathcal{L}_3$, check if the weight of $\mathbf{x}$ is $w-2p$. If no such codeword is found, return to 1.

are efficiently decodable using iterative decoding techniques exploiting the sparsity of the codes.

The class of quasi-cyclic MDPC (QC-MDPC) codes are of special interest as they are used in the QC-MDPC code-based McEliece cryptosystem [20]. The codes used in the cryptosystem are linear codes with sparse parity-check matrices of the form,

$$\mathbf{H} = (\mathbf{H}_0\ \mathbf{H}_1\ \ldots\ \mathbf{H}_{n_0-1})\,, \tag{1}$$

where n_0 is a small integer and each block $\mathbf{H}_i$, $0 \le i \le n_0 - 1$, is a circulant matrix with size $r \times r$, and $\mathbf{H}_{n_0-1}$ is invertible. For simplicity, we assume that $n_0 = 2$ throughout the paper, unless otherwise specified. Thus, we consider codes with rate $R = 1/2$, length $n = 2r$, and dimension $k = r$.

2.2. **Soft McEliece.** Soft McEliece [2] is a recent code-based McEliece PKC proposal using soft information. Instead of generating intentional errors from a Hamming ball, the authors generate noise according to a Gaussian distribution. In the key generation, as in the QC-MDPC scheme, they generate a sparse parity-check matrix $\mathbf{H}$ with the form of (1) and use it as the secret key. The public key can be derived as the (dense) generator matrix $\mathbf{G}$ in systematic form corresponding to $\mathbf{H}$.

Given a message $\mathbf{u} \in \mathbb{F}_2^k$, let $\mathbf{c} = \mathbf{uG}$ be the encoded codeword and $\hat{\mathbf{c}}$ the codeword in $\mathbb{R}^n$. The ciphertext is

$$\mathbf{r} = \hat{\mathbf{c}} + \mathbf{w},$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ and w_i $(1 \le i \le n)$ is AWGN. The generation of $\mathbf{w}$ is repeated until the number of bit errors in $\mathbf{r}$ reaches a certain threshold. The decryption – decoding the received vector – can be performed using an iterative soft LDPC/MDPC decoder that uses the secret $\mathbf{H}$, see [2, 20].

2.2.1. *Parameter settings.* In [2], the authors suggested parameters $(n, \sigma) = (7202, 0.44091)$ for 80-bit security, and $(15770, 0.41897)$ for 128-bit security. The complexity of decoding a received vector $\mathbf{r}$ knowing only the public generator matrix $\mathbf{G}$ must be larger than 2^{80} and 2^{128}, respectively.

2.3. **The Stern Algorithm.** The Stern algorithm finds a low weight codeword of Hamming weight w in the code described by $\mathbf{G}$. Transform the generator matrix $\mathbf{G}$ to systematic form with generator matrix

$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J}),$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{Q}$ is a $k \times l$ matrix and $\mathbf{J}$ is a $k \times (n-k-l)$ matrix. Let $\phi(\mathbf{x})$ be the value of a vector $\mathbf{x}$ in positions $k+1$ to $k+l$, i.e. $\phi(\mathbf{x}) = (x_{k+1}, x_{k+2}, \cdots, x_{k+l})$. The algorithm description is given in Algorithm 1.

3. A Soft Version of the Stern Algorithm.

We now present as the main contribution a version of the Stern algorithm that uses soft information.

3.1. A One-Pass Soft Stern Algorithm.

Receiving the vector $\mathbf{r}$, one can obtain a binary vector by making bitwise hard decisions. We define

$$\operatorname{sgn}(r_i) = \begin{cases} 1, & \text{if } r_i \leq 0, \\ 0, & \text{otherwise.} \end{cases}$$

Assuming that c_i is uniformly distributed over $\mathbb{F}_2$, according to Bayes' law, the conditional probability $\mathbf{Pr}\left[c_i = \operatorname{sgn}(r_i) | r_i\right]$, denoted p_i, can be written as

$$p_i = \frac{1}{1 + e^{-|L_i|}}. \tag{2}$$

Also, define the bias as $\tau_i = |p_i - 1/2|$. The problem of recovering the message from a ciphertext is solved by finding a minimum-weight codeword from a linear code with a generator matrix

$$\begin{pmatrix} \mathbf{G} \\ \operatorname{sgn}(r_1), \operatorname{sgn}(r_2), \ldots, \operatorname{sgn}(r_n) \end{pmatrix}.$$

This would, however, give a poor performance compared to what can be achieved when we use the soft information. Instead, we suggest to use the Stern algorithm as a basis and to modify the different steps to make use of the soft information in the best possible way. Initially, we consider only a single round in this algorithm, which will give a (small) probability q of success. In many (cryptographic) applications this is sufficient as one might repeat the decoding attempt many times and thus achieve an expected complexity which is a factor $1/q$ larger than the complexity of a single round. Later on, in Section 6.2, we indicate how to extend the algorithm to allow for multiple iterations.

The new algorithm can be divided into three steps in the following way:

3.1.1. *Transformation.* This step performs a column permutation and some transformations. Instead of selecting a random column permutation as in the original Stern algorithm, we consider only a single round and we use a permutation that puts the most reliable positions as the $k+l$ first columns. These columns will correspond to the information set and l additional positions.

Firstly, all the n coordinates r_i are sorted according to the absolute value of their LLR and then we choose a set $\mathcal{S}$ containing the $k+l$ most significant coordinates. Denote the set containing the other positions by $\mathcal{O}$. We use π to denote a permutation such that $\pi(\mathcal{S}) = \{1, \ldots, k+l\}$.

The second condition on π is that the first k columns of $\pi(\mathbf{G})$ are independent, forming a basis. We then derive a systematic generator matrix $\mathbf{G}^*$ from $\mathbf{G}$ by permuting the columns using π and performing Gaussian elimination, giving

$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J}),$$

where $\mathbf{Q}$ is a $k \times l$ matrix. The received vector $\mathbf{r}$ is permuted accordingly, giving vector $\pi(\mathbf{r})$. The k first positions are now an information set, denoted $\mathcal{I}$.

We next perform a transformation to ensure that the reliability value for each variable in the information set is positive. We first determine the most likely value for the variables in the information set, denoted by $\mathbf{m}$, where $m_i = \mathrm{sgn}(r_{\pi^{-1}(i)})$, for $1 \le i \le k$. This $\mathbf{m}$ corresponds to the codeword $\mathbf{c}' = \mathbf{m}\mathbf{G}^*$. Then the vector $\pi(\mathbf{r})$ is transformed to the vector $\mathbf{r}' = (r'_1, \ldots, r'_n)$, where

$$r'_i = r_{\pi^{-1}(i)} \cdot (-1)^{c'_i}, \quad 1 \le i \le n. \tag{3}$$

We have the following proposition.

Proposition 1. If $D(\hat{\mathbf{c}}, \mathbf{r}) = \delta$, then $D(\hat{\mathbf{c}''}, \mathbf{r}') = \delta$, where $\mathbf{c}'' = \mathbf{c} + \mathbf{c}'$.

Therefore, the transformation has not changed the problem, but the first k positions now all have positive reliability, which may ease the description in the continuation.

