How to Lose Some Weight – A Practical Template Syndrome Decoding Attack

Sebastian Bitzer[1], Jeroen Delvaux[2], Elena Kirshanova[2], Sebastian Maaßen[3], Alexander May[3], and Antonia Wachter-Zeh[1]

[1] Technical University of Munich, Munich, Germany
[2] Technology Innovation Institute, Abu Dhabi, UAE
[3] Ruhr University Bochum, Bochum, Germany

Abstract. We study the hardness of the Syndrome Decoding problem, the base of most code-based cryptographic schemes, such as Classic McEliece, in the presence of side-channel information. We use ChipWhisperer equipment to perform a template attack on Classic McEliece running on an ARM Cortex-M4, and accurately classify the Hamming weights of consecutive 32-bit blocks of the secret error vector $\mathbf{e} \in \mathbb{F}_2^n$. With these weights at hand, we optimize Information Set Decoding algorithms. Technically, we show how to speed up information set decoding via a dimension reduction, additional parity-check equations, and an improved information set search, all derived from the Hamming weight information.

Consequently, using our template attack, we can practically recover an error vector $\mathbf{e} \in \mathbb{F}_2^n$ in dimension $n = 2197$ in a matter of seconds. Without side-channel information, such an instance has a complexity of around 88 bit. We also estimate how our template attack affects the security of the proposed McEliece parameter sets. Roughly speaking, even an error-prone leak of our Hamming weight information leads for $n = 3488$ to a security drop of 89 bits.

1 Introduction

Hardness of Syndrome Decoding. Central to all code-based schemes that advanced to the 4th Round of the NIST Post-Quantum Standardization Process [ARBC+20,ABB+23,MAB+23] lies the *Syndrome Decoding* (SD) *problem*: given a parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, where $\mathbb{F}_2$ denotes the binary finite field, a syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$, and an integer $w < n$, find the error vector $\mathbf{e}$ such that $H\mathbf{e} = \mathbf{s}$ and $|\mathbf{e}| < w$, where $|\cdot|$ denotes the Hamming weight.

An algorithm for solving this problem for a uniformly random H leads to a message or key recovery attack for the aforementioned schemes. Therefore, the syndrome decoding problem has received a significant amount of attention, resulting in various methods to solve it: Information Set Decoding (ISD) [Pra62,Ste89,MMT11], Statistical Decoding [Al 01,CDMHT22], and, recently, Sieving-style algorithms [GJN23,DEEK23]. Despite this extensive theoretical effort, the problem remains tractable for relatively small dimensions.

Concretely, in the setting of Classic McEliece (e.g., $w \approx \frac{n}{5\log_2 n}$ and $k \approx 0.8n$), the largest solved instance reported today [ALL19] is for $n = 1470$, and it already requires an optimized GPU implementation of an advanced information set decoding algorithm [NFK23], together with significant computational resources.[4]

Side-Channel Attacks. For the practical security of code-based schemes, it is important that the syndrome decoding problem also offers sufficient robustness against realistic side-channel attacks using leaks of the secret error vector $\mathbf{e} \in \mathbb{F}_2^n$. Compared to the comprehensive study of the syndrome decoding problem's classical security, its side-channel resistance has received much less attention.

Some initial theoretical work of Horlemann et al. [HPR+22] classifies different leakages and shows how to incorporate them into ISD algorithms to solve the syndrome decoding problem faster. One of the leakages considered in [HPR+22, Section 4] is *known Hamming weights of error blocks.*

In this leakage setting, one knows $\{|\mathbf{e}_i|\}_{i\leq t}$, where $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_t)$ and all $\mathbf{e}_i$'s (except, may be the last $\mathbf{e}_t$) are of the same length, i.e., the *word size* of the Central Processing Unit (CPU). For example, for an ARM Cortex-M4, each word $\mathbf{e}_i$ consists of 32 bits. Typical target instructions are *loads*, which move 32-bit words from SRAM to CPU registers, and *stores*, which move 32-bit words from CPU registers to SRAM. When executing such instructions, the power consumption is slightly different for each possible weight $|\mathbf{e}_i|$, and these unique characteristics can be condensed into a so-called *template* [CRR03]. We call the respective modified syndrome decoding problem, which additionally receives $\{|\mathbf{e}_i|\}_{i\leq t}$, the *template syndrome decoding* (template SD) problem.

While Horlemann et al. [HPR+22] describe a potential template syndrome decoding attack, their attack remains purely theoretical. Neither do the authors realize concrete power trace leaks, nor do they provide an improved information set decoding implementation. Thus, the practical implications of code-based template attacks remain unclear.

