Higher-Order Lookup Table Masking in Essentially Constant Memory *

Annapurna Valiveti and Srinivas Vivek

IIIT Bangalore, IN
annapurna@iiitb.org,srinivas.vivek@iiitb.ac.in

Abstract. Masking using randomised lookup tables is a popular countermeasure for side-channel attacks, particularly at small masking orders. An advantage of this class of countermeasures for masking S-boxes compared to ISW-based masking is that it supports pre-processing and thus significantly reducing the amount of computation to be done after the unmasked inputs are available. Indeed, the "online" computation can be as fast as just a table lookup. But the size of the randomised lookup table increases linearly with the masking order, and hence the RAM memory required to store pre-processed tables becomes infeasible for higher masking orders. Hence demonstrating the feasibility of *full* pre-processing of higher-order lookup table-based masking schemes on resource-constrained devices has remained an open problem.

In this work, we solve the above problem by implementing a higher-order lookup table-based scheme using an amount of RAM memory that is essentially independent of the masking order. More concretely, we reduce the amount of RAM memory needed for the table-based scheme of Coron et al. (TCHES 2018) approximately by a *factor* equal to the number of shares. Our technique is based upon the use of pseudorandom number generator (PRG) to minimise the randomness complexity of ISW-based masking schemes proposed by Ishai et al. (ICALP 2013) and Coron et al. (Eurocrypt 2020). Hence we show that for lookup table-based masking schemes, the use of a PRG not only reduces the randomness complexity (now logarithmic in the size of the S-box) but also the memory complexity, and without any significant increase in the overall running time. We have implemented in software the higher-order table-based masking scheme of Coron et al. (TCHES 2018) at *tenth* order with full pre-processing of a single execution of all the AES S-boxes on a ARM Cortex-M4 device that has 256 KB RAM memory. Our technique requires only 41.2 KB of RAM memory, whereas the original scheme would have needed 440 KB. Moreover, our 8-bit implementation results demonstrate that the online execution time of our variant is about 1.5 times faster compared to the 8-bit bitsliced masked implementation of AES-128.

Keywords: Side-channel attacks · Masking · S-box · Probing leakage model · PRG · SNI security · IoT security · Software implementation.

1 Introduction

An IoT ecosystem helps several heterogeneous devices to collect, send and act on the data acquired from their environments. Even though IoT helps devices communicate over the Internet, the security and privacy of the data being stored and exchanged is a primary concern. Moreover, the physical availability of these devices make them vulnerable to side-channel attacks [Koc96, KJJ99]. Since the cryptographic mechanisms to protect secret data incur additional performance penalty, various trade-offs between processing time and

*Final version will be available at IACR-TCHES 2021.

memory required have to be considered while designing security protocols. Specifically, to enable reliable IoT applications using resource-constrained devices keeping in mind the limited memory and computation power is a challenging task.

In order to protect cryptographic implementations from side-channel attacks, various countermeasures have been proposed. Masking is a popularly used countermeasure where the secret variable, say $x \in \{0,1\}^k$, is split among n shares, where

$$n = t + 1,$$

and t is usually referred as the *masking order*. Any t among $t+1$ shares are assigned uniform random and independent values such that:

$$x = x_1 \oplus \cdots \oplus x_n. \tag{1}$$

Since the secret is split among multiple shares, naturally, it becomes harder to recover the secret from the side-channel observations. Precisely, the effort required to recover the secret grows *exponentially* with the masking order [CJRR99, PR13, DDF14a].

To secure block ciphers using the masking countermeasure, the operations of the block cipher have to be evaluated in the presence of shares. The operations are categorised into linear and non-linear operations. It is straightforward to execute linear operations over shares since the final sharing can be obtained by evaluating the operation on the individual shares. On the other hand, evaluating non-linear operations, particularly the S-boxes, in the presence of shares is a challenging task. There are various approaches presented to mask S-boxes efficiently. These approaches can be broadly classified into ISW-based [ISW03] or lookup table-based schemes [CJRR99, Cor14, CRZ18, GTP+20].

The polynomial-based and the circuit-based masking schemes belong to the former category [CGP+12, CRV15, CPRR15, GRVV17, JS17, GR17]. The computation time of circuit-based schemes depend on the number of non-linear multiplications required to implement an S-box. Usually, the cost required per multiplication is $O(t^2)$ and is the most expensive operation. However, the RAM memory required by these schemes is small and hence higher order masking can be implemented on resource constrained devices without any memory bottleneck. Since all the shares need to be present for every operation, pre-processing is *not* possible. By *pre-processing*, we refer to the computation that can be carried out independent of the unshared secret inputs. That is, the time required for computation that can be carried out before and after the availability of the actual secret inputs is referred to as *offline* and *online* times, respectively. Lookup table-based schemes are implemented by constructing a randomised lookup table in RAM. This randomised table typically can be fully pre-processed, thus substantially improving the online execution time as the final share is computed by a table lookup. Therefore, any S-box masking countermeasure has an overhead associated w.r.t computation time and/or RAM memory.

