A Faster Third-Order Masking of Lookup Tables

Anju Alexander, Annapurna Valiveti and Srinivas Vivek

IIIT Bangalore, Bangalore, IN
{anju.alexander,annapurna,srinivas.vivek}@iiitb.ac.in

Abstract. Masking of S-boxes using lookup tables is an effective countermeasure to thwart side-channel attacks on block ciphers implemented in software. At first and second orders, the Table-based Masking (TBM) schemes can be very efficient and even faster than circuit-based masking schemes. Ever since the customised second-order TBM schemes were proposed, the focus has been on designing and optimising Higher-Order Table-based Masking (HO-TBM) schemes that facilitate masking at arbitrary order. One of the reasons for this trend is that at large orders HO-TBM schemes are significantly slower and consume a prohibitive amount of RAM memory compared to circuit-based masking schemes such as bit-sliced masking, and hence efforts were targeted in this direction. However, a recent work due to Valiveti and Vivek (TCHES 2021) has demonstrated that the HO-TBM scheme of Coron et al. (TCHES 2018) is feasible to be implemented on memory-constrained devices with pre-processing capability and a competitive online execution time. Yet, currently, there are no customised designs for third-order TBM that are more efficient than instantiating a HO-TBM scheme at third order.

In this work, we propose a third-order TBM scheme for arbitrary S-boxes that is secure in the probing model and under compositions, i.e., 3-SNI secure. It is very efficient in terms of the overall running time, compared to the third-order instantiations of state-of-the-art HO-TBM schemes. It also supports the pre-processing functionality. For example, the overall running time of a single execution of the third-order masked AES-128 on a 32-bit ARM-Cortex M4 micro-controller is *reduced* by about 80% without any overhead on the online execution time. This implies that the online execution time of the proposed scheme is approximately *eight times* faster than the bit-sliced masked implementation at third order, and it is comparable to the recent scheme of Wang et al. (TCHES 2022) that makes use of reuse of shares. We also present the implementation results for the third-order masked PRESENT cipher. Our work suggests that there is a significant scope for tuning the performance of HO-TBM schemes at lower orders.

Keywords: Side-Channel Attacks, Masking, S-box, Third Order, Probing Leakage Model, SNI Security, Software implementation.

1 Introduction

Side-channel attacks are a major security threat for cryptographic implementations [Koc96, KJJ99]. Masking is an effective countermeasure for side-channel attacks, in particular, differential power/electromagnetic attacks. The popularity of the masking countermeasure is due to its simplicity which paves the way for formal security analysis in the probing leakage model [ISW03, Cor14, BBD+16], and its connection, rather equivalence, to the more realistic noisy leakage model [CJRR99, PR13, DDF14]. Despite the shortcomings of the probing leakage model, for instance, the independent leakage assumption, it continues to attract great attention of the research community.

Masking countermeasures implemented in software, particularly those based on the probing leakage model, can be categorised into two types: circuit-based masking schemes

and Table-based Masking (TBM) schemes. Circuit-based masking schemes, that include schemes proposed in [ISW03, RP10, CGP+12, RV13, CRV14, PV16, CGPZ16, GR16, GR17, GRVV17, WGY+22] and many more, are based on representing the computation, say, of a block cipher, as a boolean or an arithmetic circuit. On the contrary, the TBM schemes, that include schemes proposed in [CJRR99, SP06, RDP08, Cor14, CRZ18, VV20, VV21] represent the non-linear computations as a lookup table. For the case of SPN-based and Feistel-based block ciphers, the S-box is represented as a lookup table that is then masked using TBM schemes. Note, however, that the linear layers of block ciphers are still represented as a circuit as the masking of these layers is relatively efficient.

At first and second orders, the TBM schemes, particularly, [CJRR99, RDP08], are more efficient than circuit-based masking schemes [Vad17]. A particular advantage of most of the TBM schemes is that they support *pre-processing*. Currently this feature is not possible with most of the circuit based schemes. However, a recent work by Wang et al. [WGY+22] has made significant progress in this direction. The authors claim that they can *reuse* all but one share across masked multiplications. Also, their approach facilitates pre-processing of the computation. In general, the pre-processing phase (*a.k.a.* offline phase) and the post-processing phase (*a.k.a.* online phase) refer to the computation that happens before and after the *availability* of the secret input(s), respectively. The goal of the offline/online paradigm is to achieve a faster online phase with the help of the pre-computed results from the offline phase.

In particular, pre-processing in a TBM scheme corresponds to the shifting of a temporary table by all the independently sampled shares, of course, except the final share. Post-processing involves the lookup of the shifted table with the final share as the index and its associated computations. However, at higher orders, the TBM schemes such as [Cor14] and its successor [CRZ18] need an amount of RAM memory that is proportional to the masking order for full pre-processing and, hence, becomes infeasible to implement on resource-constrained devices. One way to overcome the memory requirement of Higher-Order TBM (HO-TBM) schemes is to do sequential processing of the masking of S-boxes and it leads to significantly poor performance compared to the circuit-based masking schemes, particularly, bit-sliced masked implementations such as in [GR17]. A recent work [VV21] demonstrated how the RAM memory requirement can be made essentially constant for the HO-TBM scheme from [CRZ18] . They implemented a fully pre-processed single execution of the AES-128 block cipher with 10 shares (i.e., at ninth order) on an Arm-Cortex M4 microcontroller and showed that the online execution time is competitive with that of bit-sliced masking though the overall execution time was still very high compared to bit-sliced masking.

The HO-TBM schemes such as [Cor14, CRZ18, VV21] shift a temporary table by each input share at a time and do mask refresh for each row of the table after every shift. More concretely, consider an (n, m) S-box S that needs to be looked up with a masked input

$$x = x_1 \oplus x_2 \oplus \ldots \oplus x_k.$$

A temporary table T is first initialised to $(\mathrm{S}, 0, 0, \ldots, 0)$, where all the k columns except the first is set to zero. The rows of the Table T are shifted by each x_i,

$$T(j) \leftarrow T(x_i \oplus j), \quad \forall j \in \{0,1\}^n,$$

by making use of an auxiliary table. After each shift, every row of the Table T is independently refreshed and, hence, these schemes make use of a lot of computing time and (pseudo) randomness. While the complete lack of *mask refresh* can lead to security flaws as noted in [Cor14], an important consequence of these mask refreshes is that the security proofs in the probing leakage model and under compositions, i.e. $k-1$-SNI security proofs [BBD+16], can be written elegantly.

In this work, motivated by the designs of [RDP08, VV20] that make a sparing use of mask refreshes in the second-order security context, we investigate the question of designing a customised *third-order* secure TBM scheme that significantly reduces the number of mask refreshes and, hence, reduces the computation time and randomness usage compared to instantiating the current HO-TBM schemes at third order.