Contribution. In this work, we perform for the first time an explicit template attack on a Classic McEliece implementation. To this end, we realize a concrete power trace leak, from which we derive with high accuracy (but still error-prone) the desired Hamming weight information $\{|\mathbf{e}_i|\}_{i\leq t}$.

We then improve information set decoding by using and enhancing the techniques of Horlemann et al. [HPR+22]. Building on information set decoding software from Esser, May, and Zweydinger [EMZ22], we provide a concrete implementation of these improvements.

With our (erroneous but easily correctable) leakage, we run our template information set decoding and retrieve the secret $\mathbf{e} \in \mathbb{F}_2^n$. Concretely, we are able to solve the template syndrome decoding problem for Classic McEliece in dimension $n = 2197$ in a matter of seconds. Without template, such an instance has complexity around 88 bits. In more detail, our results are as follows.

[4] See also https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/WzgqEmAfnk8 for the discussion on hardness predictions for this instance.

1. We use ChipWhisperer equipment to measure the power consumption of an open-source implementation [CC21] of Classic McEliece running on an ARM Cortex-M4, or at least a decapsulation subroutine that checks whether $|\mathbf{e}| = w$. Using $48k$ traces for template building, and $12k$ for matching, the weights $\{|\mathbf{e}_i|\}$ we recover are correct with a probability of around 97%. We show how to deal with this measurement noise in the full version.
2. We modify the ISD algorithms of Prange [Pra62] and Dumer [Dum91] by incorporating the template. Specifically, we show how to encode the knowledge of weights of error blocks into the parity-check matrix H. Then, using such modified H, we show how to decrease the expected running time of the above ISD algorithms, again exploiting the leakage.
3. We provide efficient and parallelized implementations of the modified ISD algorithms. With our software we are able to solve the $n = 2197$ instance from [ALL19] in a matter of 10 seconds on AMD EPYC 7742 using 200 threads. Based on our implementation, we estimate the hardness of larger McEliece instances under this template attack.

Related work. Closely related to the template syndrome decoding is *regular syndrome decoding* introduced in [AFS05]. In regular SD, for each block $\mathbf{e}_i$ of $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_t)$ it holds that $|\mathbf{e}_i| = 1$. Note that regular SD is a special case of Template SD. Recent work of Esser and Santini [ES23] studies the hardness of regular SD, and some of their ideas apply to our setting, e.g., the construction of new parity-check equations, see also [EMZ22].

Another template attack on Classic McEliece was presented by Grosso et al. in [GCCD23]. The authors of [GCCD23] aim at the same leakage, namely, $\{|\mathbf{e}_i|\}_{i \le t}$ but they retrieve it from the matrix-vector multiplication $H \cdot \mathbf{e}$ that computes the syndrome. In our template attack, similar to [GCCD23], we discard the columns of H that correspond to the zero-weight blocks in the template. Contrary to [GCCD23], in our work we show how to make use of the *non-zero* weight blocks to speed-up ISD algorithms, and we implement our ISD algorithms in order to actually retrieve the secret.

Another side-channel attack exploiting failures of the decoding procedure in McEliece decryption is studied in [LNPS20]. The authors show how to learn the positions of 1's in the secret vector by querying the decoder with modified syndromes. Similar to our work, the authors combine the obtained information with ISD algorithms and estimate their attack performance. In contrast, we implement our (modified) ISD routines, report on concrete runtimes for feasible instance and then give estimates for large dimensions.

In summary, in contrast to [GCCD23] and [LNPS20] we do not only *estimate* the effects on ISD, but we retrieve Hamming weight side-channel information, correct errors, provide improved ISDs via dimension reduction and additional parity check equations, and *practically solve* an $n = 2197$-dimensional template SD instance in a matter of seconds.

Artifacts. Our software for Template ISD algorithms as well as scripts to generate the figures are available in [Tem24].

2 Template ISD

Notations. Let $|\mathbf{x}|$ denote the Hamming weight of $\mathbf{x}$ and by $[i, j)$ the interval of consecutive integers $\{i, i+1, \ldots, j-1\}$. By S_n we denote the group of all permutations on sets of size n. By $\mathbb{I}_n$ we denote the identity matrix of rank n.

Problem definitions. In the Classic McEliece KEM [ARBC+20], the decryption process receives as input a syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$ and recovers the secret message $\mathbf{e}$ by calling an efficient syndrome decoder using the McEliece secret key. Once $\mathbf{e}$ is retrieved, the decryption checks if $|\mathbf{e}| = w$, where w is the decoding capacity of the syndrome decoder. The parameter w is a fixed public parameter. Classic McEliece decryption only returns $\mathbf{e}$, if $\mathbf{e}$ passes the check $|\mathbf{e}| = w$.