For the next step, we will consider the shortened code from $(\mathbf{I}\ \mathbf{Q})$ and try to find a list of codeword candidates close to $\mathbf{r}'$ in the first $k+l$ positions. For columns with indices in $\{k+1, \ldots, k+l\}$ corresponding to the matrix $\mathbf{Q}$ in $\mathbf{G}^*$, we determine a syndrome $\mathbf{s}$ by $s_i = \mathrm{sgn}(r'_{k+i})$, for $1 \le i \le l$.

Codewords for the shortened code are vectors $\mathbf{c}^{(\mathrm{S})}$ such that $\mathbf{c}^{(\mathrm{S})}\mathbf{H}' = \mathbf{0}$, where $\mathbf{H}' = \begin{pmatrix} \mathbf{Q} \\ \mathbf{I}_l \end{pmatrix}$. As we change the signs of position $k+1, \ldots, k+l$ to be all positive when we introduced the syndrome, our problem is finding the most probable low weight vectors $\mathbf{z}$ such that $\mathbf{z}\mathbf{H}' = \mathbf{s}$, assuming that the reliability values in position $1, \ldots, k+l$ are all positive, i.e., assuming $r'_i \ge 0$, for $1 \le i \le k+l$.

We next partition the set $\pi(\mathcal{S}) = \{1, \ldots, k+l\}$ into two disjoint equal-sized parts, $\mathcal{S}_1$ and $\mathcal{S}_2$, such that

$$\prod_{i \in \mathcal{S}_1} p_i \approx \prod_{j \in \mathcal{S}_2} p_j,$$

where $p_i = \mathbf{Pr}\left[c_i^{(\mathrm{S})} = 0 | r'_i\right]$ as in (2). For simplicity, we assume that $\mathcal{S}_1 = \{1, \ldots, (k+l)/2\}$ and $\mathcal{S}_2 = \{(k+l)/2+1, \ldots, (k+l)\}$. In the algorithm, this is yet another condition to consider when selecting π. In the original Stern algorithm the choice of indices for the two sets does not influence the performance, but for the soft case it does, and this is the reason for the above condition.

3.1.2. *Creating Bit Patterns and Partial Syndromes.* We now create the most probable (low weight error words) $\mathbf{z}^{(1)}$ having nonzero values only in $\mathcal{S}_1$. We store the corresponding partial syndrome for the code with transposed parity check matrix $\mathbf{H}'$, created as $(\mathbf{z}^{(1)}, \mathbf{0})\mathbf{H}'$. As all reliability values are positive, the zero word is the most likely one, then different vectors of weight one, etc. Let $\mathbf{z}^{(1)}$ run through T length-$(k+l)/2$ binary vectors with the largest probability $\prod_{i \in \mathcal{S}_1} \mathbf{Pr}\left[c_i^{(\mathrm{S})} = z_i^{(1)} | r'_i\right]$. We build a table $\mathcal{L}_1$ to store all selected $\mathbf{z}^{(1)}$ together with the vector $(\mathbf{z}^{(1)}, \mathbf{0})\mathbf{H}'$. The table $\mathcal{L}_1$ is sorted according to this partial syndrome.

Algorithm 2 The Soft Stern algorithm

Input: Generator matrix $\mathbf{G}$, received vector $\mathbf{r}$, parameters $T = 2^l, \delta$

Step 1:

(1a) Choose a column permutation π such that 1) the first $k+l$ positions in $\pi(\mathbf{G})$ have the $k+l$ largest $|r_i|$'s and 2) the first k columns are independent and 3) $\prod_{i=1,2,\dots(k+l)/2} p_i \approx \prod_{i=(k+l)/2+1,\dots(k+l)} p_i$.

(1b) Make the permuted generator matrix $\pi(\mathbf{G})$ systematic:

$$\mathbf{G}^* = (\mathbf{I}\ \mathbf{Q}\ \mathbf{J})\,.$$

Permute and transform the received sequence $\mathbf{r}$ to make the reliability value for each coordinate in positions $1, 2, \dots k$ positive, following (3), giving $\mathbf{r}'$.

(1c) Calculate the corresponding partial syndrome $\mathbf{s}$ and change the sign of any negative values of $r'_{k+1}, \dots r'_{k+l}$.

Step 2: Construct a list $\mathcal{L}_1$ storing the most probable vectors $\mathbf{z^{(1)}}$ and the corresponding partial syndromes $(\mathbf{z^{(1)}}, \mathbf{0})\mathbf{H}'$. Then construct another list $\mathcal{L}_2$ storing the most probable vectors $\mathbf{z^{(2)}}$ and the corresponding partial syndromes $\mathbf{s} + (\mathbf{0}, \mathbf{z^{(2)}})\mathbf{H}'$.

Step 3: Sort the two lists according to their partial syndromes and search for collisions. For each colliding syndrome $(\mathbf{z^{(1)}}, \mathbf{0})\mathbf{H}'$ and $\mathbf{s} + (\mathbf{0}, \mathbf{z^{(2)}})\mathbf{H}'$, create a new vector $\mathbf{u}$ by choosing the first k entries of $(\mathbf{z^{(1)}}, \mathbf{z^{(2)}})$ and compute the corresponding $\hat{\mathbf{c}} = (\hat{c}_1, \dots, \hat{c}_n)$, s.t. $\hat{c}_i = (-1)^{c_i}$, where $\mathbf{c} = \mathbf{u}\mathbf{G}^*$.

If $D(\hat{\mathbf{c}}, \mathbf{r}') \leq \delta$, invert the transformations to get the codeword close to the original $\mathbf{r}$ and return it. If no $\mathbf{c}$ with $D(\hat{\mathbf{c}}, \mathbf{r}') \leq \delta$ is found, return failure.

We now repeat the same thing but for the subset $\mathcal{S}_2$, creating another table $\mathcal{L}_2$ in a similar manner. In this case we run through the most probable vectors $\mathbf{z}^{(2)}$ with nonzero positions only in $\mathcal{S}_2$. Each entry in the table consists of $\mathbf{z}^{(2)}$ together with the partial syndrome $\mathbf{s} + (\mathbf{0}, \mathbf{z^{(2)}})\mathbf{H}'$ sorted according to the latter. Note that we add $\mathbf{s}$ in this case.

3.1.3. *Colliding Partial Syndromes.* Next, we search for partial syndrome collisions between the tables $\mathcal{L}_1$ and $\mathcal{L}_2$. On average we obtain $T^2/2^l$ colliding vectors. Later we assume that we choose $T \approx 2^l$ to minimize time-complexity.

For each collision, we add the corresponding vectors $(\mathbf{z}^{(1)}, \mathbf{0})$ and $(\mathbf{0}, \mathbf{z^{(2)}})$, and create a new vector $\mathbf{u}$ by choosing its first k entries. Then we get a candidate codeword $\mathbf{u}\mathbf{G}^*$. As a final step, we check the Euclidean distance between each candidate codeword and the received vector $\mathbf{r}'$. If it is sufficiently small we return it as the desired closest codeword.

3.2. **How to Create the Most Probable Vectors.** In this part, we explain how to create the T most probable vectors required in Step 2 of the previous description.