Our Contribution: We propose an efficient third-order secure TBM scheme for arbitrary S-boxes (see Algorithm 3). Our scheme is secure in the probing leakage model and under compositions, more specifically, it is 3-SNI secure [BBD+16]. We design the scheme in two steps. In Step 1, we propose a 3-NI randomised lookup table scheme *without any explicit mask refresh*. Whereas in Step 2, we refresh the output obtained from Step 1 using a 3-SNI mask refresh, the 3-RB procedure, from [BDF+17, BBD+20] (see Algorithm 2). Hence, the composition of these two steps is 3-SNI secure. Our approach requires (explicit) mask refresh only once in the final step, which is a significant reduction in the number of calls to *mask refresh* per (n, m) S-box to *one* from $\left(3 \cdot 2^n + 1\right)$ in the third-order instantiation of [Cor14, CRZ18]. We would like to note that the speedup of the pre-processing step for our scheme comes only from our 3-NI Algorithm 1. We chose the 3-SNI refresh algorithm 3-RB from [BDF+17, BBD+20] instead of the 3-SNI full-refresh mainly to make the online time competitive with the state-of-the-art table-based and circuit-based masking schemes.

Altogether, our proposed third-order randomised lookup table scheme is very efficient compared to the third-order instantiation of the most efficient (in terms of the overall computation time) HO-TBM scheme from [CRZ18]. In the experiments section (Section 3), we demonstrate that our scheme reduces the *total* running time of a single execution of third-order masked AES-128 by 78.87%, and also facilitates full pre-processing. The experiments were run on a 32-bit Arm-Cortex M4 microcontroller. In Table 4, we provide a detailed comparison of our work with the circuit-based implementations of [RP10, GR17, WGY+22]. The online execution time for Algorithm 3 is approximately 8 *times faster* than the bit-sliced masked implementation of AES-128 at third order, and it is comparable with the online execution times for [Cor14, CRZ18, WGY+22] (see Tables 3 and 4).

To further improve the overall execution time, one may try to consider an implementation of the proposed scheme on processors with *large registers*. But, it turns out that the extension of Algorithm 3 to large register variant (LRV) suffers from a second-order attack (see Appendix D). We would like to stress that our basic scheme still beats the overall running time of increasing shares LRV variant of [CRZ18] (see the third row of Table 3) without making use of large registers. For completeness, Table 1 provides estimates for the number of bit operations, RAM memory and randomness usage *per* (n, m) S-box for Algorithm 3, and the HO-TBM schemes from [CRZ18] and [VV21].

Table 1: Comparison of our proposal with the HO-TBM schemes [CRZ18, VV21] instantiated at third order to mask a single (n, m) S-box. The schemes are compared in terms of RAM memory (in bits), true random values (in bits), and the total running time (in number of bit operations). RAM memory includes the number of bits required for the randomised lookup table along with the auxiliary table.

	RAM	#True random	Time
Algorithm 3	$3 \cdot m \cdot 2^n$	$m \cdot (2^n + 4)$	$m \cdot (6 \cdot 2^n + 2 \cdot n)$
[CRZ18] (Increasing shares)	$m \cdot 2^{(n+3)}$	$10 \cdot m \cdot 2^n$	$28 \cdot m \cdot 2^n$
[VV21] (Multiple PRG variant)	$m \cdot \left(2^{(n+1)} + 40 \cdot n\right)$	$40 \cdot n \cdot m$	$m \cdot 2^n \cdot (3 \cdot m + 31)$

The rest of the paper is organised as follows. In Section 2, we present our third-order TBM scheme (Algorithm 3) along with its 3-SNI security proof. The experiment results are presented in Section 3 and the paper concludes with Section 4.

2 Proposed Third-Order TBM Scheme

This section describes our proposal for a third-order TBM scheme. As mentioned previously, the motivation for our scheme are the resource-efficient second-order TBM schemes from [RDP08, VV20]. Our goal is to securely compute $y = \mathrm{S}(x)$, where

$$\mathrm{S} : \{0,1\}^n \to \{0,1\}^m,$$

is stored in the form of a lookup table, and the input x is in the secret-shared form

$$x = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \ \in \{0,1\}^n.$$

Needless to say, a naïve extension of the second-order scheme from [VV20] (a variant of [RDP08] that supports pre-processing) to the third order scenario will be insecure, as demonstrated in Appendix A. Also, the third-order attack from [CPR07] on the HO-TBM scheme from [SP06] is now well-known. To prevent these sorts of third-order attacks, we opted for the shift of the table by x_1 in addition to the shift by a combination of two shares as done in [RDP08, VV20]. This brings down the total number of shifts of the randomised table from 3 in [Cor14, CRZ18, VV21] to 2. More importantly, we avoid any explicit mask refresh used in the current HO-TBM schemes. But, a mere construction of the randomised table in two shifts *cannot* make the scheme secure since it leads to an attack as described in Appendix B. Hence, we opt to use a vector of random masks to protect the randomised table in the second shift. Moreover, the order of random masks has to be chosen with caution, otherwise, it will lead to a tuple of intermediate variables whose simulation demands the balanced S-box property (see Section 2.2 (Para 1) and Appendix C).

This approach of constructing the randomised table in two shifts with a sufficient amount of random values will only result in a 3-NI secure scheme (see Remark 1). One way to achieve a 3-SNI secure instantiation that assures *composability* is to refresh the outputs of the 3-NI scheme using a 3-SNI *mask refresh* [BBD+16, Proposition 4, Section 5]. A natural choice for the 3-SNI *mask refresh* from the literature is the *full refresh* algorithm instantiated at third-order from [BBD+16]. This procedure is nothing but multiplying the secret with *one* using the ISW multiplication over $\mathbb{F}_{2^n}$ [ISW03]. Since the computation and randomness complexity of this algorithm is quadratic in the number of shares, there will be an additional overhead on the online execution time of the resulting scheme. To reduce the overhead associated with the SNI *mask refresh*, we make use of the *RefreshBlock* gadget by Barthe et al. [BDF+17] which was later proven to be SNI secure in [BBD+20]. We have instantiated their scheme at third order and call it 3-RB. For convenience, we slightly modified their notation and presented their scheme in Algorithm 2. Concretely, the amount of randomness and computation time for 3-RB and 3-SNI *full refresh* are 4 random values and 8 xors vs. 6 random values and 12 xors, respectively.