Without knowledge of the secret key, message recovery attacks on Classic McEliece require solving the Syndrome Decoding (SD) problem.

Definition 1 (Syndrome Decoding (SD)). *Let $H \in \mathbb{F}_2^{(n-k)\times n}$ be a random-looking parity-check matrix, $\mathbf{e}$ an error vector of Hamming weight w, and $\mathbf{s} = H\mathbf{e} \in \mathbb{F}_2^{n-k}$ the corresponding syndrome. SD asks to find the unique weight-w $\mathbf{e} \in \mathbb{F}_2^n$ satisfying $H\mathbf{e} = \mathbf{s}$.*

The side-channel attack we consider in this work creates a template for the function that computes $|\mathbf{e}|$. In the ideal scenario, such a template allows the attacker to learn the blockwise weight of $\mathbf{e}$. We call the SD problem that in addition receives the blockwise weight *template Syndrome Decoding.*

Definition 2 (Template SD). *Let $H \in \mathbb{F}_2^{(n-k)\times n}$ be a parity-check matrix of a random code and $\mathbf{s} = H\mathbf{e} \in \mathbb{F}_2^{n-k}$, for some $\mathbf{e}$ of Hamming weight w. Let further $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_t)$ with $\mathbf{e}_i \in \mathbb{F}_2^b$ for $i = [1, t)$, $\mathbf{e}_t \in \mathbb{F}_2^{n-b\cdot(t-1)}$, and $w_i = |\mathbf{e}_i|$. Template ISD asks to find $\mathbf{e}$ given $H, \mathbf{s}$, and $\{w_i\}_{i \le t}$.*

Definition 3 (Guess). *We call any vector $\{\hat{w}_i\}_{i\le t} \in \mathbb{N}_0^t$ a* guess*. The* accuracy *of a guess is the percentage of correctly identified weights, i.e. $\frac{|\{i \in [1,t] | \hat{w}_i = w_i\}|}{t}$. A guess is* error-free *if it has accuracy 1, otherwise it is* error-prone*. Notice that in general error-prone guesses do not satisfy $\sum_{i=1}^{t} \hat{w}_i = w$.*

The block size b, and, therefore, also the template's length depends on the target architecture's specifications. Our template attack targets ARM Cortex-M4 processor that operates on words of 32 bits. Hence, our guesses will be of length $t = \lceil n/32 \rceil$.

Running Example $n = 2197$. Our running example uses the parameters $n = 2197$, $k = 1758$, and $w = 37$. Therefore, a guess is a string of length $t = 69$, its i-th entry indicating the weight of the i-th 32-bit block of $\mathbf{e}$.

3 Algorithms for Template ISD

3.1 Permutation-based Template ISD – Improving Prange

Let us start with the fundamental information set decoding algorithm due to Prange [Pra62]. Prange's algorithm permutes the columns of H, which is equivalent to permuting the positions of 1's in $\mathbf{e}$.

Let $\pi \in S_n$ be a permutation and let $\pi(H) = (Q \mid \cdot)$ be the result of applying the permutation π to H such that $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$ is invertible (this event occurs with constant probability). Multiplying by Q^{-1} from the left both $\pi(H)$ and $\mathbf{s}$ leads to an equivalent SD instance written in systematic form:

$$H'\pi(\mathbf{e}) = \mathbf{s}', \quad \text{where} \quad H' = Q^{-1}H = (\mathbb{I}_{n-k} \mid \cdot), \text{ and } \mathbf{s}' = Q^{-1}\mathbf{s}.$$

If $\pi(\mathbf{e})$ has weight 0 on the last k coordinates, then $|\mathbf{s}'| = w$. This means that the first $(n-k)$ coordinates of $\pi(\mathbf{e})$ are given by $\mathbf{s}'$ and $\mathbf{e}$ can be reconstructed.

Dimension reduction. As already noticed in the work of Grosso et al. [GCCD23], any weight-0 block with $w_i = 0$ does not contribute to the solution $\mathbf{e}$. Let m_0 denote the number of error-free blocks. Then $b \cdot m_0$ columns of H do not contribute and can be eliminated, leading to a modified parity-check matrix $\bar{H} \in F_2^{(n-k)\times\bar{n}}$ with $\bar{n} = n - b \cdot m_0$ columns. This in turn reduces the dimension of the solution $\mathbf{e}$ from n to $\bar{n} = n - b \cdot m_0$ leaving its weight w unchanged.