Since the reliability values are all transformed to be positive, for the partitions $\mathcal{S}_m$, $m = 1, 2$, the most probable pattern is the all-zero vector $\mathbf{0}$, with probability $P_m = \prod_{i \in \mathcal{S}_m} p_i$. The probability for a pattern with ones in positions in $\mathcal{J}$ and the remaining positions all zero is $\exp(-\sum_{j \in \mathcal{J}} L_j) \cdot P_m$.

For $\mathcal{S}_1$, our problem can then be described as follows. Given $(k+l)/2$ positive real numbers $(L_1, \ldots, L_{(k+l)/2})$ which are sorted in increasing order, i.e., $L_1 \leq L_2 \leq \ldots \leq L_{(k+l)/2}$, define a function $f(\mathcal{I}) = \sum_{j \in \mathcal{I}} L_j$, where $\mathcal{I} \subset \{1, \ldots, \frac{(k+l)}{2}\}$. Our goal is to find the T different index sets $\mathcal{I}_i, 1 \leq i \leq (k+l)/2$ with smallest corresponding values $f(\mathcal{I}_i)$. The method for solving this problem is based on [27].

Let $I_{i_1,i_2,\ldots,i_k}$, where $i_1 < i_2 < \cdots i_k$, denote the bit pattern with the value 1 in positions $i_1, i_2, \ldots, i_k$, and the value zero in the other positions. For such a bit pattern, its function value is again $f(\mathcal{I}) = \sum_{j \in \mathcal{I}} L_j$, where $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$.

Now, let $\mathcal{R}_i$ denote the set of bit patterns with a 1 in position i and zeros in all positions after i. Sort the elements in $\mathcal{R}_i$ by its function values in increasing order to form the list R_i. Given a pattern $I \in R_i$, by the successor of I we mean the next pattern in R_i.

To solve our original problem we use a binary tree, where each node represents one of the lists $R_2, R_3, \ldots, R_{(k+l)/2}$. Initially, let the nodes store the top element in each list R_i, being the patterns $I_2, I_3, \ldots, I_{(k+l)/2}$, respectively. Also, let each node store an index value 0. The root of the tree will have the pattern with the smallest function value, which initially is I_2. Each parent node in the tree has a smaller function value than its child nodes.

Let A denote a list of the bit patterns we have found sofar, and their corresponding function values. Initialize this list with the all zeros pattern at index 0 and the pattern with a 1 in the first position at the index 1.

In each step of our algorithm we add the pattern of the root node to A. Assume the root node has the pattern $I_{i_1,i_2,\ldots,i_m,i}$. Then we replace the node label by the next pattern in the list R_i. This is found by starting in A at the index of the pattern $I_{i_1,i_2,\ldots,i_m}$ and finding the next pattern $I_{j_1,j_2,\ldots,j_n,i}$ in A such that $j_n < i$. If no such pattern exists, we have used all patterns in R_i and we can delete the node from the tree. Otherwise, we replace the node label by the pattern $I_{j_1,j_2,\ldots,j_n,i}$ and we also store the index in A of the pattern $I_{j_1,j_2,\ldots,j_n}$. In either case, we end by rearranging the tree such that each parent node has a smaller function value than its child nodes.

The most expensive part of the algorithm is rearranging the tree. This requires at most $\lceil \log_2(k+l)/2 \rceil$ function comparisons. If we store the function value for each pattern in A, calculating the function value for a new pattern in the tree only requires a single addition.

3.2.1. *Example of How to Find the Most Probable Vectors.* An example of how to find the most probable bit patterns is illustrated in Figure 1. In this case we work with vectors of length 8 and the corresponding 8 real values are

$$[0.1622, 0.1656, 0.2630, 0.3112, 0.5285, 0.6020, 0.6541, 0.7943].$$

For the sake of clarity we work with the whole bit patterns, but storing only the indices of the positions with the value 1 is of course more efficient. At the beginning the list $A = [(00000000, 0), (10000000, 0.1622)]$. In each step we add the root node and its corresponding function value to A. We use the index of the root node to determine where in A we start looking for a new bit pattern. When we have found

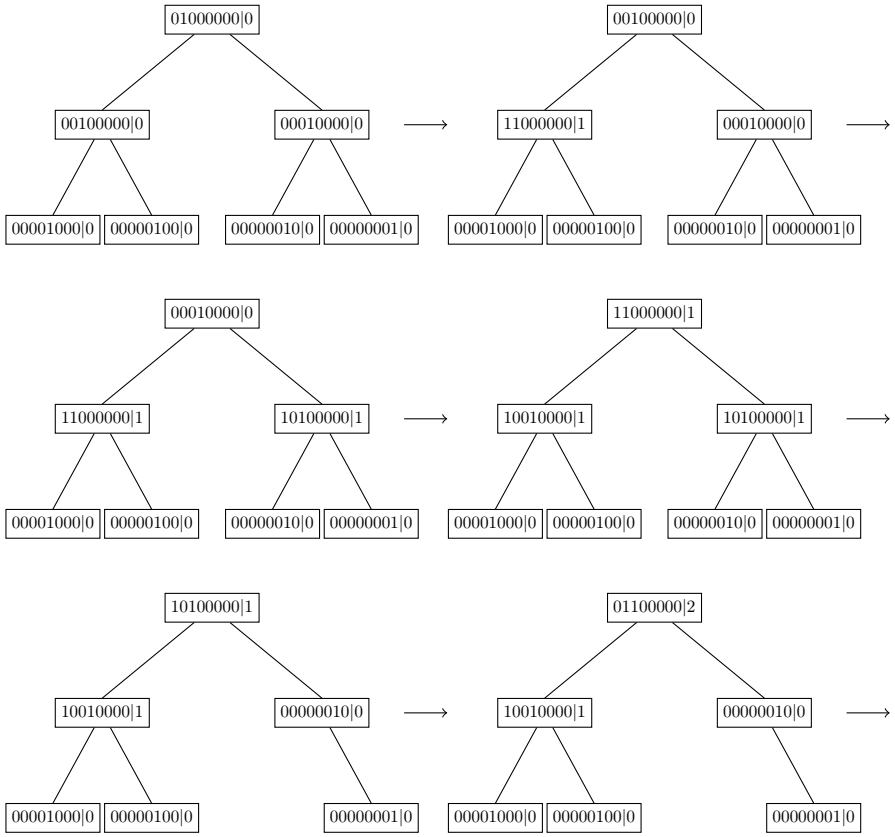

FIGURE 1. An ilustration of what the binary tree in the algorithm for finding the most probable bit patterns looks like in the first six steps.

the next pattern we modify the root node and rearrange the tree. Notice that the bit pattern 11000000 does not have a successor node. Therefore, after adding the pattern to A the corresponding node in the tree is removed. By the time we have reached the sixth binary tree in Figure 1 we have

$$A = [(00000000, 0.0000), (10000000, 0.1622), (01000000, 0.1656), (00100000, 0.2630), \\ (00010000, 0.3112), (11000000, 0.3278), (10100000, 0.4252)].$$

The bit patterns in A gives us the 7 most probable bit patterns. By looking at the root node we see that pattern number 8 is 01100000.

