Assessing the Feasibility of Single Trace Power Analysis of Frodo

Joppe W. Bos[1], Simon Friedberger[1,2], Marco Martinoli[3], Elisabeth Oswald[3], and Martijn Stam[3]

[1] NXP Semiconductors joppe.bos@nxp.com
[2] KU Leuven - iMinds - COSIC simon.friedberger@esat.kuleuven.com
[3] University of Bristol, United Kingdom
marco.martinoli, elisabeth.oswald, martijn.stam@bristol.ac.uk

Abstract. Lattice-based schemes are among the most promising post-quantum schemes, yet the effect of both parameter and implementation choices on their side-channel resilience is still poorly understood. Aysu et al. (HOST'18) recently investigated single-trace attacks against the core lattice operation, namely multiplication between a public matrix and a "small" secret vector, in the context of a *hardware* implementation. We complement this work by considering single-trace attacks against software implementations of "ring-less" LWE-based constructions.
Specifically, we target *Frodo*, one of the submissions to the standardisation process of NIST, when implemented on an (emulated) ARM Cortex M0 processor. We confirm Aysu et al.'s observation that a standard divide-and-conquer attack is insufficient and instead we resort to a sequential, extend-and-prune approach. In contrast to Aysu et al. we find that, in our setting where the power model is far from being as clear as theirs, both profiling and less aggressive pruning are needed to obtain reasonable key recovery rates for SNRs of practical relevance. Our work drives home the message that parameter selection for LWE schemes is a double-edged sword: the schemes that are deemed most secure against (black-box) lattice attacks can provide the *least* security when considering side-channels. Finally, we suggest some easy countermeasures that thwart standard extend-and-prune attacks.

Keywords: Side-channel analysis · LWE · Frodo · Template attacks · Lattices

1 Introduction

Recent advances in quantum computing [7,8] have accelerated the research into schemes which can be used as replacements for currently popular public-key encryption, key-exchange and signature schemes, all of which are vulnerable to quantum attacks. The attention of the cryptographic research community in this direction is boosted by the current NIST standardisation process [16].

Investigating the security of new public-key cryptography proposals in different security settings is an important part of this standardisation process. The

The current document is the pre-proceeding version of the paper, which was accepted at the Selected Areas in Cryptography (SAC) 2018 conference.

current trend, in the era of Internet of Things (IoT), is to connect more and more devices and enable them to transmit sensitive data to other devices or the cloud. These IoT devices can often be physically accessed by potential adversaries, allowing for side-channel attacks. However, the challenges when implementing these novel post-quantum schemes are not as well analysed as for the RSA or ECC-based systems they aim to replace.

Over a third of the submissions to NIST's standardisation process are lattice-based constructions [16]. They come in a number of flavours, of which the two dominant classes are those based on learning with errors (LWE [17]) and its variants (Ring-LWE [11] and Module-LWE [9]). For both scenarios, the key to be recovered is typically a vector of relatively small integers, but the computations involving this vector differ considerably: Ring-LWE and Module-LWE often rely on the Number-Theoretic Transform (NTT) to compute polynomial multiplication, whereas standard LWE depends on textbook matrix–vector multiplication.

One of the standard LWE-based proposals is Frodo. Originally conceived as a key agreement protocol it was expanded to a Key Encapsulation Mechanism (KEM), for the later NIST submission [5,15]. Frodo relies on the equation $\mathbf{B} = \mathbf{AS} + \mathbf{E}$, where $\mathbf{A}, \mathbf{B}, \mathbf{S}$, and $\mathbf{E}$ are all various matrices over $\mathbb{Z}_q$ for q a power of two. The dimensions of these matrices, the modulus q, as well as the distributions from which the error $\mathbf{E}$ and the secret $\mathbf{S}$ are drawn, are all parameters to the scheme. Overall, the Frodo designers proposed six concrete parameter sets, yet the natural resistance of the corresponding matrix multiplication against side-channel analysis is still understood only partially.

Recently, Aysu et al. [2] demonstrated the efficacy of horizontal Correlation Power Analysis (CPA) in a single trace setting against Frodo's matrix multiplication $\mathbf{AS}$ when implemented in hardware. Their attack assumes knowledge of the architecture in order to target specific intermediate registers, as well as that the Hamming distance is a good approximation of *their specific* device's leakage. Even so, for a distinguisher to succeed, knowledge of the algorithm's state so far is required. Aysu et al. cope with this challenge by describing what is known as an extend-and-prune strategy. Seemingly unaware that their method is essentially part of the established methodology of template attacks [6], they do not further explore challenges that may arise in contexts where the device's leakage is too far from Hamming weight/distance for an unprofiled method to work.

Our contribution. We fill this gap by investigating single-trace attacks against software implementations of "ring-less" LWE-based constructions, as used by Frodo. When Frodo is used as key agreement protocol, the secret $\mathbf{S}$ is ephemeral and the calculation of $\mathbf{AS} + \mathbf{E}$ that we target is only performed once (or twice), resulting in only a single trace. This limited usage implies only a subset of side-channel techniques apply. When Frodo is used as a KEM, the overall private key (of the KEM) *is* used repeatedly for decapsulation and the usual techniques relying on a variable number of traces do apply. However, even then our work provides useful guidance on security, and indeed, we expect our results can be

translated to any "small secret" LWE scheme, that is any scheme where the individual entries of $\mathbf{S}$ are "small" in the space over which the scheme is defined.

Even if only a single trace corresponding to $\mathbf{AS}+\mathbf{E}$ is available, each *element* in $\mathbf{S}$ is still used multiple times in the calculation of $\mathbf{AS}$, enabling so called horizontal differential power analysis. Here the single trace belonging to $\mathbf{AS}$ is cut up into smaller subtraces corresponding to the constituent $\mathbb{Z}_q$ operations. Hence, the number of subtraces available for each targeted $\mathbb{Z}_q$ element (of $\mathbf{S}$) is bounded by the dimension of the matrix $\mathbf{A}$. For square $\mathbf{A}$ as given by the suggested parameters, this immediately leads to a situation where high dimensions for $\mathbf{A}$, thus $\mathbf{S}$, on the one hand imply more elements of $\mathbf{S}$ need to be recovered (harder), yet on the other hand more subtraces per element are available (easier). To complicate matters, the elements of $\mathbf{S}$ are chosen to be relatively small in $\mathbb{Z}_q$, with the exact support differing per parameter set. All in all, the effect of parameter selection on the natural side-channel resistance is multi-faceted and potentially counterintuitive; we provide guidance in this respect in Section 5.