Improved permutation. The idea of the permutation π in Prange's algorithm is to move all w 1-entries of $\mathbf{e}$ upfront to the first $n-k$ coordinates. Having weight w_i for the i-th block, we permute a number proportional to w_i upfront. Concretely, in Algorithm 2, we use the following *template-optimized permutation.*

Let P be a permutation matrix and $v_i \in \mathbb{Z}$ with $0 \le v_i \le b$ and $\sum_{i=1}^{t} v_i = n-k$. Further, denote the permuted error vector as

$$P\mathbf{e} = (\mathbf{e}', \mathbf{e}'') = (\mathbf{e}'_1, \ldots, \mathbf{e}'_t, \mathbf{e}''_1, \ldots, \mathbf{e}''_t)$$

with $\mathbf{e}'_i \in \mathbb{F}_2^{v_i}$ and $\mathbf{e}''_i \in \mathbb{F}_2^{b-v_i}$. Then, P is a template permutation if $\mathbf{e}'_i$ and $\mathbf{e}''_i$ originate from $\mathbf{e}_i$ for all i. The success probability $P(\sum_i |\mathbf{e}''_i| = 0)$ is determined by the v_i. In [HPR+22], a greedy algorithm for optimizing v_i is given. We observe that this optimal choice corresponds to the setting $v_i \approx \frac{w_i}{w} \cdot (n-k)$, i.e., the number of columns is chosen *proportional* to the weight of the block. This proportional assignment of columns generalizes the puncturing of [GCCD23]: columns of H with $w_i = 0$ are implicitly ignored by taking 0 columns from the blocks with $w_i = 0$. In Algorithm 2, the procedure that samples a template permutation as described above is called `TemplatePerm`.

In practice, $v_i = \frac{w_i}{w} \cdot (n-k)$ cannot be used directly due to rounding issues and the restriction $v_i \le b$. In our implementation, we minimize $|v_i - \frac{w_i}{w} \cdot (n-k)|$.

Additional Parity Check Equations. Note that $|\mathbf{e}_i| = w_i$ implies that the sum of the entries of $\mathbf{e}_i$ is $w_i \bmod 2$, see also [EMZ22]. Hence, for $w_i > 0$, one can extend the parity-check matrix by appending a row of the shape $(0, \ldots, 0, 1, \ldots, 1, 0, \ldots 0)$, where the all-1 block is at the positions $[i \cdot b, (i+1) \cdot b)$. The syndrome $\mathbf{s}$ is extended by appending $w_i \bmod 2$. Each appended check increases the co-dimension of the code by one to eventually $n-k+t-m_0$. This makes it simpler for ISD to find a permutation that puts all weight to the first co-dimension many positions.

Algorithm 1: Prange

```
Input : $H$, $\mathbf{s}$, $w$
Output: $\mathbf{e}$
1 repeat
2     Sample $P \in S_n$
3     Let $H' = Q^{-1}HP$ be the systematic form of $HP$
4     $\mathbf{s}' = Q^{-1}\mathbf{s}$
5 until $|\mathbf{s}'| = w$
6 return $P^{-1} \cdot (\mathbf{s}', 0^k)$.
```

Algorithm 2: Template Prange

```
Input : $H \in \mathbb{F}_2^{(n-k)\times n}$, $\mathbf{s}$, $\{w_i\}_{i\le t}$; $\sum_i w_i = w$
Output: $\mathbf{e}$
1 Let $m_0 := |\{i \le t \mid w_i = 0\}|$, $\bar{n} = n - m_0 b$, $\bar{k} = k + m_0 - t$.
2 Obtain $\bar{H} \in \mathbb{F}_2^{(n-k)\times \bar{n}}$ by removing zero blocks.
3 Obtain $\bar{H} \in \mathbb{F}_2^{(n-\bar{k})\times \bar{n}}$, $\bar{\mathbf{s}} \in \mathbb{F}_2^{n-\bar{k}}$ by adding checks.
4 repeat
5     $P \leftarrow$ TemplatePerm($w$)
6     Let $H' = Q^{-1}\bar{H}P$ be the systematic form of $\bar{H}P$
7     $\mathbf{e}' = Q^{-1}\bar{\mathbf{s}}$.
8 until $|\mathbf{e}'| = w$
9 return $P^{-1} \cdot (\mathbf{e}', 0^k)$.
```

We summarize all modifications to the parity-check matrix and the optimized permutations in Figure 1.