Lookup table-based masking schemes. The first provably secure first-order lookup table-based masking scheme was proposed by Chari et al. in [CJRR99]. According to this scheme, to securely evaluate a (k, k') S-box S, a randomised lookup table is constructed in RAM such that the S-box lookup table is shifted using x_1 and the shift is protected using an output mask, say, y_1. Formally,

$$T(u) = \mathrm{S}(u \oplus x_1) \oplus y_1, \quad \forall u \in \{0,1\}^k, \tag{2}$$

where $x_1 \in \{0,1\}^k$ and $y_1 \in \{0,1\}^{k'}$ are uniformly and independently chosen. It is evident from (2) that the memory required to store T is $k' \cdot 2^k$ bits. It can also be observed that the lookup table is constructed using x_1 alone and x_2 is used only in the final table lookup. Schramm and Paar [SP06], and Rivain, Dottax and Prouff [RDP08] suggested second-order lookup table masking schemes that have randomised lookup tables of size $2 \cdot k' \cdot 2^k$ and

$k' \cdot 2^k$ bits, respectively. Coron et al. [CRZ18] improved the lookup table-based scheme of [Cor14] and both these schemes require a RAM memory of $2 \cdot n \cdot k' \cdot 2^k$ bits, where, the number of shares $n = t + 1$ for the former scheme and $n = 2t + 1$ for the latter scheme, and t is the masking order. The schemes in [CRZ18] are proven t-SNI secure with the number of shares $n = t + 1$.

The idea of *compressing* a lookup table was first discussed by Rao et al. [RRST02] for the first-order lookup table masking scheme described above. The compressed randomised table size will be $k' \cdot 2^{k-1}$ bits which is a factor *two* reduction. Later, Vadnala [Vad17] generalised this idea for first and second orders. This approach offers lookup table size vs. execution time trade-offs based on a *compression parameter* l, where $1 \le l \le (n - 1)$. The size of the table is shown to be $\approx k' \cdot 2^{k-l} + (k - l) \cdot 2^l$ bits and $\approx k' \cdot 2^{k-l} + (k - l + 1) \cdot 2^l$ bits for first and second orders, respectively. Vivek [Viv17] showed a second-order attack on Vadnala's scheme. Recently, Valiveti and Vivek [VV20] proposed a second-order secure compression scheme that optimises memory, online execution time as well as randomness required for the implementation. Precisely, their scheme requires $k' \cdot (k - l + 1) + 2^l \cdot (k - l + k') + k' \cdot (2^{k-l} + 2^l)$ bits of RAM memory. An approach to reduce the size of lookup table at higher orders was recently proposed by Guo et al. in [GTP$^+$20]. This scheme is also secure against horizontal attacks. Even though this scheme reduces the size of the lookup table by a factor of *two*, it cannot take any advantage of pre-processing since the randomised lookup table construction is dependent on all input shares. Therefore, reducing the RAM memory required at higher orders for table-based masking schemes while still permitting full pre-processing has remained as an open problem.

Our Contribution. In this work, we solve the problem mentioned above of implementing a higher-order lookup table-based scheme using an amount of RAM memory that is essentially independent of the masking order but still facilitating full pre-preprocessing. Recall that the table-based masking schemes in [CRZ18] need $2 \cdot k' \cdot 2^k \cdot n$ bits of RAM memory per S-box, where n is the number of shares. We propose a variant scheme that needs only $(2 \cdot k' \cdot 2^k + k' \cdot k \cdot (n - 1)^2)$ bits of RAM memory and hence making the memory requirement to be essentially independent of the number of shares. Moreover, the randomness complexity is improved to $O(k' \cdot k \cdot n^3)$ compared to $O(k' \cdot 2^k \cdot n)$ for the original scheme. We achieve this reduction in RAM memory using the following observation: the masked lookup table of higher-order lookup table-based scheme [CRZ18] requires $k' \cdot 2^k \cdot n$ bits of RAM memory per each of the two tables that includes the *temporary* table. For each of these tables $k' \cdot 2^k \cdot (n - 1)$ bits of memory are actually used to hold randomly generated masks. For instance, to implement a *10-th* order secure 128-bit AES using the randomised lookup table scheme, we require $2 \cdot 10 \cdot 2^8 = 5120$ bytes of RAM memory per S-box to store random masks. Therefore, our idea is to generate these masks using a psuedorandom generator (PRG) as in [IKL$^+$13, CGZ20]. However, a straightforward use of PRG will not necessarily lead to a reduction in memory utilisation as we still need to store the n columns of the temporary tables. Our key idea is to use the *locality refresh* [IKL$^+$13, CGZ20] to restrict the input dependent tables values to a *single* column and store only this column. During the table lookup, we re-generate the random values corresponding to the remaining columns *on-the-fly*. The latter strategy was recently used in [VV20] in the context of second-order table compression. We would like to note that the techniques used in this work to achieve higher-order table compression is different from the one used in [VV20] for the second-order case. Next we briefly elaborate on the above mentioned technique.