Overall, our scheme consists of two sub-procedures. The first procedure is presented in Algorithm 1, and Algorithm 2 describes the second procedure. Algorithm 1 begins with an *offline* phase that deals with the computation of the randomised lookup table. This phase consists of constructing an auxiliary table T_{aux} to hold the result of the shift by x_1, whose entries are shifted further with x_2 and x_3 in one step to build the final table T. The offline phase is followed by an *online* phase, where a lookup of the table T using x_4 results in the secret sharing of $\mathrm{S}(x)$. Continue the *online* phase further in Algorithm 2 by receiving the outputs from Algorithm 1 and mask refresh them using Algorithm 2 to generate the final output sharing of $\mathrm{S}(x)$. We would like to stress that this final step of

mask refresh is crucial to prove the 3-SNI security of our scheme. We summarise the steps of our proposed third-order lookup table scheme in Algorithm 3. We have also explicitly marked the offline and online computation phases in the description of the methods. Note that for the table indexing and access, we have used the parenthesis notation $\big($like $T(a)\big)$, and for the vector Y access we have used the [ ] notation (like $Y[i]$). Table 1 provides estimates for the number of bit operations, RAM memory and randomness usage *per* (n, m) S-box for Algorithm 3.

Remark 1. The scheme presented cannot be proved 3-SNI due to the following: suppose that if the probed triplet is $\big((x_2 \oplus \upsilon \oplus x_3), \mathrm{S}(x_1 \oplus \upsilon \oplus a) \oplus y_1, y_1\big)$. This triplet requires the knowledge of three input shares for the simulation but as per the definition of SNI, we are allowed to use only two shares since there are only two intermediate variables in this triplet and the other probed variable is an output share.

Algorithm 1: 3-NI randomised LUT scheme.

Input :

- Input shares x_1, x_2, x_3, x_4, such that $x = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.
- An (n, m) S-box lookup table S.

Output : Output shares: y_1, y_2, y_3, y_4, such that $\mathrm{S}(x) = y_1 \oplus y_2 \oplus y_3 \oplus y_4$.

`// Pre-processing (Offline phase)`

$y_1 \xleftarrow{\$} \{0,1\}^m$
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $T_{\text{aux}}(a) \leftarrow \mathrm{S}(x_1 \oplus a) \oplus y_1$
end
$\upsilon \xleftarrow{\$} \{0,1\}^n$
$y_2 \xleftarrow{\$} \{0,1\}^m$
$d \longleftarrow (x_2 \oplus \upsilon) \oplus x_3$
for $i \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $Y[i] \xleftarrow{\$} \{0,1\}^m$
end
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $b \longleftarrow a \oplus d$
  $T(b) \longleftarrow (T_{\text{aux}}(a \oplus \upsilon) \oplus Y[b]) \oplus y_2$
end

`// Post-processing (Online phase)`

$y_3 = Y[x_4]$
$y_4 = T(x_4)$

2.1 Correctness

The following equations provide the proof of correctness of Algorithm 1.

$$\begin{aligned}
T(x_4) &= T_{\text{aux}}(\upsilon \oplus x_4 \oplus d) \oplus Y[x_4] \oplus y_2 \\
&= T_{\text{aux}}(\upsilon \oplus x_4 \oplus x_2 \oplus \upsilon \oplus x_3) \oplus y_3 \oplus y_2 && \because y_3 = Y[x_4] \\
&= T_{\text{aux}}(x_4 \oplus x_2 \oplus x_3) \oplus y_3 \oplus y_2 \\
&= \mathrm{S}(x_1 \oplus x_2 \oplus x_3 \oplus x_4) \oplus y_1 \oplus y_3 \oplus y_2 && \because T_{\text{aux}}(a) = \mathrm{S}(x_1 \oplus a) \oplus y_1 \\
&= \mathrm{S}(x) \oplus y_1 \oplus y_2 \oplus y_3.
\end{aligned}$$

Algorithm 2: 3-RB [BDF+17, BBD+20]

Input : $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \{0,1\}^m$.
Output : $\beta_1, \beta_2, \beta_3, \beta_4$ such that $\beta_1 \oplus \beta_2 \oplus \beta_3 \oplus \beta_4 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4$.

// Post-processing (Online phase)

1 $r_1, r_2, r_3, r_4 \xleftarrow{\$} \{0,1\}^m$
2 $\beta_1 \leftarrow (\alpha_1 \oplus r_1) \oplus r_4$
3 $\beta_2 \leftarrow (\alpha_2 \oplus r_2) \oplus r_1$
4 $\beta_3 \leftarrow (\alpha_3 \oplus r_3) \oplus r_2$
5 $\beta_4 \leftarrow (\alpha_4 \oplus r_4) \oplus r_3$
6 return $\beta_1, \beta_2, \beta_3, \beta_4$.

Algorithm 3: Third-order secure masked S-box computation with *pre-processing*

Input : $x_1, x_2, x_3, x_4 \in \{0,1\}^n$.
Output : $y_1, y_2, y_3, y_4 \in \{0,1\}^m$ such that $y_1 \oplus y_2 \oplus y_3 \oplus y_4 = \mathrm{S}(x)$.

1 Process the input shares of x using Algorithm 1 to obtain the shares of $\mathrm{S}(x)$
2 Refresh the shares of $\mathrm{S}(x)$ using Algorithm 2

2.2 Security Proof

Before proceeding with the security proof, we will first recollect the $t-$NI and $t-$SNI security notions from [BBD+16].

Definition 1. **t-Non-Interference (t-NI)** [BBD+16]. Let G be a gadget that takes secret input shares $(x_1, x_2, \ldots, x_k)$ and let the output shares be $(y_1, y_2, \ldots, y_k)$, where $y_1 \oplus y_2 \oplus \cdots \oplus y_k = y = f(x)$. Let the adversary observe t_I many input and intermediate shares, and t_O output shares such that $t_I + t_O \leq t$. Then, G is said to be t-NI secure if the set of t observations can be perfectly simulated using $t_I + t_O$ many input shares of x.

Definition 2. **t-Strong Non-Interference (t-SNI)** [BBD+16]. Let G be a gadget that takes $(x_1, x_2, \ldots, x_k)$ as the input shares of $x = x_1 \oplus x_2 \oplus \cdots \oplus x_k$ and let the output shares be $(y_1, y_2, \ldots, y_k)$, where $y_1 \oplus y_2 \oplus \cdots \oplus y_k = y = f(x)$. Let the adversary observe t_I many input and intermediate shares, and t_O many output shares such that $t_I + t_O \leq t$. Then, G is said to be t-SNI secure if the set of t observations can be simulated using only t_I input shares of x, where $t_I \leq t$.

From the given definitions, it can be observed that the bound on the number of input shares in the context of SNI simulation depends only on the observations of the input/intermediate variables vs. the total number of observations in NI notion. We first prove Lemma 1, showing that the Algorithm 1 (using shift and lookup using final share, x_4) is 3-NI. The output shares from Algorithm 1 are refreshed using 3-RB (Algorithm 2). Finally, we will conclude by showing that the composed construction presented in Algorithm 3 is 3-SNI. We would like to stress that, unlike the second-order [RDP08] scheme, our scheme *does not require the S-box balancedness property* for the simulation. To recall, an (n, m) S-box S, $m \leq n$, is balanced if every output word is the image under S of exactly 2^{n-m} input words.