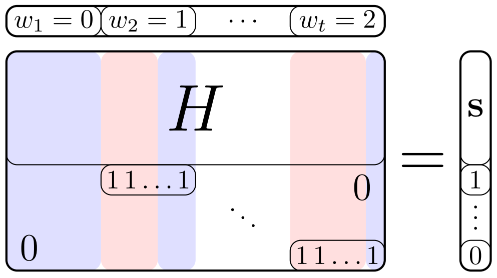

Fig. 1: Illustration of our improved Template ISD method. Columns in blocks with error weight $w_i = 0$ are punctured. For $w_i \neq 0$, an additional check is appended to the parity-check matrix and the syndrome. For each block, the number of columns chosen for permutation upfront (colored red) is set proportionally to the error weight.

Theorem 1. *Let $\{w_i\}_{i\le t}$ be a an error-free guess with m_0 many zeros. The expected number of permutations of Algorithm 2 for solving Template SD is*

$$\prod_{i=1}^{t} \binom{b}{w_i} \binom{\lfloor \frac{w_i}{w}(n-k+t-m_0) \rceil}{w_i}^{-1}.$$

Proof. Our Algorithm 2 finds a good permutation if for all t blocks of length b, all w_i-many 1's from the i-th block will be moved upfront to the first $n-\bar{k} = n-k+t-m_0$ coordinates. As from each block we take $\lfloor \frac{w_i}{w}(n-k+t-m_0) \rceil$ many positions, the expected number of required permutation follows. □

Running example $n = 2197$. According to [EVZB23], the concrete complexity of Algorithm 1 for $n = 2197$, $k = 1758$, $w = 37$ is estimated as 116 bits. Dimension reduction by weight-0 blocks reduces the complexity of this instance to 71 bits. With improved permutation and additional parity check equations from Algorithm 2, the complexity further decreases to 62 bits. Figures for larger McEliece instances are available in [Tem24].

3.2 Enumeration-based Template ISD – Improving Dumer

Recall from Section 3.1 that for a parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ Prange's algorithm finds a permutation that shifts all w 1-entries of $\mathbf{e}$ upfront to the first $n-k$ entries. That is why Prange is called a permutation-based ISD.

Instead, enumeration based ISD algorithms like [Dum91,FS09,MMT11] choose small parameters p, ℓ and permute $\mathbf{e}$ such that weight $w-2p$ lands on the first $n-k-\ell$ coordinates, and the remaining weight $2p$ lands on the last $k+\ell$ coordinates. On the one hand, such a permutation is way more likely to find the secret. On the other hand, we now have to enumerate a search space of size $\binom{k+\ell}{2p}$, in Dumer's algorithm in a meet-in-the-middle fashion. For usual McEliece such a tradeoff pays off, i.e., the benefit of faster finding a suitable permutation outweighs the drawback of enumeration.

In this work, we chose to adapt Dumer's algorithm to the Template ISD setting. In the parameter range that we practically solve, Dumer's algorithm is known to perform best, whereas more advanced algorithm like [MMT11] are taking over for large n of cryptographic size [EMZ22].

Although we now choose with Dumer an enumeration-based ISD algorithm, the benefits from the Template ISD still contribute to a large extent to the search for a suitable permutation. Namely, analogous to Section 3.1, we obtain the template version of Dumer using the following modifications and improvements:

Dimension reduction: 0-weight blocks from the guess $\{w_i\}_{i\le t}$ are removed, let m_0 be their number. Such a dimension reduction helps to significantly speed up permutation search, and it also decreases the search space for enumeration by $m_0 b$.

Additional parity checks: Encoding all $w_i \ge 1$ into additional check equations increases the co-dimension from $n-k$ to $n-k+t-m_0$. This speeds up permutation search further, and slightly reduces the enumeration cost.

Improved permutation: Similar to Section 3.1, we permute upfront proportionally to the weights w_i to improve the permutation. For this, we set $v_i \approx \frac{w_i \cdot c}{w-2p}(n-\bar{k}-\ell)$, where $c = \frac{w-2p}{w}$ is a re-scaling factor. We do not exploit non-zero weights for enumeration.

The resulting algorithm Template Dumer is given in Algorithm 3.

Theorem 2 (Template Dumer). *Let $k' := \bar{n} - (n-\bar{k}) + \ell$ with $\bar{n}$ and $\bar{k}$ as in Algorithm 3. Then, the number of iterations that Template Dumer ISD performs on average is the inverse of the success probability*

$$\binom{k'/2}{p}^2 \binom{k'}{2p}^{-1} \sum_{p_1+\ldots+p_t=2p} \prod_{i=1}^{t} \binom{\lfloor \frac{w_i}{w}(n-\bar{k}-\ell)\rceil}{w_i - p_i} \binom{b}{w_i}^{-1}, \tag{1}$$

where each iteration has a meet-in-the-middle cost of $2\binom{k'/2}{p} + \binom{k'/2}{p}^2 \cdot 2^{-\ell}$.