4. **A Decoding Example.** This section contains a complete example of how a message is encoded, how Gaussian noise is added, how the errors are corrected using the proposed Soft Stern algorithm, and finally how the original message is recovered. The extended, binary Golay code is a linear, systematic, error-correcting code with parameters $(n, k, d_{min}) = (24, 12, 8)$ and generator matrix $\mathbf{G}$ equal to

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1
\end{pmatrix}.$$

Assume sending the message $\mathbf{u} = [0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0]$, transforming each 1 to -1 and each 0 to 1, and adding Gaussian noise with $\sigma = 1$. In this example the received vector $\mathbf{r}$ is

$$\begin{array}{cccccccc}
[0.8437 & -0.8059 & 1.5800 & -0.1491 & 0.5741 & 0.6922 & 1.0734 & -1.9306 \\
-1.5678 & 1.1405 & 0.8998 & 2.6440 & 1.2390 & -0.6847 & 0.0724 & -0.2315 \\
0.1800 & -0.8032 & -1.3533 & -2.8248 & 0.1319 & 0.0888 & -1.2301 & -0.3469].
\end{array}$$

Let us use $l = 4$ and $T = 2^l = 2^4 = 16$. After performing a permutation of the positions such that the first $k + l$ are the most reliable, such that the first k columns in the generator matrix are linearly independent, and such that the first $k + l$ positions are split into two parts with approximately equal products of p_i values, we obtain an $\mathbf{r}^*$ vector equal to

$$\begin{array}{cccccccc}
[-1.2301 & 0.6922 & 0.8437 & -1.3533 & 1.1405 & 1.0734 & 2.6440 & -0.6847 \\
-1.5678 & 1.5800 & 0.8998 & -0.8059 & -2.8248 & -1.9306 & 1.2390 & -0.8032 \\
0.5741 & -0.3469 & -0.2315 & 0.1800 & -0.1491 & 0.1319 & 0.0888 & 0.0724].
\end{array}$$

The corresponding systematic generator matrix $\mathbf{G}^*$ is

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1
\end{pmatrix}.$$

Encoding the message $\mathbf{m}$, where each of the k positions of $\mathbf{m}$ are 1 if the corresponding position in $\mathbf{r}^*$ is positive and 0 otherwise, then changing the sign for each position in $\mathbf{r}^*$ where the corresponding position in the encoded vector $\mathbf{mG}^*$ is positive, results in an $\mathbf{r}'$ vector equal to

$$\begin{matrix}[1.2301 & 0.6922 & 0.8437 & 1.3533 & 1.1405 & 1.0734 & 2.6440 & 0.6847 \\ 1.5678 & 1.5800 & 0.8998 & 0.8059 & 2.8248 & 1.9306 & -1.2390 & -0.8032 \\ 0.5741 & 0.3469 & 0.2315 & 0.1800 & -0.1491 & -0.1319 & 0.0888 & -0.0724].\end{matrix}$$

We notice that the partial syndrome is $\mathbf{s} = [0\ 0\ 1\ 1]$ (by looking at the signs of positions $k+1$ to $k+l$). Picking the first $k+l$ values of $\mathbf{r}'$, switching signs to make all the values positive, calculating the corresponding LLR values, and sorting the LLR values such that each half of the values are in increasing order, gives a vector of LLR values equal to

$$\begin{matrix}[1.3694 & 1.3844 & 1.6875 & 2.1468 & 2.2811 & 2.4602 & 2.7066 & 5.2879 \\ 1.6063 & 1.6119 & 1.7996 & 2.4779 & 3.1356 & 3.1600 & 3.8611 & 5.6495].\end{matrix}$$

Picking the parity-check matrix corresponding to the first $k+l$ columns of $\mathbf{G}^*$, then permuting the columns according to the permutation done to sort the LLR values, results in the permuted parity-check matrix $\mathbf{H}$ equal to

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Using the first half of the LLR values to create the T most probable vectors on the form $(\mathbf{z}_1\ 0)$ and their corresponding syndromes $(\mathbf{z}_1\ \mathbf{0})\mathbf{H}^T$ we get the following list of vectors and syndromes

$$\mathbf{L}_1 = \left[\begin{array}{cccccccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}\right].$$

Using the last half of the LLR values to create the T most probable vectors on the form $(\mathbf{0}\ \mathbf{z}_2)$ and their corresponding syndromes $(\mathbf{0}\ \mathbf{z}_2)\mathbf{H}^T + \mathbf{s}$ we get the following list of vectors and syndromes

$$\mathbf{L}_2 = \left[\begin{array}{cccccccc|cccc} 0&0&0&0&0&0&0&0&0&0&1&1\\ 1&0&0&0&0&0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0&0&0&1&1&0\\ 0&0&1&0&0&0&0&0&1&1&0&1\\ 0&0&0&1&0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0&1&1&1&0\\ 0&0&0&0&0&1&0&0&1&0&0&0\\ 1&1&0&0&0&0&0&0&0&1&1&1\\ 1&0&1&0&0&0&0&0&1&1&0&0\\ 0&1&1&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0&0&1&1&1\\ 1&0&0&1&0&0&0&0&0&0&0&0\\ 0&1&0&1&0&0&0&0&0&1&0&0\\ 0&0&1&1&0&0&0&0&1&1&1&1\\ 1&0&0&0&1&0&0&0&1&1&1&1\\ 0&1&0&0&1&0&0&0&1&0&1&1 \end{array}\right].$$

Colliding these vectors we get the following list of possible candidates for a solution (where the first half of each row corresponds to $\mathbf{z}_1$ and the second half corresponds to $\mathbf{z}_2$)

$$\left[\begin{array}{cccccccc|cccccccc} 0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\ 1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0\\ 0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0\\ 1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\ 1&0&1&0&0&0&0&0&1&1&0&0&0&0&0&0\\ 1&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0\\ 0&1&0&1&0&0&0&0&1&1&0&0&0&0&0&0\\ 0&1&0&1&0&0&0&0&0&0&0&0&0&0&1&0\\ 1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0 \end{array}\right].$$

For each candidate we invert the permutation corresponding to the sorting of the LLR values. Then we pick the k first bits and create the message $\mathbf{u}_0$. We then encode the message using $\mathbf{G}^*$ to $\mathbf{c}_0 = \mathbf{u}_0\mathbf{G}^*$ and transform each 1 to -1 and each 0 to 1, creating $\hat{\mathbf{c}_0}$. Then we calculate the Euclidean distance between $\hat{\mathbf{c}_0}$ and $\mathbf{r}'$. The vector $\mathbf{c}_0$ that will lead us to the original message is probably the one with the smallest Euclidean distance. In this example the smallest Euclidean norm corresponds to candidate number 4.

Inverting the sorting step and picking the k first bits gives us

$$\mathbf{u}_0 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1].$$

We then get

$$\mathbf{c}_0 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1].$$

To get back to the solution to the original problem we flip the bits in $\mathbf{c}_0$ where the corresponding positions in $\mathbf{r}'$ and $\mathbf{r}^*$ differ in sign and get the vector

$$[1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0].$$

Then we invert the first transformation and get the vector

$$[0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1].$$

Now we notice that the first k positions in this vector are identical to the original message $\mathbf{u}$ and we have thus found the original message.