We now proceed with the security proof of Algorithm 1. The intuition behind the security proof presented in Lemma 1 is as follows. Since the offline phase in this algorithm uses three input shares x_1, x_2, x_3 to build the randomised lookup table, it is trivial to see that the simulation of any 3-tuple of variables from the offline phase is possible using at most three input shares. This leaves us with the online phase where the final table lookup

Table 2: List of variables in Algorithm 1.

Inputs	
I	x_1, x_2, x_3, x_4
Outputs	
II	$y_1, y_2, y_3 = Y[x_4]$ and $y_4 = T(x_4) = \mathrm{S}(x) \oplus y_1 \oplus y_2 \oplus y_3$
Remaining variables	
j	I_j, $0 \leq a < 2^n$
1	$(x_1 \oplus a)$
2	$\mathrm{S}(x_1 \oplus a)$
3	$T_{\mathrm{aux}}(a) = \mathrm{S}(x_1 \oplus a) \oplus y_1$
4	random value: υ, $\upsilon \oplus a$
5	$(x_2 \oplus \upsilon)$
6	$d = (x_2 \oplus \upsilon) \oplus x_3$
7	$b = a \oplus d = a \oplus \big((x_2 \oplus \upsilon) \oplus x_3\big)$
8	random masks: $Y[b]$ where $b = a \oplus d$
9	$T_{\mathrm{aux}}(\upsilon \oplus a) = \mathrm{S}(x_1 \oplus \upsilon \oplus a) \oplus y_1$
10	$T_{\mathrm{aux}}(\upsilon \oplus a) \oplus Y[b] = \mathrm{S}(x_1 \oplus \upsilon \oplus a) \oplus y_1 \oplus Y[b]$
11	$T(b) = \big(T_{\mathrm{aux}}(\upsilon \oplus a) \oplus Y[b]\big) \oplus r_2 = \big(\mathrm{S}(x_1 \oplus \upsilon \oplus a) \oplus y_1 \oplus Y[b]\big) \oplus y_2$

x_4 outputs $y_4 = T(x_4)$. Since x_4 and y_4 are not combined with any other variables of Algorithm 1, the adversary has to probe either x_4 or y_4 individually to obtain x. With the remaining two probes, he can observe at most two input/intermediate variables. Hence, the task is two prove that the simulation of the observed variables together from the offline and the online phases depends on at most three input shares. For ease of reference, we list all the input, output, and intermediate variables of Algorithm 1 in Table 2.

Lemma 1. *Algorithm 1 is 3-NI secure.*

Proof. The gadget here is the S-box S that takes as input x in the form of four input shares $x_1, x_2, x_3, x_4 = x \oplus x_1 \oplus x_2 \oplus x_3$, and outputs shares $y_1, y_2, y_3, y_4 = \mathrm{S}(x) \oplus y_1 \oplus y_2 \oplus y_3$. To demonstrate that the construction is 3-NI, we need to prove that the simulation of any three variables of Algorithm 1 (as listed in Table 2) requires the knowledge of a maximum of three input shares. Let I be the set of probes the adversary chooses to observe in the gadget. We construct an index set J that holds the set of input share indices required for simulating the observed probes from I. The goal is to show that $|J| \leq 3$.

1. Initialise $J = \phi$.
2. Probing x_i or y_i (except y_3) results in $J = J \cup \{i\}$. The output variable y_3 is an exception because it depends on x_4 as $y_3 = Y[x_4]$. Hence, update the index set as $J = J \cup \{4\}$ when y_3 is probed. This covers the inputs and outputs of the gadget.
3. Add 1 to J as $J = J \cup \{1\}$ when any variable from $\{I_1, I_2, I_3\}$ is probed.
4. When *a pair of variables* from the subset $\{I_4, \ldots, I_{11}\}$ are probed, then update $J = J \cup \{2, 3\}$. Note that the above pair of variables have the random variable υ in

common. Even though I_6 is a random mask chosen independent of the secret, the index of this mask still depends on v. Note that this case covers probing the same variable at distinct values of index a, $0 \leq a < 2^n$.

It can be observed from the above index set construction that we add at most one input share index per probed variable, thus $|J| \leq 3$. Note that the indices $\{2, 3\}$ are added only by probing a pair of variables. Now we are going to discuss the simulation of the set of probed variables I using the input shares $x_{|J}$.

1. It is trivial to simulate any of the probed output share(s) $y_1, y_2,$ or y_3 by assigning them uniform and independent random value(s) since the same would have happened in the actual implementation. If the final output share y_4 is probed, we can still assign a uniform random value since there always exists an unprobed output mask $y_i, i \neq 4$, that randomises y_4.

2. Any probed random variable like I_4 and I_8 can be simulated with a random value as this would have happened in the actual implementation. Recollect that $I_8 = Y[a \oplus d]$, where $d = (x_2 \oplus v) \oplus x_3$ and a being the loop counter. But the case of the variable $Y[b]_{\text{ at } a=c}$, requires careful attention when probed along with y_3. It is possible that y_3 and $Y[b]$ maybe same or distinct. The simulation in this case happens using the input share x_4. Recall that probing y_3 resulted in $J = J \cup \{4\}$. If v remains unprobed, then sample a random value for d and compute $b = c \oplus d$, else compute d using the input shares x_2 and x_3. To simulate the probed variable, compare the value of b with x_4. If $b = x_4$, then generate a random value and return the same value for both y_3 and $Y[b]$, else return two independent uniform random values.

3. Needless to say, any intermediate variable depending on at most one input share (including constants, sampled randomness, and input shares) is straightforward to simulate. The intermediate variables $\{I_1, \ldots, I_5\}$ fall under this category which also covers the variables in the construction of T_{aux}.

4. The simulation of the intermediates I_6 or I_7 is as follows: if any other variable involving v i.e, from the subset $\{I_4, \ldots, I_{11}\}$, is not probed, then assign a random value. Otherwise, compute the probed variable using x_2 and x_3.

5. Similarly, depending on whether the output mask y_1 is probed or not, simulate I_9 either with a randomly chosen value or calculate the observed value using x_1.

6. The simulation of the variables I_{10} or I_{11} that appear in the construction of T that involve more than one input shares is as follows:

   (a) *at least one unprobed random mask:* if either the output share y_1 or the random mask $Y[d]$ remains *unprobed*, the simulation does not require the knowledge of any input share since the unprobed value acts as a one-time pad.

   (b) Otherwise, this would belong to Case 4 described above. Then we would have $1 \in J$ due to y_1 and $2, 3 \in J$ due to probing the pair $\big(I_{10}$ (or $I_{11}), Y[d]\big)$. So, compute the observed variables using input shares x_1, x_2, x_3.

   (c) Even the pair $(I_{11 \text{ at } a=c_1}, I_{11 \text{ at } a=c_2})$, c_1, c_2 being constants (or I_{10} at two distinct indices), can be assigned values, thanks to the vector of random values, Y.