Proof. Since $\lfloor \frac{w_i}{w}(n-\bar{k}-\ell)\rceil$ positions of the i-th block are moved upfront, it contributes p_i errors to the last k' positions with probability $\binom{\lfloor \frac{w_i}{w}(n-\bar{k}-\ell)\rceil}{w_i-p_i}\binom{b}{w_i}^{-1}$.

Algorithm 3: Template Dumer

```
Input  : $H \in \mathbb{F}_2^{(n-k)\times n}$, $\mathbf{s}$, $\{w_i\}_{i\le t}$; $\sum_i w_i = w$, $b$, $p$, $\ell$
Output: $\mathbf{e}$
1  Let $m_0 := |\{i \le t \mid w_i = 0\}|$, $\bar{n} = n - m_0 b$, $\bar{k} = k + m_0 - t$, $k' = \bar{n} - (n - \bar{k}) + \ell$.
2  Obtain $\bar{H} \in \mathbb{F}_2^{(n-k)\times \bar{n}}$ by removing zero blocks.
3  Obtain $\bar{H} \in \mathbb{F}_2^{(n-\bar{k})\times \bar{n}}$, $\bar{\mathbf{s}} \in \mathbb{F}_2^{n-\bar{k}}$ by adding checks.
4  repeat
5      $P \leftarrow$ TemplatePerm$(w - 2p)$
6      Let $H' = Q^{-1}\bar{H}P$ be the quasi-systematic form of $\bar{H}P$,
       $(\mathbf{s}', \mathbf{s}'') = Q^{-1}\bar{\mathbf{s}} \in \mathbb{F}_2^{n-\bar{k}-\ell} \times \mathbb{F}_2^{\ell}$.
7      for all collisions $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^{k'/2}$ with weight $p$ do
8          Compute $\mathbf{e}' := H_1\mathbf{e}_1 + H_2\mathbf{e}_2 + \mathbf{s}'$.        ▷ via Meet-in-the-Middle
9  until $|\mathbf{e}'| = w - 2p$
10 return $P^{-1} \cdot (\mathbf{e}', \mathbf{e}_1, \mathbf{e}_2)$.
```

Similar to [HPR+22], the probability of $2p$ errors in the last k' positions is obtained by summing over all possibilities $p_1 + \ldots + p_t = 2p$. Further, the error needs to split evenly in the last k' positions. Randomizing the order of these coordinates, this probability is $\binom{k'/2}{p}^2 \binom{k'}{2p}^{-1}$. The meet-in-the-middle step requires enumerating $2\binom{k'/2}{p}$ vectors $\mathbf{e}_1$, $\mathbf{e}_2$, leading to $\binom{k'/2}{p}^2 2^{-\ell}$ collisions on average.

Running example $n = 2197$. For $n = 2197$, $k = 1758$, $w = 37$, we pick $\ell = 16$ and $p = 2$. The performance differs between guesses. On average, $2^{19.9}$ iterations are sufficient, each with a Meet-in-the-Middle cost of processing $2^{14.7}$ vectors.

3.3 Dealing with Noisy Guesses

The full version provides an algorithm that deals with noisy guesses. In particular, we show that Template Prange and Template Dumer are robust to single (or very few) misclassifications.

4 Side-Channel Experiments

4.1 Measurement Setup

We target an open-source C implementation of McEliece, which is made by Chen and Chou [CC21], optimized for the ARM Cortex-M4, and unprotected against side-channel attacks. The targeted function is a Hamming-weight computation in the decryption, as specified in Listing 1.1 [CC22]. To accelerate our measurements, we do not run the entire decapsulation, and instead communicate via UART with a custom wrapper around function weight_3488. Likewise, although solving $n = 3488$ is computationally feasible, we reduce n to 2197 for faster results. The code is compiled by arm-none-eabi-gcc using O3 optimization. Although the right-shift of the 32-bit word $\mathbf{v}$ in Listing 1.1 might leak bit-level information, we only aim to recover word-level information, i.e., weights $|\mathbf{v}|$.

```c
static int weight_3488(uint32_t *v)
{
  int i, w = 0;
  for (i = 0; i < 3488; i++)
    w += (v[i>>5] >> (i&31)) & 1;
  return w;
}
```

Listing 1.1: Targeted C function [CC22].

The power consumption is measured using ChipWhisperer equipment: a CW308 UFO board, an STM32F405RGT6 target that contains an ARM Cortex-M4, and a Husky oscilloscope. The clock frequency is set to 16 MHz and the sampling frequency is set to 128 MHz, i.e., there are 8 samples per clock period. To synchronize traces, the wrapper raises a trigger signal right before function weight_3488 is executed. To capture the entire operation, 201559 samples suffice. As the Husky has a buffer of 131070 samples, we stitch together 2 traces by varying the offset from the trigger. Traces for template building and template matching are taken from the same STM chip, which is fair: to build templates, the attacker can perform unlimited encapsulations to obtain known pairs $(\mathbf{c}, \mathbf{e})$.