For our investigation, we opted for the ARM Cortex M0 as platform for Frodo's implementation. The Cortex-M family has high practical relevance in the IoT panorama, where our choice for the M0 is primarily instigated by the availability of the ELMO tool [13], which we use to simulate Frodo's power consumption (see Section 2 for details). We believe our results are representative for other 32-bit ARM architectures as well.

Our first research question is how well the unprofiled correlation power analysis, as successfully deployed by Aysu et al. [2] against a hardware implementation of Frodo, works in our software-oriented context. The main operations relevant for Frodo are $\mathbb{Z}_q$ addition and multiplication, which are both known to be poor targets for side-channel attacks [4,10]. This is usually compensated for by employing a larger number of traces and by using a power model sufficiently close to the device's leakage profile. The former is intrinsically not possible in the setting we consider, while the latter necessarily requires a profiling phase in cases where the leakage profile of a device is not well-known (as is the case for registers leaking Hamming distance in Aysu et al.'s case).

Overall, we target up to three points of interest, corresponding to loading of a secret value, the actual $\mathbb{Z}_q$ multiplication, and updating an accumulator with the resulting product. For a classical divide-and-conquer attack, where all positions of the secret matrix $\mathbf{S}$ are attacked independently, the templates can easily be profiled at the start, but as we find in Section 3, the resulting algorithmic variance is too high to allow meaningful key recovery.

Therefore we switch to an extend-and-prune technique (Section 4), allowing inclusion of predictions on intermediate variables (such us partial sums stored into an accumulator). This approach drastically reduces the algorithmic variance and hence increases the effective signal strength. We show how different pruning strategies allow for a trade-off between performance and success, concluding that for reasonable levels of success, this type of pruning needs to be less aggressive than that employed by Aysu et al. [2]. We also find that of the two Frodo

parameter sets given in the NIST proposal, the one designed for higher security is in fact the most vulnerable against our side-channel cryptanalysis.

We finish with a discussion on possible countermeasures (Section 5). In particular, we propose a simple alternative way of evaluating the matrix multiplication that frustrates the extend-and-prune attack, reintroducing the algorithmic variance effectively for free. This deterministic method significantly improves the security of what is otherwise still an unprotected implementation.

2 Preliminaries

Notation. Vectors are denoted by lower case boldface letters and the i-th component of a vector $\mathbf{v}$ is $\mathbf{v}[i]$, where indexing starts at 1. Matrices are denoted by upper case boldface letters and their elements are also indexed using square brackets notation in row major order. The n-dimensional identity matrix is denoted by $\mathbf{I}_n$.

Drawing a random sample x from a distribution $\mathcal{D}$ over a set S is denoted by $x \leftarrow\!\!\$\, \mathcal{D}(\mathrm{S})$ or just by $x \leftarrow\!\!\$\, \mathcal{D}$ if the set is clear from the context. We denote drawing a random vector of dimension n made of independent and identically distributed random samples by $\mathbf{x} \leftarrow\!\!\$\, \mathcal{D}^n(\mathrm{S})$. The support of $\mathcal{D}$, i.e. the values to which $\mathcal{D}$ assigns non-zero probability, is denoted by $\mathrm{Supp}(\mathcal{D})$.

2.1 Frodo: a LWE-based Key Agreement Protocol/KEM

Originally Frodo was conceived as a key agreement protocol [5]; in the later NIST proposal [15], it was recast as a KEM. It derives its security from a variant of Regev's LWE concept [17], namely the *decisional Matrix-LWE problem with short secrets* (Definition 1), which stipulates secrets and errors as matrices of fixed dimensions, instead of vectors of arbitrary dimension.

Definition 1 ([5, Section 5.1]). *Let $n, m, q, \overline{n}$ be positive integers and χ be a distribution over $\mathbb{Z}_q$. Let $\mathbf{A} \leftarrow\!\!\$\, \mathcal{U}^{m\times n}(\mathbb{Z}_q)$ where $\mathcal{U}$ is the uniform distribution, $\mathbf{E} \leftarrow\!\!\$\, \chi^{m\times\overline{n}}(\mathbb{Z}_q)$ and $\mathbf{S} \leftarrow\!\!\$\, \chi^{n\times\overline{n}}(\mathbb{Z}_q)$. Defining $\mathbf{B}$ as $\mathbf{B} = \mathbf{AS} + \mathbf{E}$, the* decisional Matrix-LWE problem with short secrets *asks to distinguish $(\mathbf{A}, \mathbf{B})$ from $(\mathbf{A}, \mathbf{U})$, where $\mathbf{U} \leftarrow\!\!\$\, \mathcal{U}^{m\times\overline{n}}(\mathbb{Z}_q)$.*

Frodo can be instantiated with six different parameter sets, four proposed in the original key agreement protocol [5] and two as part of the NIST submission [15]. Table 1 summarises them all. Matrix dimensions are specified, as well as k, the cardinality of the support of χ. The latter distribution is a discrete Gaussian centred at zero, with range $[-\eta, +\eta]$ for $\eta = (k-1)/2$. This effectively specifies all possibilities for each secret entry.