   (d) The complex case in this setting would be probing (y_3, y_4) along with either I_{10} or I_{11}. Assigning random values to the output shares y_3 and y_4 would fix the value of $y_1 \oplus y_2$ since

$$y_3 \oplus y_4 = \mathrm{S}(x) \oplus y_1 \oplus y_2.$$

So, we cannot use the fact that neither y_1 nor y_2 are unprobed. This is where we carefully design the scheme such that the index of Y (in I_{10} or I_{11}) is randomised with the help of b. So, sample b at random, thanks to the unprobed v. We would have added 4 to J due to the probed y_4. Depending on whether the sampled d equals x_4, assign $I_{11} = y_4$. Similarly, for the case of I_{10} being probed, we can assign $I_{10} = y_4 \oplus y_3$. If the index $b \neq x_4$, the table entry can be assigned a uniform random value due to unprobed $Y[b]$. In [RDP08], this case calls for the S-box to be balanced. But, we do not require the balanced S-box property for the simulation in our scheme (see Remark 2).

Thus, we can conclude that any triple consisting can be simulated with the knowledge of at most three input shares. □

Remark 2. As explained in the security proof of the second-order scheme with pre-processing from [VV20, Theorem 1], simulating the pair $\big(y_3 = \mathrm{S}(x) \oplus y_1 \oplus y_2,\ \mathrm{S}(v \oplus a) \oplus y_1 \oplus y_2\big)$ in their 2-SNI security proof requires the S-box to be *balanced*. This is because the output masks y_1 and y_2 are reused for the entire table and there is no additional random mask left. So, the variable $\mathrm{S}(v \oplus a)$ can only be assigned a random value provided the S-box, S is balanced. But, in our case, thanks to the vector of randomness Y, the tuple can be simulated in Step 2(d) for any S-box.

Theorem 1. *Algorithm 3 is 3-SNI secure.*

Proof. Because Algorithm 1 is 3-NI and Algorithm 2 is 3-SNI, their composition is known to be 3-SNI [BBD+16, Proposition 4, Section 5]. Since the detailed proof is not available in [BBD+16], for the sake of completeness we are giving a formal proof below.

Let G_1 and G_2 be the gadgets corresponding to Algorithm 1 and Algorithm 2, respectively. To prove that Algorithm 3 is 3-SNI, we need to show that the gadget G obtained by the composition of the sub gadgets G_1 and G_2 is indeed 3-SNI. The graphical representation of the composition of gadgets is presented in Figure 1.
Let the number of input and intermediate probes on the gadget G be t_I and t_O such that

$$t_I + t_O \leq t = 3.$$

Let t_{I_1} and t_{O_1} be the number of probes on input/intermediate and output variables of the sub gadget G_1, respectively. Similarly, let t_{I_2} and t_{O_2} be the number of probes on the gadget G_2 such that

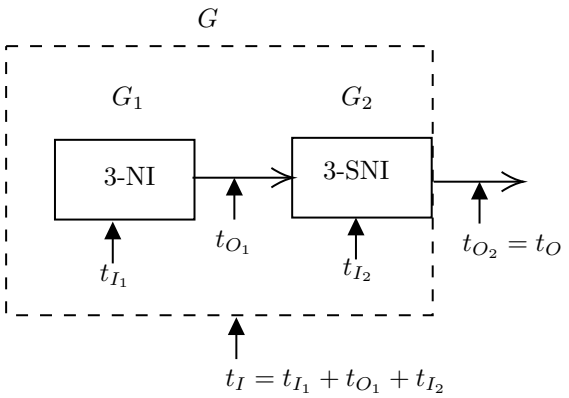

Figure 1: Our scheme resulting from the composition of gadgets 3-NI G_1 and 3-SNI G_2.

$$t_{I_1} + t_{O_1} + t_{I_2} = t_I, \tag{1}$$

and

$$t_{O_2} = t_O.$$

Let J, J_1 and J_2 be the set of input shares required for the simulation of the gadgets G, G_1 and G_2, respectively. Since the gadget G_1 is 3-NI (from Lemma 1),

$$|J_1| \leq t_{I_1} + t_{O_1}, \tag{2}$$

whereas for the 3-SNI gadget G_2, we have

$$|J_2| \leq t_{I_2}. \tag{3}$$

Note that simulating the observations in the gadget G is nothing but simulating the observations in the sub gadgets G_1 and G_2, so

$$\begin{aligned} J &= J_1 \cup J_2 \\ |J| &\leq |J_1| + |J_2| \\ &\leq t_{I_1} + t_{O_1} + t_{I_2} \qquad \because (2) \text{ and } (3) \\ &\leq t_I. \qquad \because (1) \end{aligned}$$

This shows that the set of input shares required for the simulation of the gadget G is bounded by t_I i.e., the number of probes on input/intermediate shares. Hence, we can conclude that the gadget G is 3-SNI.

□

3 Implementation

This section presents the implementation details of our scheme presented in Algorithm 3. We provide a detailed comparison of our work with the state-of-the-art masking schemes instantiated at third order. The schemes are compared in terms of the overall RAM memory, computation time, and the number of TRNG calls. Our approach achieves an improved overall execution time compared to the higher-order scheme of [CRZ18] (at third order) while maintaining the online time. The latter is achieved by making use of the full pre-processing advantage of lookup table-based masking schemes. We ran our third-order scheme for the block ciphers AES-128 and PRESENT (80-bit key variant). The source code for our third-order scheme implementation is available at [AV].

The target embedded device is NXP's FRDM-K64F which possesses a RAM memory of 256 KB, 1MB flash memory, and has a processor clock speed of 120 MHz. The device comes with an in-built hardware random-number generator clocked at 48 MHz. The RNGA module requires approximately 300 clock cycles to generate a 32-bit random word. We compile our implementations using the $-O1$ flag. Even though this setting would increase the overall clock cycle consumption when compared to other flags, further compiler optimisations may impact the side-channel security of the implementation, as reported in [BWG+22]. Our implementation includes the third-order masked full block cipher implementation of AES-128 [FIP01] and 80-bit key PRESENT [BKL+07]. While implementing the block cipher AES-128, we have used the publicly available code from [Cor] and [VV]. For the PRESENT cipher, we referred the unmasked implementation from the public repository [Klo]. The code size is 26.5 KB for the masked implementation of AES-128 using Algorithm 3, whereas for the full cipher masked implementation of PRESENT, the code size is 25.8 KB. We would like to stress that our implementation code is only

for the purpose of benchmarking and it requires additional hardening countermeasures to resist against real-world side-channel attacks [BWG+22].