4.2 Template Building

Given that error $\mathbf{e}$ spans 69 words, each having weight $W \in \{0, 1, 2, 3\}$ with overwhelming probability, $276 = 69 \times 4$ templates are built. For this purpose, we randomly generate $48k$ error vectors $\mathbf{e}$ and measure one trace for each $\mathbf{e}$. To ensure that the templates have similar qualities, we impose $P(W = 0) = P(W = 1) = P(W = 2) = P(W = 3) = 1/4$. This deviation from the McEliece distribution is optional and is only realistic for a 2-device attack. All choose-W-out-of-32 selections are equally likely. For example, words 0x80020040 and 0x01400002 are equally likely in the case of $W = 3$. For each out of 69 words, we only retain the 100 samples that matter most. All other samples primarily generate classification noise, so it's beneficial to discard them. To make a selection, we use an extension of Welch's t-test specified below, where M_W is the *sample mean*, V_W is the *sample variance*, and N_W is the number of traces for each weight W.

$$T = \frac{1}{3}\left(\frac{M_0 - M_1}{\sqrt{\frac{V_0}{N_0} + \frac{V_1}{N_1}}} + \frac{M_1 - M_2}{\sqrt{\frac{V_1}{N_1} + \frac{V_2}{N_2}}} + \frac{M_2 - M_3}{\sqrt{\frac{V_2}{N_2} + \frac{V_3}{N_3}}}\right)$$

4.3 Template Matching

For each error $\mathbf{e}$ we aim to recover, we collect $12k$ traces, and average them into a single trace X. Now, the weights W are non-uniform and follow the McEliece distribution. For each out of 69 words, the distinguisher $D_W = \sum_{i=0}^{99} |T_i| \cdot |M_W - X_i| \in \mathbb{R}^+$ is computed. The weight W for which D_W is the smallest is the best guess. The probability that this guess is correct is around 97%.

5 Practical Template SD Solving with Our Algorithms

We implemented Algorithm 2 and Algorithm 3. The source-code of our implementation can be found in [Tem24]. We ran our experiments on the parity-check matrices of Classic McEliece instances with parameters provided by [ALL19], where we generated the solution vectors $\mathbf{e}$ ourselves. We fully recovered the secret error vector for all instances $n \leq 2197$.

In the experiments, we always worked with an error-free guess. Indeed, for our running example $n = 2197$, the actual side-channel attacks gave guesses with 97% accuracy resulting in a single mispredicted block: we observed a guess $\hat{\mathbf{w}}$ with $\sum_i \hat{w}_i = w - 1$, which can be corrected with low overhead.

To accurately estimate the running time of Algorithms 1 (Original Prange), 2 (Template Prange), and 3 (Template Dumer) for all dimensions n, we generated random error vectors and measured the runtime *per permutation*. To obtain the overall runtime, we multiply by the expected number of permutations, which is computed as in Theorems 1 and 2 by averaging over different error vectors. The resulting estimates are presented in Figure 2.

Running Example $n = 2197$. Our implementation of Algorithm 3 (with $p = 2$, $\ell = 16$) on 2x AMD EPYC 7742 CPUs recovers the secret $\mathbf{e}$ for $n = 2197$ in 1019 seconds with 1134185 iterations required (the predicted number of iterations for this instance is $1.6 \cdot 10^6$). The implementation is parallelized over the choice of permutation, and with 200 threads outputs the secret in 10 seconds using only 334 MB of RAM.