5. **Complexity Analysis and Simulations.** A suitable complexity measure is given by $C_{\text{one-pass}}/\mathbf{Pr}\,[\mathcal{A}]$, where $C_{\text{one-pass}}$ is the complexity of one pass of the algorithm and $\mathcal{A}$ represents the event that after the permutation and the transformation, the actual error pattern in the first $(k+l)$ positions is a summation of two vectors in the two lists, respectively (i.e., that the Soft Stern algorithm will find the correct message).

When estimating complexity for matrix operations, we note that we can inductively implement the vector/matrix multiplication of $\mathbf{vM}$ by adding a new vector to an already computed and stored vector $\tilde{\mathbf{v}}\mathbf{M}$, where $\text{supp}(\tilde{\mathbf{v}}) \subset \text{supp}(\mathbf{v})$ and $d_H(\tilde{\mathbf{v}}, \mathbf{v}) = 1$. Thus, $C_{\text{one-pass}}$ measured in simple bit-oriented operations is roughly given by $C_{\text{Gauss}} + 4T \cdot (n-k) + C_{\text{create}}$, where C_{Gauss} is the complexity of Gaussian elimination that usually equals $0.5nk^2$ and C_{create} is the complexity for creating these most probable vectors. From Section 3.2 we have that $C_{\text{create}} = 2T\lceil \log_2((k+l)/2) \rceil$. Notice that the cost of creating the lists is low compared to calculating the partial syndromes and colliding these.

The probability $\mathbf{Pr}\,[\mathcal{A}]$ is given by

$$\mathbf{Pr}\,[\mathcal{A}] = Q_l \cdot P^{(1)} \cdot P^{(2)},$$

where

$$P^{(i)} = P_i \sum_{\mathcal{J} \in \mathfrak{P}^{(i)}} \exp\left(- \sum_{j \in \mathcal{J}} L_j \right),$$

for $i = 1, 2$. Here Q_l is the probability that $k+l$ columns in a uniformly random, binary $k \times (k+l)$ matrix have full rank. Also, $\mathfrak{P}^{(i)}$ is the set containing the T index sets corresponding to the T different vectors in $\mathcal{L}_i$, for $i = 1, 2$. For the sizes of k used in this paper, with very good precision we have

$$Q_l = \prod_{i=1+l}^{\infty} (1 - 2^{-i}).$$

Here we have $Q_0 \approx 0.2888$, and for each new column that is added the probability of not getting a matrix with full rank is roughly halved. Thus the probability of getting a full rank matrix increases fast with l. The next subsection will try to estimate (the remaining factors of) the probability $\mathbf{Pr}\,[\mathcal{A}]$.

5.1. **Estimating and Simulating $\mathbf{Pr}[\mathcal{A}]$.** As $\mathbf{Pr}[\mathcal{A}]$ depends on the received vector $\mathbf{r}$, it appears to be quite complicated to provide a useful explicit expression or approximation for $\mathbb{E}(\mathbf{Pr}[\mathcal{A}])$, where the expectation is over $\mathbf{r}$. We choose instead to provide thorough simulation results to illustrate how $\mathbf{Pr}[\mathcal{A}]$ compares to other previous algorithms. In our comparison, $\mathbf{Pr}[\mathcal{A}]$ directly translates to the success probability for the algorithm in question. We have simulated the following algorithms:

- The Soft Stern algorithm as proposed in the paper.
- Ordered Statistics Decoding (OSD). As explained in for example [15, 27, 30], we select the k most reliable and independent positions to form the most reliable basis (MRB). The error patterns in the list are chosen according to their reliability.
- Box-and-Match approach [28]. The essence of this algorithm is Stern-type, choosing the operated information set from the most reliable positions (i.e., an extension of MRB). However, they ignore the bit reliability when building the two colliding lists. For ease of comparison, we estimate the performance of its variant similar to the newly proposed algorithm but without choosing the error patterns in the colliding lists according to their reliability.
- A hard-decision Stern algorithm. This is a simple approach where we first make a hard decision in each position and then apply the original Stern algorithm. Each position of the received vector is $X_i \sim \mathbb{N}(1, \sigma)$ if zero is sent and hard decision gives a bit error if $X_i < 0$. The bit error probability is $p = \phi(1/\sigma)$. The probability of t errors is $\sum \binom{n}{t} p^t (1-p)^{n-t}$. The simulation results show that for the simulated parameter setting, this algorithm performs much worse than its three counterparts. We thereby removed it from our comparisons in the plots for readability.

For simplicity of analysis we compare single iteration versions of the algorithms. Techniques for taking advantage the soft information in multiple iterations is discussed briefly in Section 6.2, and can be applied to any of the algorithms. For a fair comparison, we assume that the complexity in one-round is approximately $C_{Gauss} + C \cdot (n-k)T$, where C is a small constant. Thus, we assume that for every algorithm, the size of one list is limited to $T = 2^l$ (to $2T$ for OSD, since only one list is built in this algorithm). The comparison of $\mathbb{E}(\mathbf{Pr}[\mathcal{A}])$ for various k, σ, T is shown in figures below. In all figures we let $n = 2k$. Thus, we have a code rate of $1/2$. In all figures we ignore the Q_l factor.[1] We look at two different scenarios, one with parameters applicable to a cryptographic scenario and one with parameters applicable to a coding theoretic scenario.

We have implemented the algorithm in Sage [24] [2]. The implementation covers the algorithm as described in Section 3. It was used to create the example from Section 4 and for simulating the success probability in this section. The source code for the implementation can be obtained upon request.

5.1.1. *Cryptographic Scenario.* In cryptographic applications of general soft decoding algorithms it is not uncommon to see a very small, but still non-negligible success probability $\mathbf{Pr}[\mathcal{A}]$. A large value of T is typically used. To compare the

[1] In a practical application of the algorithm one would have to swap in a few slightly less reliable positions if the first k positions are not linearly independent. Unless k is small this should not change the probabilities significantly.

[2] The implementation is available at https://github.com/ErikMaartensson/SoftStern