Since the computation in our schemes is divided into two phases, offline (pre-processing) and online (post-processing), the total number of clock cycles for the execution is the sum of offline and online computations. By offline computation we mean the total number of clock cycles used for the processing that is independent of the input secret. By online computation, we refer to the computation that is performed after the availability of the secret input. The total RAM memory includes the amount of space required for the pre-processed randomised lookup tables and the associated variables. This amounts to the pre-processing of the lookup tables of 160 (10 rounds $\times$ 16) and 496 (31 round $\times$ 16) S-box invocations for AES-128 and PRESENT ciphers, respectively. The true random values and the input seed to PRGs are generated using the in-built RNGA module.

Table 3 on Page 12 presents the comparison of our implementation results for the 3-SNI secure implementation of AES-128 with other 3-SNI lookup table-based schemes [CRZ18, VV21]. To maximise the speed of computation, multiple entries of the randomised table can be *packed* and processed in parallel using the large register variant (LRV) approach presented in [CRZ18]. It can be observed from Table 3 that there is an approximate factor *two* reduction in the RAM memory usage of Algorithm 3 when compared to the *increasing shares* variant of [CRZ18]. Where as there is a 78.87% reduction in the overall running time of the proposed scheme presented in Algorithm 3 when compared to the LRV variant with increasing shares approach of [CRZ18]. One may be tempted to adopt a similar LRV strategy to Algorithm 3 to improve the overall running time of our scheme. But it turns out that packing the entries using LRV variant for our generic scheme is insecure and the details can be found in Appendix D.

We present the implementation results for the circuit-based schemes [RP10, GR17, WGY+22] in Table 4 on Page 12. A recent work of Wang et al. [WGY+22] demonstrated a scheme that facilitates *reusing* $t = k - 1$ shares across the ISW multiplication gadgets. Similar to the strategy followed in the recent LUT-based schemes [VV20, VV21], the authors of [WGY+22] have divided the computation into *offline* and *online* phases. They have provided their source code optimised in C and *assembly* for a *single round* of masked AES-128 [Wan]. Also, their implementation pre-processes the linear layers. Since their code is *not* a full AES-128 cipher implementation, we could only *estimate* the clock cycles needed for a single complete execution of AES-128 instantiated at third order. Our online execution time (*without* the pre-processing of the linear layers) is comparable to that of Wang et al.'s scheme [WGY+22] with *full* pre-processing (including the linear layers) (see Row 1 of Table 3 and Row 3 of Table 4). The offline time of [WGY+22] is much faster than ours, and so is the comparison w.r.t. the randomness usage. We think that it would be an interesting research direction to explore whether the share resue technique of [WGY+22] can be adapted to the table-based masking schemes, in particular, to our scheme.

In Table 4, we compare the online execution time of our scheme with that of bitslicing and [RP10]. It can be observed that our scheme (Algorithm 3) requires *eight* times lesser clock cycles when compared to the 3-SNI instantiation of masked AES-128 implementation using bitslicing (optimised with 32-bit ISW-AND followed by mask refresh using *full refresh* [DDF14]). Where as the online time of Rivain and Prouff's third-order instantiation [RP10] is *33.5* times *slower* than our proposed third-order scheme. We also implemented the lightweight cipher PRESENT using the proposed scheme. The interesting observation here is that even though the overall execution time of the third-order masked PRESENT cipher is less than that of AES-128, the online time for the lightweight cipher is *higher* than the latter.

Table 3: Comparison of third-order masked implementation of AES-128 using our scheme (Algorithm 3) vs. [CRZ18]. Also, the results for third-order masked implementation of PRESENT using Algorithm 3. Total memory and true random values are in KB and the clock cycles are represented in millions (M).

Variant	Total Memory	Offline(M)	Online(M)	Total(M)	# True random
AES-128 Implementation					
Algorithm 3	87.22	3.80	0.10	3.90	40.62
[CRZ18] (Increasing shares)	167.1	36.19	0.10	36.29	240.46
[CRZ18] (Increasing shares with LRV)	167.1	17.94	0.76	18.7	240.46
PRESENT Implementation					
Algorithm 3	23.44	1.01	0.46	1.47	9.68

Table 4: Comparison of third-order circuit based masked implementation of AES-128. Total memory and true random values are in KB and the clock cycles are represented in millions (M).

Variant	Total Memory	Offline(M)	Online(M)	Total(M)	# True random
AES-128 Implementation					
[RP10]	7.9	-	0.96	0.96	5.62
[GR17] (Bitslicing)	9.98	-	0.84	0.84	3.75
[WGY+22] (Common shares)	5.92	1.02	0.09	1.11	0.21

4 Conclusion

In this paper, we proposed a third-order secure table-based masking scheme that significantly outperforms the state-of-the-art table-based masked software implementations in terms of time, RAM memory, and randomness usage. A future research direction on this topic is to design table-based schemes at fourth and higher orders that are significantly more efficient than the fourth-order and slightly higher-order instantiations of current higher-order table-based masking schemes. While nearly eliminating the *maskrefresh* in our proposals leads to efficient schemes, the proofs become more involved. As we saw, there is a fine line separating security and efficiency, and one needs to be vigilant of the same. The lack of formal verification tools that can directly verify the security of table-based masking schemes is making the effort to design new schemes more involved. It would be a beneficial contribution to develop such verification tools.

Acknowledgements

We would like to thank the anonymous reviewers of TCHES 2023 for their valuable comments which helped us improve this manuscript. This work was partly funded by the Infosys Foundation Career Development Chair Professorship grant for Srinivas Vivek and by the Centre for Internet of Ethical Things (CIET), IIIT Bangalore.

References

[AV] Anju S Alexander and Annapurna Valiveti. C Third-Order LUT-based AES and PRESENT Implementation. Available at https://github.com/annapurna-pvs/Higher-Order-LUT-PRG. Last accessed on October 7, 2022.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BBD+20] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *J. Cryptogr. Eng.*, 10(1):17–26, 2020.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BWG+22] Arthur Beckers, Lennert Wouters, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Provable secure software masking in the real-world. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*, volume 13211 of *Lecture Notes in Computer Science*, pages 215–235. Springer, 2022.

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-Order Masking Schemes for S-Boxes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, 2012.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In Gierlichs and Poschmann [GP16], pages 498–514.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Wiener [Wie99], pages 398–412.

[Cor] Jean-Sébastian Coron. Higher-order countermeasures for AES and DES. Available at `https://github.com/coron/htable`. Last accessed on October 1, 2022.

[Cor14] Jean-Sébastien Coron. Higher Order Masking of Look-Up Tables. In Nguyen and Oswald [NO14], pages 441–458.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast Evaluation of Polynomials over Binary Finite Fields and Application to Side-Channel Countermeasures. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014. Proc.*, volume 8731 of *LNCS*, pages 170–187. Springer, 2014.

[CRZ18] Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High Order Masking of Look-up Tables with Common Shares. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):40–72, 2018.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In Nguyen and Oswald [NO14], pages 423–440.

[FIP01] NIST FIPS. Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/NIST, November 26, 2001. Available from the NIST website, 2001.