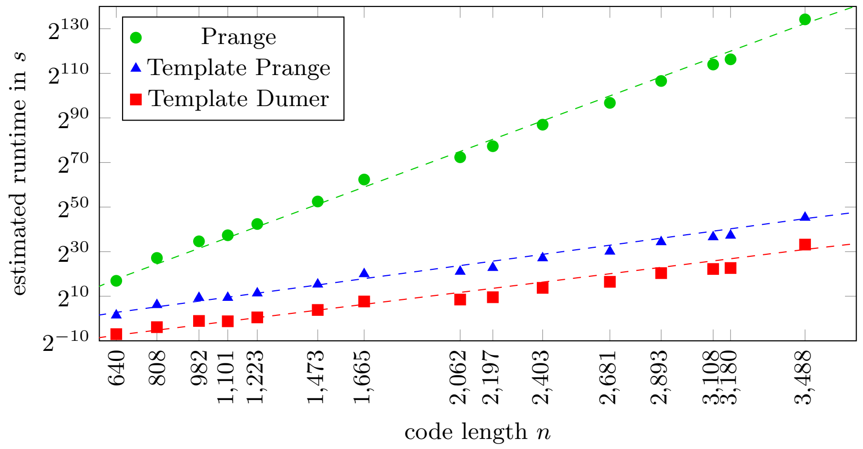

Fig. 2: Single-threaded performance of our implementation on AMD EPYC 7742

References

ABB$^+$23. Nicolas Aragon, Pailo L. Barreto, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolar Sendrier, Jean-Pierre Tillich, Vasseur Valentin, and Gilles Zémor. BIKE - Bit Flipping Key Encapsulation. https://bikesuite.org/, 2023.

AFS05. Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In *Progress in Cryptology – Mycrypt 2005*, pages 64–83, 2005.

Al 01. A. Kh. Al Jabri. A statistical decoding algorithm for general linear block codes. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *LNCS*, pages 1–8. Springer, Heidelberg, December 2001.

ALL19. Nicolas Aragon, Julien Lavauzelle, and Matthieu Lequesne. decodingchallenge.org, 2019.

ARBC$^+$20. Martin Albrecht R., Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Persichetti Edoardo, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece: conservative code-based cryptography. https://classic.mceliece.org/nist/mceliece-20201010.pdf, 2020.

CC21. Ming-Shing Chen and Tung Chou. Classic McEliece on the ARM cortex-M4. *IACR TCHES*, 2021(3):125–148, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8970.

CC22. Ming-Shing Chen and Tung Chou. Classic McEliece implementation for ARM-Cortex M4. https://github.com/pqcryptotw/mceliece-arm-m4/blob/main/pqm4-projects/crypto_kem/mceliece348864/ches2021/decrypt_n3488_t64.c, commit f2a699dd480f9f91d566eb4b910fd4e51e3bdc91, January 2022.

CDMHT22. Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0: Reducing decoding to LPN. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 477–507. Springer, Heidelberg, December 2022.

CRR03. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003.

DEEK23. Léo Ducas, Andre Esser, Simona Etinski, and Elena Kirshanova. Asymptotics and improvements of sieving for codes. *Cryptology ePrint Archive*, 2023.

Dum91. Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52. Moscow, 1991.

EMZ22. Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022,* *Part III*, volume 13277 of *LNCS*, pages 433–457. Springer, Heidelberg, May / June 2022.

ES23. Andre Esser and Paolo Santini. Not just regular decoding: Asymptotics and improvements of regular syndrome decoding attacks. *Cryptology ePrint Archive*, 2023.

EVZB23. Andre Esser, Javier A. Verbel, Floyd Zweydinger, and Emanuele Bellini. {CryptographicEstimators}: a software library for cryptographic hardness estimation. *{IACR} Cryptol. ePrint Arch.*, page 589, 2023.

FS09. Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15*, pages 88–105. Springer, 2009.

GCCD23. Vincent Grosso, Pierre-Louis Cayrel, Brice Colombier, and Vlad-Florin Drăgoi. Punctured syndrome decoding problem: Efficient side-channel attacks against classic McEliece. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 170–192. Springer, 2023.

GJN23. Qian Guo, Thomas Johansson, and Vu Nguyen. A new sieving-style information-set decoding algorithm. *Cryptology ePrint Archive*, 2023.

HPR$^+$22. Anna-Lena Horlemann, Sven Puchinger, Julian Renner, Thomas Schamberger, and Antonia Wachter-Zeh. Information-set decoding with hints. In *Code-Based Cryptography*, pages 60–83, 2022.

LNPS20. Norman Lahr, Ruben Niederhagen, Richard Petri, and Simona Samardjiska. Side channel information set decoding using iterative chunking - plaintext recovery from the "classic McEliece" hardware reference implementation. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 881–910. Springer, Heidelberg, December 2020.

MAB$^+$23. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jurjen Bos, Jean-Christophe Deneuville, Arnaud Dion, Philippe Gaborit, Jérôme Lacan, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. Hamming quasi-cyclic (HQC). https://pqc-hqc.org/, 2023.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, Heidelberg, December 2011.

NFK23. Shintaro Narisada, Kazuhide Fukushima, and Shinsaku Kiyomoto. Multiparallel MMT: faster ISD algorithm solving high-dimensional syndrome decoding problem. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 106(3):241–252, 2023.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962.

Ste89. Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, pages 106–113, 1989.

Tem24. A practical template syndrome decoding attack. https://github.com/ChuTriel/TemplateISD, 2024.