The core operation of Frodo is the calculation of $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$. Without loss of generality, we will henceforth concentrate on only a single column of the secret matrix $\mathbf{S}$, which will be denoted by $\mathbf{s}$. Thus we target the operation $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$, where we try to recover the small value $\mathbf{s}$ for known $\mathbf{A}$ and $\mathbf{b}$ based on the leakage from primarily the matrix–vector multiplication $\mathbf{As}$. We note that, given $\mathbf{A}$ and

Name	n	q	k
CCS1	352	2^{11}	7
CCS2	592	2^{12}	9
CCS3	752	2^{15}	11
CCS4	864	2^{15}	13
NIST1	640	2^{15}	23
NIST2	976	2^{16}	21

Table 1: Parameter sets for Frodo where $k = |\mathrm{Supp}(\chi)|$; for all of sets, $m = n$ and $\bar{n} = 8$.

$\mathbf{b}$, it is possible to check whether a guess $\mathbf{s}$ is correct by checking whether $\mathbf{b} - \mathbf{As}$ is in the support of χ. This suffices with very high probability, because a wrong $\mathbf{s}$ would make the result pseudorandom.

Our analysis of a single column recovery $\mathbf{s}$ could easily be extrapolated to the recovery of the full secret matrix $\mathbf{S}$ by taking into account the number of columns $\overline{n}$ and the fact that columns can be attacked independently. Furthermore, for the original Frodo key agreement, a subsequent step in the protocol to arrive at a joint secret, the so-called reconciliation, is component-wise. Consequently, correctly recovering one column of $\mathbf{S}$ immediately translates to recovering part of the eventual session key (between 8 and 32 bits, depending on the selected parameter set). A similar argument applies to the public key encryption scheme on which the KEM variant [15] is based. However, the introduction of hash functions in the final KEM protocol structurally prevents such a threat and full recovery of $\mathbf{S}$ is required.

While we focus on Frodo's operation $\mathbf{As}$, our results apply equally to the transpose operation $\mathbf{s}^\intercal\mathbf{A}$, or indeed to any scenario where a small secret vector is multiplied by a public matrix and there is a method to test (as in the case for LWE) with high probability whether a candidate $\mathbf{s}$ is correct. While we concentrate on the parameter sets relevant to Frodo (which has relatively leak-free modular reductions due to its power-of-two modulus q), the techniques apply to other parameter sets used in different LWE-based schemes as well.

Matrix–vector multiplication. Algorithm 1 contains the high level description of textbook matrix–vector multiplication. This is usually deployed as asymptotically faster methods have overhead which makes them unsuitable for the matrix dimensions found in practical lattice-based schemes.

For every iteration of the outer loop, the accumulator *sum* is initialised to zero and updated n times with as many $\mathbb{Z}_q$ multiplications. This means that for every secret entry $\mathbf{s}[i]$ an adversary can exploit n portions of the power trace, namely each time it is used in Line 5, motivating the use of a horizontal attack.

Note that Line 5 does not include an explicit modular reduction. As the modulus q is a power of two, the accumulator *sum* is allowed to exceed q and will only be reduced modulo q when it is added to the error in Line 6. The

Algorithm 1 Matrix–vector multiplication as implemented in Frodo.

Input: $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$; $\mathbf{s}, \mathbf{e} \in \mathbb{Z}_q^n$
Output: $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$

1: $\mathbf{b} \leftarrow \mathbf{e}$
2: **for** $r = 1, \ldots, n$ **do**
3: $\quad sum \leftarrow 0$
4: $\quad$ **for** $i = 1, \ldots, n$ **do**
5: $\quad\quad sum \leftarrow sum + \mathbf{A}[r, i] \cdot \mathbf{s}[i]$
6: $\quad \mathbf{b}[r] \leftarrow (\mathbf{b}[r] + sum) \bmod q$
7: **return b**

modular reduction itself boils down to truncation and similarly, in the earlier Line 5 *sum* will of course be reduced modulo the word size, in our case 32 bits.

2.2 Template attacks

Template attacks were first introduced by Chari et al. [6]. The idea is that an adversary creates statistical descriptions, called templates, of the device's leakage for specific intermediate values by profiling the target device (or an equivalent one). Subsequently, one can use Bayesian methods (e.g. maximum likelihood estimation) to determine which template best matches the observed leakage, eventually leading to key recovery.

We consider two classes of template attack. For *divide-and-conquer* the secret is split up into many sub-secrets that are recovered *independently* of each other, and subsequently these sub-secrets are recombined. In our case, it would entail recovering the components of the secret vector $\mathbf{s}$ independently of each other. Divide-and-conquer is popular for instance in the context of AES-128 and has the advantage that profiling can easily be done during a preprocessing stage.

Chari et al. already observed that for their use case (RC4), divide-and-conquer was insufficient. Instead they suggested an extend-and-prune approach, where the overall secret is still split up into many sub-secrets, but this time they are recovered *sequentially*. As a result, when recovering the ith sub-secret, it is possible to use knowledge of the preceding $i - 1$ sub-secrets to select more potent templates. The total number of possible templates increases drastically and, while it might still be just about feasible to generate them all as part of preprocessing, it is more common to generate the actually required templates on-the-fly [3].

We analyse both strategies. In Section 3 we attack the individual sub-secrets independently using divide-and-conquer. This implies that the templates necessarily cannot rely on the value of the accumulator *sum* as that depends on all the previous sub-secrets. Subsequently, in Section 4, we consider the extend-and-prune approach, generating templates on-the-fly, which allows us to profile based on the (likely) correct value of the accumulator.

2.3 Experimental Setup

As target architecture for our experiments we chose the entry level ARM architecture, the Cortex series, because it represents a realistic target and is extremely widely distributed. The Cortex series has several family members, and for the M0 a high quality leakage modelling tool exists. Understanding different attack strategies on different noise levels requires many experiments (we used well over 10^6 full column traces per parameter set), which becomes problematic on real devices. Thus we opted to use simulated *yet realistic* traces which are quicker to generate, modify, and analyse. This allowed us to speed up our analysis, and therefore enable the exploration of a wider noise spectrum.

ELMO. ELMO [12] is a tool to simulate instantaneous power consumption for the ARM Cortex M0 processor. This simulator, created by adapting the open-source instruction set emulator Thumbulator [19], has been designed to enable side-channel analysis without requiring a hardware measurement setup. ELMO takes ARM thumb assembly as input, and its output describes the power consumption, either at instruction or cycle accuracy. The resulting traces are noise free, that is, they are based deterministically on the instructions and their inputs.