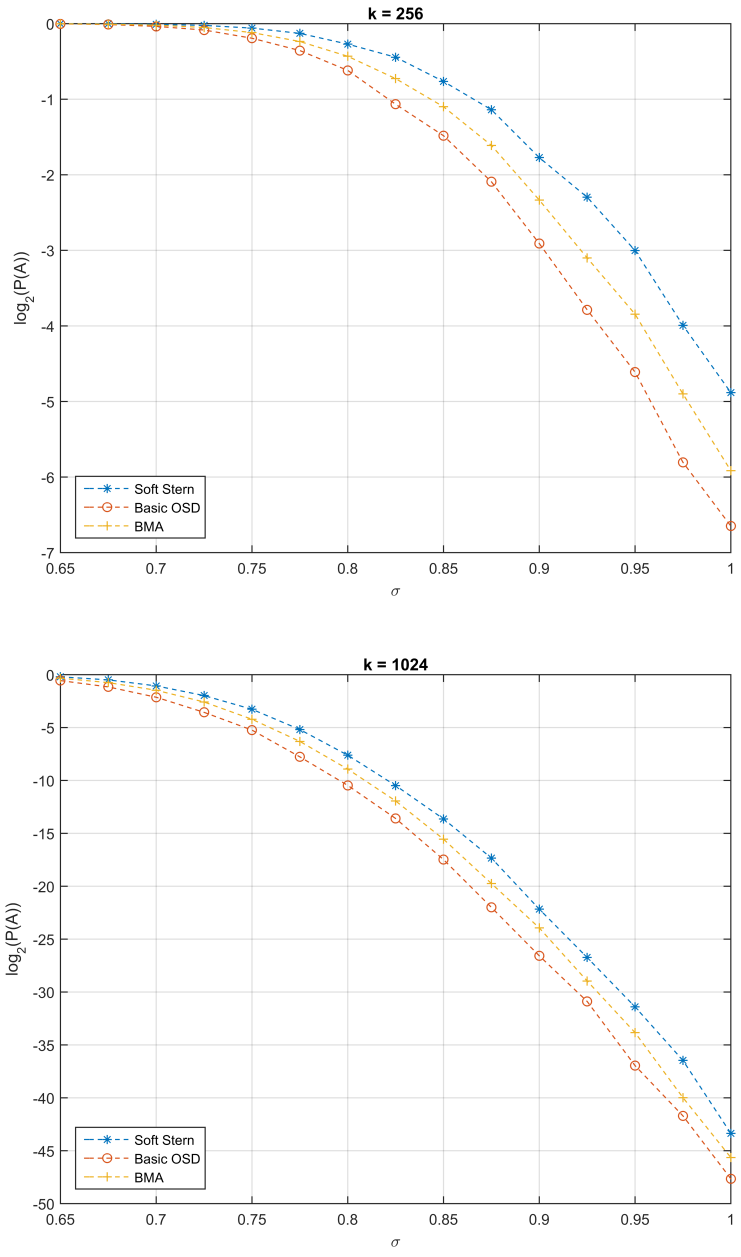

FIGURE 2. The logarithm of the success probability for the different algorithms as a function of σ.

performance of the algorithms in a crypto scenario we use large σ values. We let σ vary between 0.65 and 1 (in the latter case the capacity of the channel and the code rate are equal). We let $T = 2^l = 2^{20}$. In Figure 2, we plot the logarithm of the success probability as a function of σ in the cases where $k = 256$ and $k = 1024$. In

both cases our soft Stern algorithm performs much better than the other algorithms. Notice that the scale on the y-axis is not the same in the two plots.

5.1.2. *Coding Scenario.* In a coding scenario it is crucial that the word error probability $1 - \mathbf{Pr}[\mathcal{A}]$ is small. The acceptable value of T is smaller than in the cryptographic setting. To compare the algorithms we look at their probability of failing for small σ values. We vary σ between 0.4 and 0.65. We let $T = 2^l = 2^{10}$. In Figure 3, we plot the failure probability as a function of σ in the cases where $k = 128$ and $k = 512$. again, in both cases our soft Stern algorithm outperforms the other algorithms. Again, notice that the scale on the y-axis is not the same in the two plots.

6. Generalizations.

6.1. **Soft Output.** The algorithm can easily be modified to allow for soft output. The algorithm above outputs either the codeword $\hat{\mathbf{c}}$ closest to the received vector $\mathbf{r}$, or the first vector that is within some distance δ of $\mathbf{r}$. Instead, we can keep a number of the vectors $\mathbf{c}_i$ closest to $\mathbf{r}$. Based on the probabilities of each of the corresponding bit patterns we can then calculate the weighted average for each position. Now the algorithm can output soft information.

6.2. **Multiple Iterations.** If we are unsuccessful in our one-pass algorithm, we might want to allow for a new iteration, or many, to increase the success probability. We then suggest to swap one column from $\mathcal{S}$ with one column from $\mathcal{O}$. We want to take advantage of the reliability values, while still having a degree of randomness in the swapping. The technique we suggest is the approach experimentally tested as the optimal in [22]. Here the probability of swapping out $i \in \mathcal{S}$ is proportional to the probability that the corresponding position is wrongfully classified, that is, $(1-p_i)/\sum_{p_j \in \mathcal{S}}(1-p_j)$, where p_i is the conditional probability of having a correct bitwise hard decision, as being defined in (2). The probability of swapping in $j \in \mathcal{O}$ is proportional to the squared bias of j, that is, $\tau_j^2/\sum_{\tau_k \in \mathcal{O}} \tau_k^2$, where τ_j is the respective bias, i.e., $\tau_j = |p_j - \frac{1}{2}|$. The complexity can be analysed by employing a Markov-chain model, as was done in [7, 8].

7. Applications.

7.1. **Breaking Soft McEliece.** In [2], using soft information to enhance the performance of an attacking algorithm has been discussed, but no attacks below the security level have been presented. We show that soft McEliece can be broken by a trivial variant of Algorithm 2. The adversary will employ a simplistic version, i.e., keeping one element in each list. Therefore, she chooses l to be 0 and the considered error pattern is $\mathbf{0}$ in the k most reliable positions.

The attack can also be described as follows. The adversary chooses the k most reliable indices to form an information set $\mathcal{I}$, makes a bit-wise hard decision $\mathrm{sgn}(\cdot)$, and calculates the message $\mathbf{u}$ via a Gaussian elimination. She then tests whether this is a valid message. Otherwise, the adversary selects another ciphertext and tries again (if a single ciphertext can be broken the scheme is considered insecure). For one pass, the attack succeeds in case (i) that the sub-matrix corresponding to this information set is invertible and (ii) that there exist no errors among these positions. In implementation this latter probability is 0.98 if 80-bit security is targeted, and the expected complexity for recovering one message is about 3.5 Gaussian eliminations.