The first step in our method is to replace the true random number generator (TRNG) of the higher-order masked lookup table scheme [CRZ18] with a PRG. Ishai et al. [IKL$^+$13] introduced the notion of *robust PRG* that can be used in place of a TRNG to improve the randomness complexity of a circuit in the presence of probing leakage. Extending this idea further, Coron et al. [CGZ20] proposed the use of *multiple (non-robust) PRGs* to further improve the randomness complexity of ISW-based constructions. Both the above constructions use a variant of mask refreshing called as locality refresh (LR). The *locality* of an intermediate variable (or more precisely, a wire in a circuit) refers to the maximum number of local random bits that any intermediate variable would depend upon (in addition to the original unmasked inputs). Improving the locality would reduce the number of true random bits input to a PRG. Accordingly, we bound the locality of the higher-order lookup table-based scheme [CRZ18] to $O(k' \cdot n)$ in Section 3. For completeness, we recall the original higher-order lookup table scheme in Section 2.

Next, we achieve an essentially constant RAM memory requirement (see Table 1) to store the masked lookup tables of [CRZ18] by evaluating the PRG *on-the-fly* and without explicitly storing these outputs. Precisely, in the proposed variant, each row of the masked lookup table has a *single* value unlike an n-tuple in the original scheme. Since the masked table constructed in the Step (Shift) $i+1$ is dependent on the masked table computed during Step i, there is a need to re-generate the random masks used in the Step i. This calls for careful tracking of PRG outputs. The memory and randomness optimised variants of the higher-order lookup table construction using robust and multiple PRG approaches are described in Section 4. Coron et al. [CRZ18] presented a variant of the lookup table-based higher-order maksing scheme with *increasing no. of shares*, originally proposed to reduce the time and randomness complexity further by a factor of *two*. We extend our work to this variant too. Section 5 describes the implementation of *increasing shares* variant using robust and multiple PRG techniques. We also discuss the details of packing the randomised table on architectures supporting large registers in Appendix A. While the packing into a large register is going to improve the speed of the offline computation, we have observed that it is going to increase the online execution time. All our constructions are secure in the probing model and also satisfies the SNI composibility notion.

Finally, Section 6 presents the implementation details for AES-128 and PRESENT block ciphers implemented in software using our method. Our target architecture is an ARM Cortex-M4 device with 256 KB of RAM memory. We also explore the possibility of optimising the time complexity using a tradeoff between the calls to TRNG and PRG. This optimisation achieves a balance between the memory complexity and the online execution time. We also compare the online execution time of our method with that of the 8-bit bitsliced implementation of AES-128. Tables 5 and 6 demonstrate the results of AES-128 implemented using robust and multiple PRG approaches, respectively. We compare the results of these approaches in terms of RAM memory, pre-processing and online execution times along with number of true random bytes required for varying number of shares n=3, 5, 7, 9 and 11. We would like to mention that all our implementation results are based on 8-bit implementations. At *tenth* order and with full pre-processing of a single execution of all the AES S-boxes, our technique requires only 41.7 KB of RAM memory, whereas the original scheme would have needed 440 KB. Moreover, our implementation results in Table 7 demonstrates that the online execution time of our variant is about 1.5 times faster compared to the 8-bit bitsliced implementation of AES-128. Similarly, Table 8 presents the results for PRESENT block cipher implemented using multiple PRG approach and also compares it with the implementation of PRESENT using [CRV15].