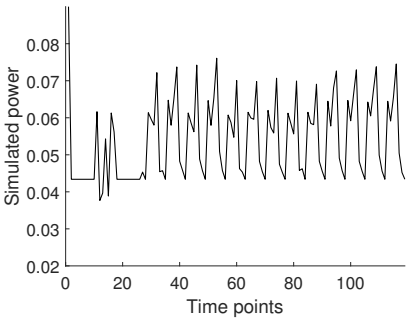
ELMO's quality has been established by comparing leakage detection results between simulated and real traces from a STM32F0 Discovery Board [13]. As raw ELMO traces are noise free, the tool is ideal to study the behaviour of template attacks across different noise levels efficiently: both template building and creating noisy traces are straightforward.

We stress that ELMO does capture differential data-dependent effects, such as those caused by neighbouring instructions, as well as higher order leakage terms. Consequently, even though ELMO traces are noise free, the trace for the same machine line of code (same operation with the same operand) will differ depending on the context, leading to *algorithmic variance* (i.e. variation in the trace that deterministically depends on those parts of the input currently not being targeted).

Reference implementation. We implement the innermost loop of Algorithm 1 in ARM assembly, which for convenience we wrapped in C code for initialization and loop control. This gives us a fine control over the code ELMO simulates the power consumption of and prevents the compiler from inserting redundant instructions which might affect leakage. We refer to Appendix A for the full code, which is then just repeated n times.

Figure 1a plots a partial power trace of our ARM implementation, as simulated by ELMO. After initialisation, a pattern neatly repeats, corresponding to the equivalent of Line 5 in Algorithm 1. After excluding unimportant points (e.g. loop structure), the most relevant instructions responsible for the pattern are given in Table 1b.

The index i stored in `r4` is used to load values from a row of $\mathbf{A}$ and $\mathbf{s}$, whose addresses are in `r1` and `r0` respectively, into `r6` and `r5`. These are then used to



(a) Power trace as simulated by ELMO of our ARM implementation

Instruction	Operation
`ldrh r5,[r0,r4]`	load $\mathbf{s}[i]$
`ldrh r6,[r1,r4]`	load $\mathbf{A}[r, i]$
`muls r5,r6`	$\mathbf{s}[i] \cdot \mathbf{A}[r, i]$
`adds r3,r3,r5`	$sum + \mathbf{s}[i] \cdot \mathbf{A}[r, i]$

(b) Breakdown of instructions forming the repeating pattern.

Fig. 1: Visual representation and detailed structure of target power traces.

perform one element multiplication, whose result overwrites `r5`, and finally the accumulator is updated in `r3` and eventually returned.

We wrap around negative numbers modulo q. This is in contrast to Frodo's original convention of taking 16-bit cut-off independently on the parameter set. We expect the higher Hamming weights resulting from modulo-2^{16} wraparound to amplify leakage, thus making our decision, motivated by simplicity of analysis, very conservative. Finally, intermediate multiplications and partial sums are truncated only when exceeding 32 bits, being the M0 a 32-bit architecture.

Realistic noise estimate. As mentioned before, ELMO traces are noise free. However, when attacking an actual ARM Cortex M0 environmental noise will be introduced. For our experiments, we will artificially add this noise, which we assume independently and identically distributed for all points of interest, according to a normal distribution with mean 0 and variance σ^2.

For the profiling that led to the development of ELMO [13], the observed value[4] of σ was around $4 \cdot 10^{-3}$. We will use this realistic level of environmental noise as benchmark throughout. Furthermore, we will consider a representative range of σ roughly centred around this benchmark. We chose σ in the interval $[10^{-4}, 10^{-2})$ with steps of $5 \cdot 10^{-4}$. Compared to the variance of the signal, our choice implies σ ranges from having essentially no impact to being on the same order of magnitude.

3 Divide-and-Conquer Template Attack

As every entry of $\mathbf{s}$ is an independently and identically distributed sample from χ, we can potentially target each position separately. Thus we first consider a divide-and-conquer template attack. A distinct advantage of this approach is

[4] Personal communication with C. Whitnall.

that the total number of templates is fairly small and hence we can preprocess the profiling.

When considering the breakdown of the inner loop (Table 1b), we ignore the loading of the public operand (it essentially leaks nothing exploitable), which leaves three potential points of interest. On the one hand, the loading of the secret operand and the multiplication contain direct leakage on the secret, and all relevant inputs appear known. For the accumulator update on the other hand, the leakage is less direct and the value of the accumulator so far cannot be taken into account: it depends on the computation so far, violating the independence requirement for divide-and-conquer. Thus, for the attack in this section we limit ourselves to *two* points of interest, namely the loading of the secret and the $\mathbb{Z}_q$ multiplication.

Of course, one could still generate templates for all *three* points of interest by treating the accumulator as a random variable. However, as the accumulator value is a direct input to the accumulator update and its register is used for the output as well, the resulting algorithmic variance would be overwhelming. Indeed, as we will see below, already for the loading of the secret there is considerable algorithmic variance related to the previous value held by the relevant register. These limitations are intrinsic to a divide-and-conquer approach; in Section 4 we show how an extend-and-prune approach bypasses these problems.

Profiling. One feature of LWE instances is that the overall space $\mathbb{Z}_q$ from which elements are drawn is fairly small as q need not be large, certainly compared to classical primitives like ECC or RSA. For Frodo, and in general for "small secret" schemes, the effective space that requires profiling is further reduced as the support of χ (from which secrets are drawn) is even smaller.

For the loading of the secret, we need k templates, whereas for the multiplication $k \cdot q$ templates suffice. We generate these templates as part of the preprocessing, where we are primarily interested in the signal, that is the deterministic part.