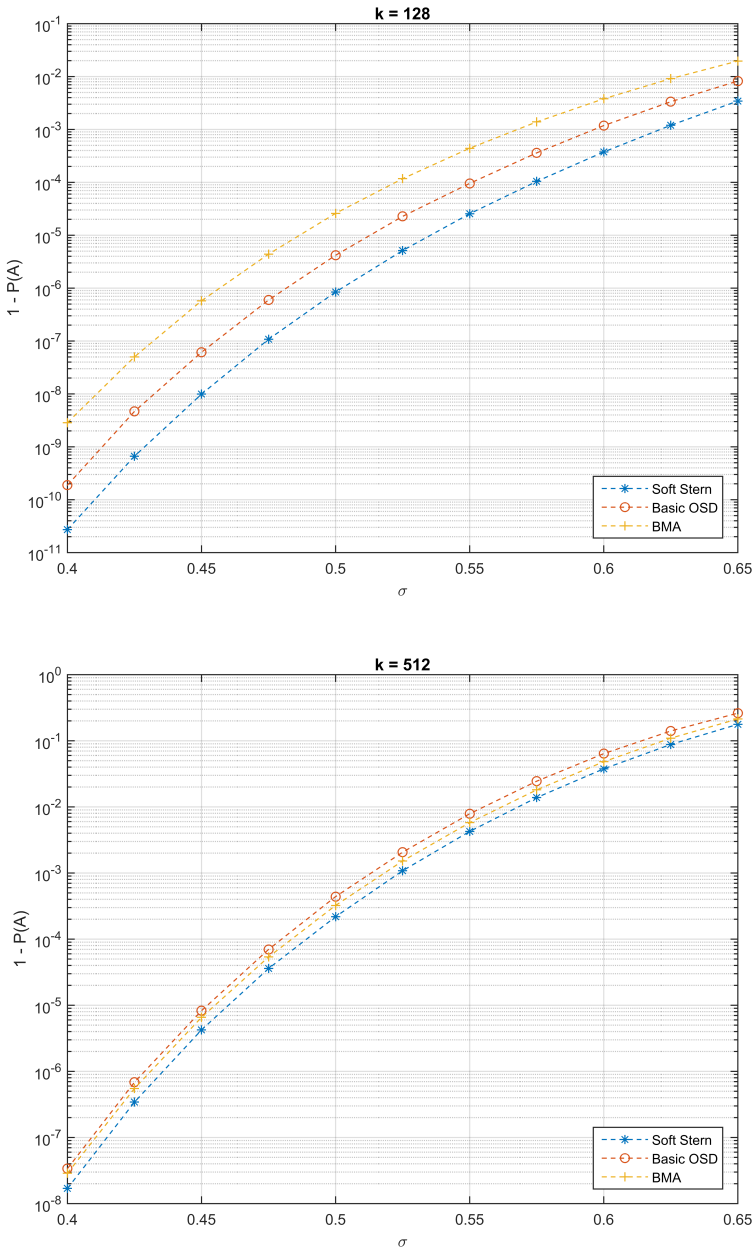

FIGURE 3. The failure probability for the different algorithms as a function of σ.

We give some intuition why this basic strategy can solve the decoding problem in soft McEliece for the proposed security parameters. In [2], one key security argument is that the total number of bit errors in one ciphertext follows a modified binomial distribution, which gives at least $\frac{n}{2}\text{erfc}\left(\frac{1}{\sqrt{2}\sigma}\right)$ bit errors. However, for the most reliable coordinates, the number of bit errors are very few. We see that

the expected number of bit errors among the $\frac{n}{2}$ most reliable bits is only 0.022 (or 0.015) using the parameters for 80 (or 128)-bit security. Most of the error positions are among the least reliable ones.

7.1.1. *Moving to a higher noise level.* Though this simplistic attack works well for soft McEliece, Algorithm 2 performs much better when the size of the targeted instance increases. Therefore, one should employ the full algorithm when aiming for cryptosystems with a reasonable security level.

A higher noise level increases the decryption (decoding) error probability. If $(n, \sigma) = (7202, 0.66)$, for instance, the 3601 most reliable bits are error-free with probability $2^{-13.0}$. Hence, on average about $29,000$ Gaussian eliminations are required using this simplistic attack. By using soft Stern, setting $l = 23$ and choosing a suitable δ in Algorithm 2, we can reduce the expected complexity to around 23 Gaussian eliminations[3].

7.2. **Applications in Side-channel Attacks.** Transforming some problems in side-channel analysis to that of decoding random linear codes originates in [5]. In this context, although the noise distribution is not exactly a Gaussian, soft information can still be exploited, making Algorithm 2 more efficient than other ISD variants. For side-channel attacks in [21, 22], the following modified version of the LPN problem occurs. Here, $\mathrm{Ber}(p)$ denotes a random variable that takes the value 1 with probability p and 0 with probability $1-p$, and $\langle \cdot, \cdot \rangle$ denotes the binary inner product of two vectors.

Definition 3 (Learning Parity with Variable Noise (LPVN) [22])**.** Let $\mathbf{s} \in \mathbb{F}_2^k$ and ψ be a probability distribution over $[0, 0.5]$. Given n uniformly sampled vectors $\mathbf{a}_i \in \mathbb{F}_2^k$, n error probabilities ϵ_i sampled from ψ, and noisy observations $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i = c_i + e_i$, where e_i is sampled from $\mathrm{Ber}(\epsilon_i)$, find $\mathbf{s}$.

They solve the problem by translating it into decoding a random linear code with soft information. They apply Stern's algorithm, but they do not sort the error patterns based on their probability of occurring. In this case the error is not Gaussian, but with some minor modifications our algorithm can still be applied. We sort the positions based on the ϵ_i values. The smaller ϵ_i is, the more reliable the position is. Next, we have

$$p_i = \mathbf{Pr}\left[b_i = c_i | \epsilon_i\right] = 1 - \epsilon_i.$$

After having done the transformations, such that the all-zero vector is the most probable vector in an index set $\mathcal{S}$, the probability of a bit pattern with ones in positions in $\mathcal{J} \subset \mathcal{S}$ and zeros in the other positions is

$$\prod_{i \in \mathcal{J}} \epsilon_i \cdot \prod_{i \notin \mathcal{J}, i \in \mathcal{S}} (1 - \epsilon_i) = \frac{\prod_{i \in \mathcal{J}} \epsilon_i}{\prod_{i \in \mathcal{J}} (1 - \epsilon_i)} \cdot \prod_{i \in \mathcal{S}} (1 - \epsilon_i) = \prod_{i \in \mathcal{J}} \frac{\epsilon_i}{(1 - \epsilon_i)} \cdot \prod_{i \in \mathcal{S}} p_i.$$

With some minor adjustments, the method for finding the most probable bit patterns, described in Section 3.2, can now be used.

[3] In the conference version [16], we presented an upper bound on the time complexity of solving this instance, i.e., 31 Gaussian eliminations. After careful simulation, we can now show a more accurate complexity estimation.

7.3. **Hybrid Decoding.** Another problem suited for our algorithm can be found in [1], where the problem of decoding linear codes with soft information appears. They analyze two codes proposed for space telecommanding. Both are LDPC codes, with $(n, k) = (128, 64)$ and $(n, k) = (512, 256)$ respectively. A hybrid approach for decoding is used. First one applies an efficient iterative decoder. In the few cases when the iterative decoder fails, one uses a decoder based on ordered statistics, thereby reducing the risk of decoding failure drastically.

However, the proposed ordered statistics algorithm does not make use of a Stern-type speed-up. It orders the positions after decreasing reliability. Then they try different error patterns in the k most reliable positions. Using our soft Stern algorithm, we instead divide the most reliable $k + l$ positions into two sets, and then look for collisions between the partial syndromes of the bit error patterns in the two sets. Adding such a Stern-type modification would greatly improve their ordered statistics decoder.

7.4. **Product Codes.** An application of the soft Stern algorithm with soft output is the decoding of product codes. Consider the serial concatenation of two codes, that do not have an efficient decoder with soft output. A small example would be the Golay code. An iterative decoder for this product code can be constructed by using the soft Stern algorithm with soft output together with a message-passing network (Tanner graph) between code symbols in the product code. Investigating this idea in more detail is an interesting research direction.

8. **Conclusions.** We have presented a new information set decoding algorithm using bit reliability, called the soft Stern algorithm. The algorithm outperforms what has been previously suggested for decoding general codes on the AWGN channel and similar tasks.