[GP16] Benedikt Gierlichs and Axel Y. Poschmann, editors. *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*. Springer, 2016.

[GR16] Dahmun Goudarzi and Matthieu Rivain. On the Multiplicative Complexity of Boolean Functions and Bitsliced Higher-Order Masking. In Gierlichs and Poschmann [GP16], pages 457–478.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How Fast Can Higher-Order Masking Be in Software? In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

[GRVV17] Dahmun Goudarzi, Matthieu Rivain, Damien Vergnaud, and Srinivas Vivek. Generalized Polynomial Decomposition for S-boxes with Application to Side-Channel Countermeasures. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 154–171. Springer, 2017.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Wiener [Wie99], pages 388–397.

[Klo] D. Klose. C PRESENT Implementation. Available at http://www.lightweightcrypto.org/implementations.php. Last accessed on October 1, 2022.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO 1996, Proc.*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[NO14] Phong Q. Nguyen and Elisabeth Oswald, editors. *EUROCRYPT 2014. Proc.*, volume 8441 of *LNCS*. Springer, 2014.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013. Proc.*, volume 7881 of *LNCS*, pages 142–159. Springer, 2013.

[PV16] Jürgen Pulkus and Srinivas Vivek. Reducing the Number of Non-linear Multiplications in Masking Schemes. In Gierlichs and Poschmann [GP16], pages 479–497.

[RDP08] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2008.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010. Proc.*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[RV13] Arnab Roy and Srinivas Vivek. Analysis and Improvement of the Generic Higher-Order Masking Scheme of FSE 2012. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013. Proc.*, volume 8086 of *LNCS*, pages 417–434. Springer, 2013.

[SP06] Kai Schramm and Christof Paar. Higher Order Masking of the AES. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.

[Vad17] Praveen Kumar Vadnala. Time-Memory Trade-Offs for Side-Channel Resistant Implementations of Block Ciphers. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2017.

[VV] Annapurna Valiveti and Srinivas Vivek. Implementation of Higher-order Lookup Table using PRG. Available at https://github.com/annapurna-pvs/Higher-Order-LUT-PRG. Last accessed on October 1, 2022.

[VV20] Annapurna Valiveti and Srinivas Vivek. Second-order masked lookup table compression scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):129–153, 2020.

[VV21] Annapurna Valiveti and Srinivas Vivek. Higher-order lookup table masking in essentially constant memory. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):546–586, 2021.

[Wan] Weijia Wang. Software Implementation of Masked AES round function with common shares. Available at https://github.com/wjwangcrypto/MaskingWithCommomShares.git. Last accessed on October 1, 2022.

[WGY+22] Weijia Wang, Chun Guo, Yu Yu, Fanjie Ji, and Yang Su. Side-channel masking with common shares. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):290–329, 2022.

[Wie99] Michael J. Wiener, editor. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.

A Trivial Extension of [RDP08]

Algorithm 4: Trivial extension of [RDP08] variant from [VV20].

Input : Input shares x_1, x_2, x_3, x_4 such that $x = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.
Output : Output shares: y_1, y_2, y_3, y_4, such that $\mathrm{S}(x) = y_1 \oplus y_2 \oplus y_3 \oplus y_4$.

$\upsilon \xleftarrow{\$} \{0,1\}^n$
$y_1, y_2, y_3 \xleftarrow{\$} \{0,1\}^m$
$d \longleftarrow (x_2 \oplus \upsilon) \oplus x_3$
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $b \longleftarrow a \oplus d$
  $T(b) \longleftarrow \big((\mathrm{S}(x_1 \oplus \upsilon \oplus a) \oplus y_1) \oplus y_2\big) \oplus y_3$
end
$y_4 = T(x_4)$

The algorithm presented in Algorithm 4 on Page 16 which is a trivial extension of the second-order scheme of [VV20] (a [RDP08] variant that supports pre-processing) to third-order is insecure due to the attack tuple $\big(x_1 \oplus \upsilon, (x_2 \oplus \upsilon) \oplus x_3, x_4\big)$ since the random value υ *can not be reused* to combine input shares.

B Constructing the Randomised Table in Two Shifts

To avoid the straightforward attack discussed in Appendix A, one can construct the randomised table T in two steps. Construct a T_{aux} in Step 1 by shifting it with x_1 and protect the shift by the output mask y_1. In Step 2, shift T_{aux} using $(x_2 \oplus \upsilon) \oplus x_3$ followed by y_2 and y_3. The steps are summarised in Algorithm 5 on Page 16.

Algorithm 5: The scheme with two shifts and three output masks.

Input : Input shares x_1, x_2, x_3, x_4 such that $x = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.
Output : Output shares: y_1, y_2, y_3, y_4, such that $\mathrm{S}(x) = y_1 \oplus y_2 \oplus y_3 \oplus y_4$.

$y_1, y_2, y_3 \xleftarrow{\$} \{0,1\}^m$
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $T_{\text{aux}}(a) \leftarrow \mathrm{S}(x_1 \oplus a) \oplus y_1$
end
$\upsilon \xleftarrow{\$} \{0,1\}^n$
$d \longleftarrow (x_2 \oplus \upsilon) \oplus x_3$
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $b \longleftarrow a \oplus d$
  $T(b) \longleftarrow \big(T_{\text{aux}}(a \oplus \upsilon) \oplus y_2\big) \oplus y_3$
end
$y_4 = T(x_4)$

This attack is possible due to the fact that the output mask combination is common across T. This scheme can be attacked using the tuple $\big(y_4, T(b)_{a=c}, \upsilon\big)$, where c is a constant,

$$\begin{aligned} & y_4 \oplus T(b)_{a=c} \\ &= \big(\mathrm{S}(x) \oplus y_1 \oplus y_2 \oplus y_3\big) \oplus \big(\mathrm{S}(x_1 \oplus \upsilon \oplus c) \oplus y_1 \oplus y_2 \oplus y_3\big) \\ &= \mathrm{S}(x) \oplus \mathrm{S}(x_1 \oplus \upsilon \oplus c), \end{aligned}$$

this value combined with υ together depends on x.

C Scheme which works only for Balanced S-boxes

One natural way to thwart the attack described in Appendix B is to to use a vector of random masks to protect T since repeating the same set of output masks across the table results in an insecure scheme. This idea is presented in Algorithm 6 on Page 18.

The order of the masks has to be carefully chosen since it may lead to a simulation that requires the *balanced* S-box property. Consider the tuple $y_3, y_4, (T_{\text{aux}}(a) \oplus y_2)_{a=c}$, c is a constant. Then,

$$\begin{aligned} & y_3 \oplus y_4 \oplus \big(T_{\text{aux}}(a) \oplus y_2\big)_{a=0}\big) \\ &= y_3 \oplus \big(\mathrm{S}(x) \oplus y_1 \oplus y_2 \oplus y_3\big) \oplus \big(\mathrm{S}(x_1 \oplus \upsilon) \oplus y_1 \oplus y_2\big) \\ &= \mathrm{S}(x) \oplus \mathrm{S}(x_1 \oplus \upsilon). \end{aligned}$$

One can simulate the above expression with a random value only when the S-box is assumed to balanced. The simulation fails, otherwise.