Although ELMO is completely deterministic, the power trace it emulates for a given operation still depends on preceding operations, thus introducing algorithmic variance. To profile the loading of secret s, we use the weighted average of k traces, corresponding to the previous value of the register involved, as the deterministic part. For reference, depending on the parameter set, the algorithmic variance is between $1.4 \cdot 10^{-3}$ and $2.9 \cdot 10^{-3}$. For the multiplication, we assumed no algorithmic variance in our profiling and simply performed the operation once for each template.

Estimating success rates. For each entry $\mathbf{s}[i]$, the distinguisher outputs a distinguishing score vector that can be linked back to a perceived posterior distribution. Selecting the element corresponding to the highest score corresponds to the maximum a posteriori (MAP) estimate and the probability that the correct value is returned this way is referred to as the first-order success rate.

Ultimately, we are more interested in the first order success rate of the full vector $\mathbf{s}$. As we assume independence for a divide-and-conquer we can easily extrapolate the success rates for $\mathbf{s}$ based on those for individual positions as a full vector is recovered correctly iff all its constituent positions are. The advantage of using extrapolated success rates for $\mathbf{s}$, rather than using direct sample means, is that it provides us useful estimates even for very small success rates (that would otherwise require an exorbitant number of samples). Thus, analysing the recovery rates of single positions is extremely informative. Additionally, it gives insights on why the extend-and-prune attack in Section 4 greatly outperforms divide-and-conquer.

Other metrics, beyond first-order recovery rate, are of course possible to compare distinguishers [18]. However, we regard those alternatives, such as oth-order recovery or more general key ranking, only of interest when first order success rate is low. While for divide-and-conquer this might be the case, for extend-and-prune the first order recovery is sufficiently high to warrant concentrating on that metric only.

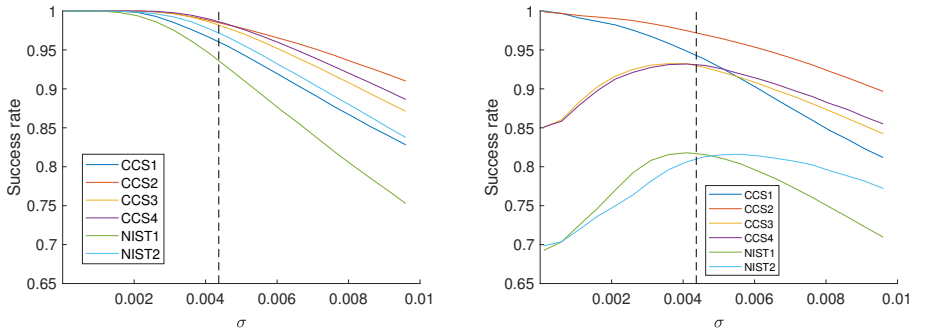
Estimating position success rate. Let $\Pr[S]$ be the first order position recovery rate where S is the event that the distinguisher indeed associates the highest score to the actual secret value. We experimentally evaluate $\Pr[S]$ based on the formula

$$\Pr[S] = \sum_{s \in \mathrm{Supp}(\chi)} \Pr[S \mid s] \Pr[s]$$

where $\Pr[s]$ corresponds to the prior distribution χ and the values for $\Pr[S \mid s]$ are estimated by appropriate sample means. To ensure our traces are representative, we range over $\mathbf{A}$ and $\mathbf{s}$ (and $\mathbf{e}$) for the relevant experiments and generate traces for the *full* computation $\mathbf{b} \leftarrow \mathbf{As}+\mathbf{e}$. This allows us to zoom in on individual positions, highlighting where algorithmic variance occurs. While one could also use direct, position-specific sample means for $\Pr[S]$, our approach links more closely to the confusion matrix and has the advantage that it depends less on the sampling distribution of $\mathbf{s}$ when running experiments.

Extrapolating overall success rate. If we assume independence of positions, it is easy to express the overall success rate for recovering $\mathbf{s}$. If we, temporarily, make the simplifying assumption that $\Pr[S]$ is the same for all n positions, then the first order recovery rate for $\mathbf{s}$ is $\Pr[S]^n$ (recovery of $\mathbf{s}$ will be successful if and only if recovery of each of its elements is). Even for extremely high $\Pr[S]$, this value quickly drops, e.g. $0.99^n \approx 5.5 \cdot 10^{-5}$ for NIST2.

Experimental results. We target each position of $\mathbf{s}$ individually, but only report on the first and second one. Fig. 2 displays the success rate for all parameter sets. Each point in each curve is based on $8 \cdot 10^5$ experiments. The left panel (Fig. 2a) plots the success rate for the first position, whereas the right panel (Fig. 2b) plots it for the second position. The second position is representative for all subsequent positions, but the first position stands out as being significantly easier to tackle due to the lack of algorithmic variance.



(a) Recovery rate for first position only. (b) Recovery rate for second position only.

Fig. 2: Comparison of recovery rates between first and second positions. The dashed black line indicates our choice of realistic noise level.

The impact of algorithmic variance. The striking difference between Figs. 2a and 2b, especially in the low environmental noise regime, is due to algorithmic variance. As we mentioned before, algorithmic variance particularly affects the loading of the secret, i.e. the first instruction in Table 1b, due to the previous register value contributing to the leakage. This problem only appears from the second position onward; for the first position, no algorithmic variance is present as the initial state is fixed (and profiled for).

With the exception for the two small CCS parameter sets, even with virtually no environmental noise, the success rate for the second position is far from 1. Moreover, when environmental noise is added, the success rate initially goes up. This phenomenon is known as stochastic resonance [14] and has been observed for side-channels before [20]. Even for CCS1 and CCS2, that have the lowest algorithmic variance level, the success rate for the second position is slightly lower than for the first position.

For completeness, our assumption that the noise covariance matrix $\mathbf{\Sigma}$ for our two points of interest is a diagonal matrix $\sigma \cdot \mathbf{I}_2$, is suboptimal in the presence of algorithmic variance. Using a diagonal matrix $\mathbf{\Sigma}$ that incorporates the algorithmic variance would improve the distinguisher while reducing the stochastic resonance. As the extend-and-prune approach from the next section is far more convincing, we refrain from a full analysis.