2 Recap of Higher-Order Masked Lookup Table Scheme from [CRZ18]

In [Cor14], to secure a (k, k') S-box S against a t-probing adversary, the author proposed a higher-order lookup table-based scheme with the number of shares $n = 2t + 1$. According to this scheme, each row is represented as an n-dimensional vector. In order to protect the S-box evaluation, shift the S-box lookup table by the input shares iteratively, one at a time. At the same time, the shifts by input shares are protected by masking table rows

Table 1: Comparison of asymptotic complexities of our work vs. [CRZ18] to implement a (k, k') S-box using higher-order masked lookup table scheme. The schemes are compared in terms of locality, RAM memory in bits, number of TRNG and PRG calls, and time to generate a pseudorandom bit.

	Locality	RAM	$\#TRNG$	$\#PRG$	time PRG
[CRZ18] Normal shares	–	$2^{k+1} \cdot k' \cdot O(n)$	$2^k \cdot k' \cdot O(n^2)$	0	–
[CRZ18] Increasing shares	–	$2^{k+1} \cdot k' \cdot O(n)$	$2^{k-1} \cdot k' \cdot O(n^2)$	0	–
Our work - single robust PRG	$O(k' \cdot n)$	$2^{k+1} \cdot k' + O(k' \cdot k \cdot n^3)$	$O(k' \cdot k \cdot n^3)$	$2^k \cdot O(k' \cdot n^2)$	$O(k' \cdot k^2 \cdot n^3)$
Our work - multiple PRG	$O(k')$	$2^{k+1} \cdot k' + O(k' \cdot k \cdot n^2)$	$O(k' \cdot k \cdot n^2)$	$2^k \cdot O(k' \cdot n^2)$	$O(k' \cdot k^2 \cdot n)$

using distinct output masks. Moreover, table entries are randomised across shifts using *refreshmasks* from [RP10]. Essentially, at the end of shift by the last but one share x_{n-1}, for every index $u \in \{0,1\}^k$ of the table T, the following holds:

$$\bigoplus_{1 \leq i \leq n} T(u) = \mathrm{S}(u \oplus x_1 \oplus \ldots \oplus x_{n-1}). \tag{3}$$

After shifting the table by $n-1$ input shares, the final output sharing is obtained by a single lookup of table T at the index x_n. We can observe that the final share x_n is used only in the last step. Hence, it is possible to *pre-process* the randomised lookup table independent of the secret by choosing uniform random values for $n-1$ input shares. This *pre-processing* significantly reduces the amount of computation after the availability of the secret input x, thus reducing the *online* execution time.

Recently in [CRZ18], Coron et al. improved the security proof of the higher-order S-box implementation from [Cor14] with the number of shares $n = t+1$ instead of $n = 2t+1$ to achieve t-th order probing security. Moreover, the S-box gadget from [CRZ18] was proven t-SNI secure under composition i.e, combining any number of t-SNI secure gadgets results in a t-SNI secure gadget. These improvements are possible due to the seminal work of Barthe et al. [BBD+16], in which the authors introduced a stronger security notion of the probing leakage model under compositions. This framework guarantees the circuit to be t-probing secure using $n = t+1$ shares only. For completeness, we recollect *refreshmasks* procedure [RP10] and the higher-order lookup table-based scheme from [CRZ18] in Algorithms 1 and 2, respectively.

In addition to the improved security proof, [CRZ18] discusses another refinement to reduce the randomness complexity to approximately half of the original scheme. The idea behind this improvement is that, while shifting the table using input share x_i, the masked S-box values can be protected using only $i-1$ output masks, instead of $n-1$ masks as in the original scheme. Table 1 presents the randomness and memory complexity of this variant. It can be observed from Table 1 that the randomness complexity is exponential in k and also depends on the masking order t. Generating exponentially many random values may involve significant overheads for higher values of t, particularly for resource-constrained devices.

Algorithm 1: *refreshmasks* [RP10].

```
Input  : $x_i$, $1 \leq i \leq n$.
Output : $y_i, 1 \leq i \leq n$ such that $\bigoplus_{1\leq i\leq n} y_i = \bigoplus_{1\leq i\leq n} x_i$.
$y_1 \longleftarrow x_1$
for $j \leftarrow 2$ to $n$ do
    $s \xleftarrow{\$} \{0,1\}^{k'}$
    $y_j \longleftarrow x_j \oplus s$
    $y_1 \longleftarrow y_1 \oplus s$
end
return $y_1, \ldots, y_n$
```