Algorithm 6: Instance of a scheme violating 3-SNI.

Input : Input shares x_1, x_2, x_3, x_4 such that $x = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.
Output : Output shares: y_1, y_2, y_3, y_4, such that $\mathrm{S}(x) = y_1 \oplus y_2 \oplus y_3 \oplus y_4$.

$y_1, y_2 \xleftarrow{\$} \{0,1\}^m$
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $T_{\mathrm{aux}}(a) \leftarrow \mathrm{S}(x_1 \oplus a) \oplus y_1$
end
$v \xleftarrow{\$} \{0,1\}^n$
$d \longleftarrow (x_2 \oplus v) \oplus x_3$
for $i \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $Y_3[i] \xleftarrow{\$} \{0,1\}^m$
end
for $a \leftarrow 0$ ***to*** $2^n - 1$ **do**
  $b \longleftarrow a \oplus d$
  $T(b) \longleftarrow (T_{\mathrm{aux}}(a) \oplus y_2) \oplus Y_3[b]$
end
$y_3 = Y_3[x_4]$
$y_4 = T(x_4)$

D Attack on Extension to Large Registers

Recall that the objective of the scheme described in Section 2 is to improve the overall running time (along with memory and randomness) compared to the state-of-the-art. In Algorithm 3, the computation deals with an m-bit value at a time. The target hardware can support, say w-bit computations, $w > m$. We follow a similar approach presented in second-order scheme of [RDP08] and extended to higher-order in [Cor14, CRZ18]. The idea is to *pack* multiple entries of the S-box prior to shifting the S-box with the input share(s) [Cor14, CRZ18]. It turns out that extending our scheme (Algorithm 3) to *large register* variant (LRV) leads to a second-order attack as explained below.

Let the processor support w-bit word instructions and w.l.o.g, say, $w = 2^i$ and let $m = 2^j$ which implies the register can hold $z = 2^{(i-j)}$ such m-bit values. Precisely, $w = z \cdot m$. Let each n-bit input x_i be viewed as

$$x_i = x_i^{(1)} || x_i^{(2)},$$

where, $x_i^{(1)} \in \{0,1\}^{\beta_1}$, $x_i^{(2)} \in \{0,1\}^{\beta_2}$ and $\beta_1 + \beta_2 = n$. Also, $\beta_2 = \log_2(z)$. Hence, every row of the randomised table is a pack of z S-box outputs and is represented as:

$$T(u) \longleftarrow \mathrm{S}(u \,||\, 0) || \ldots \mathrm{S}(u \,||\, (z-1)),$$

where $u \in \{0,1\}^{\beta_1}$. Algorithm 7 on Page 19 describes the offline computation using the LRV. It can be observed from Algorithm 7 (Step 5) that the output mask y_1' is formed by attaching the m-bit mask y_1, z times. The same is the case for y_2. This ensures all the rows of table are masked with the same y_1 and y_2 as in Algorithm 3.

During the online phase, a lookup of the table $T(x_4^{(1)})$ would result in a row using which construct a small table of size 2^z by unpacking the elements of the row. Also, corresponding output mask y_3' will be obtained by $Y_3'[x_4^{(1)}]$. One would shift this table with $x_i^{(2)}$, $i \in \{1,2,3,4\}$, to obtain the final sharing of $\mathrm{S}(x)$. But, there is a *second-order attack* on this variant by probing the pair $\left(T(x_4^{(1)}), Y_3'[x_4^{(1)}]\right)$. The attack works as follows.

Algorithm 7: Extension of Algorithm 3 to *large register* variant.

Input :

- Shares x_1, x_2, x_3. `// Shares for offline`
- w, the register size
- An (n, m) S-box lookup table S.

Output : Randomised lookup table T, y_1, y_2, Y_3'.

`// Pre-processing/Offline phase`

$z \longleftarrow w/m$
$\beta_2 \longleftarrow \log_2 z$
$\beta_1 \longleftarrow (n - \beta_2)$
$y_1 \xleftarrow{\$} \{0,1\}^m$
$y_1' = (y_1||\ldots||y_1)$, z times
for $a \leftarrow 0$ ***to*** $2^{\beta_1} - 1$ **do**
  $T(a) \longleftarrow \mathrm{S}(a||0)||\ldots||\mathrm{S}(a||(z-1))$
end
for $a \leftarrow 0$ ***to*** $2^{\beta_1} - 1$ **do**
  $T_{\text{aux}}(a) \leftarrow T(x_1^{(1)} \oplus a) \oplus y_1'$
end
$y_2 \xleftarrow{\$} \{0,1\}^m$
$y_2' = (y_2||\ldots||y_2)$, z times
for $i \leftarrow 0$ ***to*** $2^{\beta_1} - 1$ **do**
  $Y_3'[i] \xleftarrow{\$} \{0,1\}^w$
end
$\upsilon^{(1)} \xleftarrow{\$} \{0,1\}^{\beta_1}$
$d \longleftarrow (x_2^{(1)} \oplus \upsilon^{(1)}) \oplus x_3^{(1)}$
for $a \leftarrow 0$ ***to*** $2^{\beta_1} - 1$ **do**
  $b \longleftarrow a \oplus d$
  $T(b) \longleftarrow (T_{\text{aux}}(a \oplus \upsilon^{(1)}) \oplus Y_3'[b]) \oplus y_2'$
end

$$T(x_4^{(1)}) = \big(\mathrm{S}(x^{(1)}||0)||\mathrm{S}(x^{(1)}||1)||\ldots||\mathrm{S}(x^{(1)}||(z-1))\big) \oplus (y_1||\ldots||y_1) \oplus (y_2||\ldots||y_2) \oplus Y_3'[x_4^{(1)}].$$

$$T(x_4^{(1)}) \oplus Y_3'[x_4^{(1)}] = \big(\mathrm{S}(x^{(1)}||0)||\mathrm{S}(x^{(1)}||1)||\ldots||\mathrm{S}(x^{(1)}||(z-1))\big) \oplus (y_1||\ldots||y_1) \oplus (y_2||\ldots||y_2).$$

Since the output masks y_1 and y_2 are common across the entries that are packed together, shift the w-bit values and obtain the individual entries to cancel the effect of $y_1 \oplus y_2$. This leads to

$$\mathrm{S}(x^{(1)}||0)||\mathrm{S}(x^{(1)}||1)||\ldots||\mathrm{S}(x^{(1)}||(z-1)).$$

These individual values together depend on the β_1 most significant bits of the secret, x.