Full vector recovery. The success rates for full vector are more relevant to compare either amongst parameter sets or with other attacks, be it lattice or other side-channel attacks. As a simplification, we assume that the recovery rate for the second position (Fig. 2b) is representative for all positions: we checked this assumption holds for all bar the first position, whose contribution is limited anyway given concrete values of n (the total number of positions).

To ease comparison, for each parameter set we determined the σ for which the divide-and-conquer attack approximately achieves a success rate for recovering $\mathbf{s}$ of around 2^{-128} (corresponding to 128-bit security). For the smallest parameter sets, CCS1 and CCS2, all the σ in our range are susceptible (i.e. lead to success rates of at least 2^{-128}), whereas for the NIST parameter sets, none of the σ appear insecure. For the original large sets CCS3 and CCS4, any σ below $7 \cdot 10^{-3}$, which includes our realistic benchmark, leads to a loss of security below the 128-bit level.

As a caveat, a further reduction in residual bit security will be possible by explicitly incorporating algorithmic variance in the templates and by considering key ranking, or possibly even novel lattice reduction algorithms that take into account side-channel information. However, we anticipate none of these approaches will allow straightforward and almost instant key recovery for all parameter sets for realistic levels of noise (as introduced by σ).

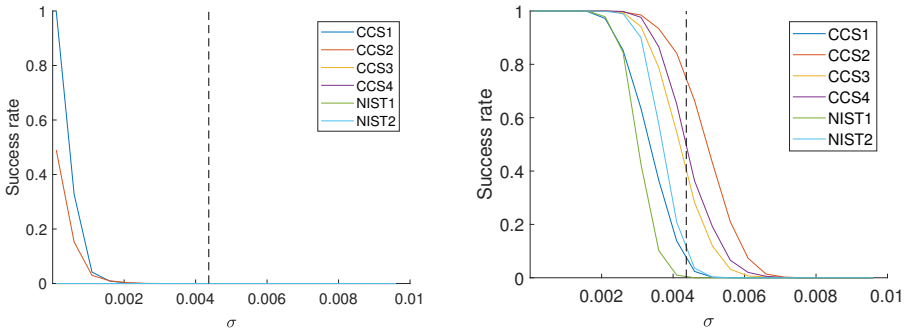
4 Extend-and-Prune Template Attack

For the divide-and-conquer approach from the previous section, we assumed that the positions of $\mathbf{s}$ are independent of each other. While this assumption is valid for the generation of $\mathbf{s}$, it turned out that for the leakage, it is not. However, Algorithm 1 deals with the elements of $\mathbf{s}$ *sequentially*, from position 1 to position n, which we will exploit by a well-known extend-and-prune approach.

In our case, the extend-and-prune algorithm operates as follows. We imagine a k-ary tree of depth n where the nodes at level i in the tree correspond to a partial guess $\mathbf{s}[1], \ldots, \mathbf{s}[i-1]$ for the secret; for a given node at level i, its k out-going edges are labelled by the k possible values that $\mathbf{s}[i]$ can take. This way, each path from the root to one of the k^n possible leaves uniquely corresponds to one of the possible values that the secret vector $\mathbf{s}$ can take. A distinguisher can sequentially calculate a score for a vector $\mathbf{s}$ by traversing the tree from the root to the leaf representing $\mathbf{s}$ where for each edge it encounters it cumulatively updates $\mathbf{s}$'s score.

The challenge of an extend-and-prune algorithm is to efficiently traverse a small part of the tree while still ending up with a good overall score. The standard way of doing so is to first calculate the score for all nodes at level 2. For each level-2 node, the score will be that of the edge from the root to that node. Thus the trivial level-1 guess is *extended* to all possible level-2 guesses. The next stage is to *prune* all these guesses to a more reasonable number. For all the remaining level-2 guesses, one then extends to all possible level-3 guesses, and then again these guesses are pruned down. This process repeats until reaching the final level $(n+1)$, where the complete $\mathbf{s}$ is guessed.

The advantage of this approach is that, when calculating a score for $\mathbf{s}[i]$, the distinguisher already has a guess for $\mathbf{s}[1], \ldots, \mathbf{s}[i-1]$, which allows it to create templates based on this guess. Our distinguisher will only use the previous secret $\mathbf{s}[i-1]$ and the value of the accumulator so far (an inner product of $(\mathbf{s}[1], \ldots, \mathbf{s}[i-1])$ with the relevant part of $\mathbf{A}$) to create a template. As the total



(a) Column recovery rate of divide-and-conquer template attack.

(b) Column recovery rate of extend-and-prune template attack.

Fig. 3: Comparison between column recovery of our two template attacks.

number of possible templates becomes rather unwieldy (around $k^2 \cdot q \cdot 2^{32}$), the profiling is interleaved with the tree traversal and pruning is used to keep the number of templates manageable.

The success of an extend-and-prune attack depends on the pruning strategy, specifically how many candidates to keep at each step. To the best of our knowledge, there is no comprehensive study comparing different pruning strategies in different scenarios. When Chari et al. [6] introduced template attacks to the cryptanalyst's arsenal, they suggested a pruning strategy that depends on the scores themselves. We instead fix the same number of candidates to keep at each step, which is a classical approach known as *beam search*. The size of the beam, that is the number of candidates to keep after pruning, is denoted by b.

Greedy pruning using a laser beam ($b = 1$). We start by considering the greediest pruning strategy by restricting the beam size to $b = 1$, meaning that after each step we only keep a single candidate for the secret recovered so far. This "knowledge", provided it is correct, has two very immediate effects. Firstly, the algorithmic variance we observed in the loading of the secret can be reduced as we assume we typically know the previous secret held by the relevant register. Secondly, by recovering $\mathbf{s}$ from first to last we can predict the value of the accumulator, which brings into play a third point of interest, namely the update of the accumulator (the last point in Table 1b), as here too the algorithmic variance disappears.