It can be utilized for a very efficient message-recovery attack on a recently proposed McEliece-type PKC named Soft McEliece [2], for an improved hybrid approach of decoding LDPC codes as in [1], and for side-channel attacks as in [21, 22]. We have also mentioned its use for decoding product codes.

Some modifications, such as multiple iterations of the algorithm, and producing soft output values, were discussed but not explicitly analyzed. Some ideas of future work may include further analyzing its use in iterative decoding and extending and deriving the exact algorithmic steps when considering multiple iterations.

Acknowledgments. The authors would like to thank the anonymous reviewers from ISIT 2017 and the reviewers for Advances in Mathematics of Communications for helping us improve the quality of this paper.

REFERENCES

[1] M. Baldi, N. Maturo, E. Paolini and F. Chiaraluce, On the use of ordered statistics decoders for low-density parity-check codes in space telecommand links, *EURASIP Journal on Wireless Communications and Networking*, **2016** (2016), 1–15.

[2] M. Baldi, P. Santini and F. Chiaraluce, Soft McEliece: MDPC code-based McEliece cryptosystems with very compact keys through real-valued intentional errors, in *IEEE International Symposium on Information Theory ISIT*, IEEE, (2016), 795–799.

[3] A. Becker, A. Joux, A. May and A. Meurer, Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding, in *Advances in Cryptology – EUROCRYPT* (eds. D. Pointcheval and T. Johansson), Springer, (2012), 520–536.

[4] S. Belaïd, J.-S. Coron, P.-A. Fouque, B. Gèrard, J.-G. Kammerer and E. Prouff, Improved Side-Channel Analysis of Finite-Field Multiplication., in *Cryptographic Hardware and Embedded Systems – CHES* (eds. T. Güneysu and H. Handschuh), Springer, (2015), 395–415.

[5] S. Belaïd, P.-A. Fouque and B. Gèrard, Side-channel analysis of multiplications in GF(2^{128}), in *Advances in Cryptology – ASIACRYPT* (eds. P. Sarkar and T. Iwata), Springer, (2014), 306–325.

[6] D. J. Bernstein, T. Lange and C. Peters, Smaller decoding exponents: Ball-collision decoding, in *Advances in Cryptology – CRYPTO* (eds. P. Rogaway), Springer, (2011), 743–760.

[7] D. J. Bernstein, T. Lange and C. Peters, Attacking and Defending the McEliece Cryptosystem, in *International Workshop on Post-Quantum Cryptography – PQCrypto* (eds. J. Buchmann and J. Ding), Springer, (2008), 31–46.

[8] A. Canteaut and F. Chabaud, A new algorithm for finding minimum-weight wordsin a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511, *IEEE Transactions on Information Theory*, **44** (1998), 367–378.

[9] D. Chase, A class of algorithms for decoding block codes with channel measurement information, *IEEE Transactions on Information theory*, **18** (1972), 170–182.

[10] J. Conway and N. Sloane, Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice, *IEEE Transactions on Information Theory*, **32** (1986), 41–50.

[11] I. Dumer, Sort-and-match algorithm for soft-decision decoding, *IEEE Transactions on Information Theory*, **45** (1999), 2333–2338.

[12] S. Dziembowski, S. Faust, G. Herold, A. Journault, D. Masny and F. Standaert, Towards sound fresh re-keying with hard (physical) learning problems, in *Advances in Cryptology – CRYPTO* (eds. M. Robshaw and J. Katz), Springer, (2016), 272–301.

[13] M. Finiasz and N. Sendrier, Security bounds for the design of code-based cryptosystems, in *Advances in Cryptology – ASIACRYPT*, (eds. M. Matsui), Springer, (2009), 88–105.

[14] M. P. Fossorier and S. Lin, Soft-decision decoding of linear block codes based on ordered statistics, *IEEE Transactions on Information Theory*, **41** (1995), 1379–1396.

[15] D. Gazelle and J. Snyders, Reliability-Based Code-Search Algorithms for Maximum-Likelihood Decoding of Block Codes, *IEEE Transactions on Information Theory*, **43** (1997), 239–249.

[16] Q. Guo, T. Johansson, E. Mårtensson and P. Stankovski, Information Set Decoding with Soft Information and some Cryptographic Applications, in *IEEE International Symposium on Information Theory – ISIT*, IEEE, (2017), 1793–1797.

[17] R. Koetter and A. Vardy, Algebraic soft-decision decoding of Reed-Solomon codes, *IEEE Transactions on Information Theory*, **49** (2003), 2809–2825.

[18] P. J. Lee and E. F. Brickell, An observation on the security of McEliece's public-key cryptosystem, in *Workshop on the Theory and Application of Cryptographic Techniques*, Springer, (1988), 275–280.

[19] A. May and I. Ozerov, On computing nearest neighbors with applications to decoding of binary linear codes, in *Advances in Cryptology - EUROCRYPT* (eds. E. Oswald and M. Fischlin), Springer, (2015), 203–228.

[20] R. Misoczki, J. P. Tillich, N. Sendrier and P. S. Barreto, MDPC-McEliece: New McEliece variants from moderate density parity-check codes, in *IEEE International Symposium on Information Theory – ISIT*, IEEE, (2013), 2069–2073.

[21] P. Pessl, L. Groot Bruinderink and Y. Yarom, To BLISS-B or not to be: Attacking strongSwan's Implementation of Post-Quantum Signatures, in *ACM SIGSAC Conference on Computer and Communications Security – CCS*, ACM, (2017), 1843–1855.

[22] P. Pessl and S. Mangard, Enhancing side-channel analysis of binary-field multiplication with bit reliability, in *RSA Conference Cryptographers' Track (CT-RSA)* (eds. K. Sako), (2016), 255–270.

[23] E. Prange, The use of information sets in decoding cyclic codes, *IRE Transactions on Information Theory*, **8** (1962), 5–9.

[24] The Sage Developers, SageMath, the Sage Mathematics Software System, http://www.sagemath.org, 2018.

[25] J. Stern, A method for finding codewords of small weight, in *Coding Theory and Applications* (eds. G. Cohen and J. Wolfmann), (1988), 106–113.

[26] A. Valembois, Fast soft-decision decoding of linear codes, stochastic resonance in algorithms, in *IEEE International Symposium on Information Theory – ISIT*, IEEE, (2000), 91.

[27] A. Valembois and M. Fossorier, Generation of binary vectors that optimize a given weight function with application to soft-decision decoding, in *IEEE Information Theory Workshop*, IEEE, (2001), 138–140.
[28] A. Valembois and M. Fossorier, Box and match techniques applied to soft-decision decoding, *IEEE Transactions on Information Theory*, **50** (2004), 796–810.
[29] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding,* McGraw-Hill, New York, 1979.
[30] Y. Wu and C. N. Hadjicostis, Soft-decision decoding using ordered recodings on the most reliable basis, *IEEE transactions on information theory*, **53** (2007), 829–836.

Received 1031 2018; revised xxxx 20xx.

E-mail address: qian.guo@eit.lth.se
E-mail address: thomas.johansson@eit.lth.se
E-mail address: erik.martensson@eit.lth.se
E-mail address: paul.stankovski@eit.lth.se