Fig. 3 presents the vector recovery rates of both last section's divide-and-conquer attack (in the left panel, Fig. 3a), and of extend-and-prune using $b = 1$ (Fig. 3b). Note that the former is extrapolated based on position recovery rates, whereas the latter has been estimated directly, based on $2 \cdot 10^3$ experiments per setting.

The difference between Figures 3a and 3b is striking. For the extend-and-prune approach we almost completely removed algorithmic variance and, when virtually no environmental noise is present either ($\sigma \approx 10^{-4}$), this resulted in a vector recovery rate of essentially 1. However, when considering the realistic noise level as indicated by the dashed vertical line, not all parameter sets are as affected and especially for NIST1 there might be still some hope (for the other parameters, recovery rates exceed 5% which translates to less than 5 bits of security, so badly broken).

Increasing the beam size ($b > 1$). So far we only considered $b = 1$. Increasing the beam size b will result in a slower key recovery (linear slowdown in b) but should yield higher recovery rates. For $b = 1$ we mentioned two advantages of extend-and-prune, namely reduced algorithmic variance and an additional point of interest. For $b > 1$ a third advantage appears, namely the ability for the distinguisher to self-correct. This self-correcting behaviour has also been observed (for the first position) by Aysu et al. [2], who essentially used a beam size $b > 1$ for the first position and then revert to $b = 1$ for all remaining ones.

Name	$b_{\min}$	b								
		2	3	4	5	6	7	8	9	10
CCS1	30709	0	0	0	0	0	0	0	0	0
CCS2	27	0.1	0.13	0.36	0.53	0.68	0.76	0.85	0.90	0.94
CCS3	12	0	0.48	0.77	0.90	0.94	0.96	0.99	0.99	0.99
CCS4	11	0.03	0.63	0.91	0.97	0.97	0.98	0.98	0.99	0.99
NIST1	63	0	0	0.01	0.03	0.13	0.24	0.33	0.41	0.50
NIST2	11	0	0.07	0.63	0.84	0.96	0.99	0.99	0.99	0.99

Table 2: Minimum values of b to achieve column recovery rate equal to 1, and heuristic column recovery when b is fixed to the listed values.

To assess the effect of the beam size b, we ran two types of experiments. Firstly, for each parameter set and noise level $\sigma = 0.0096$, we ran around 10^3 experiments and looked at the smallest beam b for which all experiments ended with the actual secret $\mathbf{s}$ part of the final beam (allowing an adversary to identify $\mathbf{s}$ by a subsequent enumeration of all final beam candidates). The resulting values are reported in the $b_{\min}$ column of Table 2. With the exception of CCS1, we notice that $b_{\min}$ is at most 2^6, so again only a few bits of security remain. As $b_{\min}$ will invariably grow as the number of experiments does, until eventually it is as large as the key space, for our second set of experiment, we estimated final vector recovery rate as a function of the beam size, for $b \leq 10$. The results are again reported in Table 2 and are fairly damning: even for NIST1 a recovery rate of around 50% is achieved.

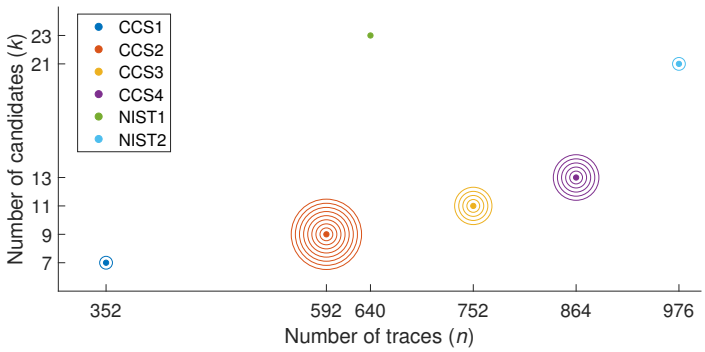


Fig. 4: Visual representation of all parameter sets. For each of them, the x axis lists n, and the y axis lists k. The number of concentric circles around each parameter set encodes how successful our attack is against it.

5 Learning the Lesson: How to Thwart Extend-and-Prune

Choosing your parameters. So far, we have compared increasingly effective attack strategies, where we compared different parameter sets purely by name, so without further reference to their actual parameters. We now investigate the effect of these parameters on the efficacy and efficiency of the attack. Specifically, we consider the effects of n and k on the natural side-channel vulnerability of the resulting matrix–vector multiplication. We completely ignore the effect on the security of the LWE instance and indeed, leave the combination of side-channel information with lattice reduction methods a tantalizing open problem.

Figure 4 provides a scatter plot of (n, k) for the various parameter sets suggested [5,15]. Furthermore, we encoded the success rate of our extend-and-prune attack with beam $b = 1$ (Section 4) and realistic noise level (dashed line in Figure 3b) with concentric circles around each parameter set. The number of circles is simply the ceil of said success rate times ten, and is helpful in visually quantifying the outcome we achieved in each setting.

The effect that the choice (n, k) has on the hardness of the LWE instance has been well studied [1], but from a side-channel perspective, new meaning emerges: n corresponds both to the number of (sub)traces an adversary obtains on each component of $\mathbf{s}$ *and* to the number of positions to retrieve, whereas k quantifies the keyspace size for individual positions.

Although the divide-and-conquer attack suffers badly when more positions need to be recovered, the extend-and-prune approach is far more robust in this respect. For instance, the main difference between CCS1 and CCS2 is that the latter has a twice as big n, thus providing a much easier target for our attack. Thus increasing n overwhelmingly has the effect of making life easier for an adversary as more leakage will be available. In other words, while increasing the dimension n makes the LWE instance *harder*, it makes the underlying

matrix–vector multiplication *easier* to attack in our side-channel scenario. This conclusion does rely on square $\mathbf{A}$, so $n = m$. In case $\mathbf{A}$ is a non-square matrix, then m refers to the number of traces and n to the number of positions to recover. The hardness of LWE appears is mainly governed by n, where increasing n makes both the LWE instance harder and it complicates side-channel cryptanalysis. Similarly, both for LWE and for the side-channel analysis, increasing m makes attacks potentially easier, with the effect for side-channels much, much more pronounced.

The qualitative effect of increasing k is fairly straightforward: a large keyspace means that there are more options to choose from, with corresponding signals that are closer together, making distinguishing harder. This effect is illustrated by comparing the two parameter sets NIST1 and CCS2. These two sets have roughly equal n, but NIST1's k is about thrice that of CCS2: our attacks confirm that CCS2 is a lot easier to attack than NIST1.

Effect of modifying NIST1. We conducted a final experiment to gain more insights on parameter set selection. We focused our attention on the two NIST parameter sets: they have roughly the same k (it differs by only two) but NIST1 has less than two thirds less traces than NIST2. We therefore increased n in NIST1 to match NIST2's ($n = 976$) and analysed the extend-and-prune attack in two settings: when $b = 1$ and σ is our realistic value, and when $b = 10$ and $\sigma = 0.0096$, i.e. the worst noise level we consider. In the former case the success rate increased from 0.01 to 0.11, almost equating the success rate of 0.12 observed in the NIST2 setting. In the $b = 10$ case, the success rate reported in Table 2 (0.50) skyrocketed to 0.94, again very close to NIST2's. This strongly indicates how having larger matrices, hence more traces per secret element, goes in favour of the adversary. Therefore in general being overpessimistic in the choice of n might prove fatal if side-channel attacks are a concern.

A simple countermeasure. Aysu et al. [2] briefly discuss potential countermeasures, including shuffling, based on the observation that randomness is usually introduced to mitigate DPA attacks. However, randomness for countermeasures can be expensive, so we present a much simpler *deterministic* countermeasure that has the effect of re-introducing algorithmic variance in the system even when attempting an extend-and-prune attack.

In order to reduce algorithmic variance, our extend-and-prune attack relies on the sequential manner in which the textbook $\mathbf{As}$ multiplication processes $\mathbf{s}$: for each inner product of a row of $\mathbf{A}$ with $\mathbf{s}$, the elements of the latter are accessed in the same order. However, there is no reason to do so, and we suggest to calculate the rth inner product starting at position r instead. This corresponds to changing Line 5 of Algorithm 1 to

$$sum \leftarrow sum + \mathbf{A}[r, (i + r - 1) \mod n] \cdot \mathbf{s}[(i + r - 1) \mod n] \ .$$

The consequence is that there is no longer a clear ordering of $\mathbf{s}$'s elements for an extend-and-prune attack to exploit and, without novel ideas, the attack's success degrades to that of the earlier divide-and-conquer one (Section 3).

A natural alternative to frustrate extend-and-prune is to mask the accumulator by setting it to some random value at the beginning, that is only subtracted at the very end. While this alternative would make exploiting the accumulator update hard (as for divide-and-conquer), on its own it would still allow an extend-and-prune attack to reduce algorithmic variance in the loading of the secrets. Thus our first suggestion is preferable.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme Marie Skłodowska-Curie ITN ECRYPT-NET (Project Reference 643161) and Horizon 2020 project PQCRYPTO (Project Reference 645622). Furthermore, Elisabeth Oswald was partially funded by H2020 grant SEAL (Project Reference 725042). We thank the authors of ELMO for their kind help, comments and feedback.

References

1. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.
2. A. Aysu, Y. Tobah, M. Tiwari, A. Gerstlauer, and M. Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *to appear in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018*, 2018.
3. L. Batina, L. Chmielewski, L. Papachristodoulou, P. Schwabe, and M. Tunstall. Online template attacks. In *INDOCRYPT 2014*, pages 21–36, 2014.
4. A. Biryukov, D. Dinu, and J. Großschädl. Correlation power analysis of lightweight block ciphers: From theory to practice. In M. Manulis, A.-R. Sadeghi, and S. Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 537–557. Springer, Heidelberg, June 2016.
5. J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 16*, pages 1006–1018. ACM Press, Oct. 2016.
6. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski Jr., Çetin Kaya. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, Aug. 2003.
7. M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.
8. J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and J. M. Martinis. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519:66–69, 2015.
9. A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

10. K. Lemke, K. Schramm, and C. Paar. DPA on n-bit sized Boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 205–219. Springer, Heidelberg, Aug. 2004.
11. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May 2010.
12. D. McCann, E. Oswald, and C. Whitnall. Implementation of ELMO. `https://github.com/bristol-sca/ELMO`. Accessed: 27-11-2017.
13. D. McCann, E. Oswald, and C. Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 199–216, 2017.
14. M. D. McDonnell, N. G. Stocks, C. E. M. Pearce, and D. Abbott. *Stochastic Resonance – From Suprathreshold Stochastic Resonance to Stochastic Signal Quantization*. Cambridge University Press, 2008.
15. M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2017. available at `https://frodokem.org/`.
16. National Institute of Standards and Technology. Post-quantum cryptography standardization. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`.
17. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.
18. F.-X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, Heidelberg, Apr. 2009.
19. D. Welch. Thumbulator. `https://github.com/dwelch67/thumbulator.git/`.
20. C. Whitnall and E. Oswald. A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 316–334. Springer, Heidelberg, Aug. 2011.

A ARM assembly code for inner product

Assembly

```
.syntax unified
.text
.thumb

.global Vec_Mult

.func Vec_Mult
Vec_Mult:

 push {r1-r7}
  @Load and prepare the data
  @ i->0
  movs r4, #0
  @ number limit->address limit
  lsls r2, #1
 loop:
  @Load first[i]
  ldrh r5,[r0,r4]
  @Load second[i]
  ldrh r6,[r1,r4]
  @Multiply
  muls r5,r6
  @Add
  adds r3,r3,r5
  @Update i as address
  adds r4,r4,#2
  @Compare with limit
  cmp r4,r2
 bne loop
  @Return Value
  mov r0,r3
  pop {r1-r7}
  bx lr
 .endfunc
```