3 Improving Locality of the Scheme Using Locality Refresh

The objective of *refreshmasks* procedure (see Algorithm 1) is to re-randomise the n-sharing of the input x. This is achieved by xoring input share x_i, $i \leq 2 \leq n$ with random mask s_i at the same time adding the same mask to the first share. For ease of explanation, we will be using the first share to add the remaining random masks, unlike the last index n as in the original scheme. Precisely, it is a *re-sharing* of the input x. From this procedure, it can be observed that the first output share is computed using x_1 and $n-1$ random masks whereas all the remaining output shares are of the form $x_i \oplus s_i$, $2 \leq i \leq n$. The procedure *refreshmasks* is called a total of $n-1$ times, once per shift, while constructing a randomised lookup table. Finally, the output $T(x_n)$ is refreshed before the final output. Hence, the first share in the final sharing of $\mathrm{S}(x)$ is influenced by a total of $k' \cdot n \cdot (n-1)$ bits of randomness thus resulting in a larger *locality* value. In simple terms, the amount of randomness any variable would depend as part of the implementation is referred as *randomness locality*. Formally,

Definition 1. Locality of Randomness ([IKL⁺13]). A circuit C is said to have l-locality if the value of each of its intermediate/output variables depends on its unshared input and at most l-random bits used in the circuit.

Theorem 1. *[CGZ20, Theorem 2]. A gadget obtained by composition of l-locality gadgets has l-locality.*

One has to assume that a gadget always receives the locality refreshed inputs. Hence, the randomness from these refreshed inputs should be taken into consideration while computing the locality of a gadget. According to the Definition 1, the *locality* of the higher-order scheme recalled in Algorithm 2 is $(n-1)^2$. It is possible to reduce the randomness complexity of the implementation by reducing the locality of the scheme. As suggested by the authors of [IKL⁺13], replacing *refreshmasks* with *locality refresh* (LR) would bring down locality of the circuit. For completeness, we recall the *locality refresh* [CGZ20] in Algorithm 3.

Security Models. For completeness, we recall the various security notions followed for proving the SCA countermeasures. For the ease of definitions, we consider a gadget G that takes an n-sharing of a single input x and outputs an n-sharing of $y = f(x)$.

Definition 2. Soundness of t-private circuit [ISW03]. A private circuit implementing a function $f : \{0,1\}^k \rightarrow \{0,1\}^{k'}$ is a transformer (I, C, O) where $I : \{0,1\}^k \rightarrow \{0,1\}^{k_i}$ is an encoder and $C : \{0,1\}^{k_i} \rightarrow \{0,1\}^{k_o}$ is a Boolean circuit using randomness $A \in \{0,1\}^p$. Then, $Pr[(O(C(I(x))), A) = f(x)] = 1$ for any input $x \in \{0,1\}^k$ where the probabilities are over the randomness used in I and C.

Algorithm 2: Computation of $y = \mathrm{S}(x)$ using higher-order masked lookup table computation [CRZ18, Algorithm 1, Section 3].

Input :

- x_i, $1 \leq i \leq n \in \{0,1\}^k$.
- An (k, k') S-box lookup table.

Output : $y_i, 1 \leq i \leq n-1$ and $y_n := \mathrm{S}(x) \oplus y_1 \oplus \ldots y_{n-1} \in \{0,1\}^{k'}$.

```
for $u \leftarrow 0$ to $2^k - 1$ do
    $T(u) \longleftarrow (\mathrm{S}(u), 0, \ldots, 0)$ // initialise table to S-box
end
for $i \leftarrow 1$ to $n - 1$ do
    for $u \leftarrow 0$ to $2^k - 1$ do
        for $j \leftarrow 1$ to $n$ do
            $T_{\mathrm{aux}}(u)[j] \longleftarrow T(u \oplus x_i)[j]$ // shift by $x_i$
        end
    end
    for $u \leftarrow 0$ to $2^k - 1$ do
        $T(u) \longleftarrow refreshmasks(T_{\mathrm{aux}}(u))$ // refresh table rows
    end
end
$y_1, \ldots, y_n = refreshmasks(T(x_n))$
return $y_1, \ldots, y_n$
```

The first formal study of noisy leakage using the *noisy leakage model* has been introduced by Chari et al. [CJRR99] and extended by [PR13] demonstrated that the number of samples required to recover the secret is at least exponential in the masking order t. According to this model, the adversary obtains the noisy leakage of the intermediate variables during the computation. In practice, the noisy leakage model is often considered more realistic, since experimental leakages are noisy. However, as stated in [DDF14a], this security notion is not always convenient for cryptographic setting.

Ishai et al. [ISW03] initiated the study of the formal security of masked algorithms against t-probing attacks such that the private circuit remains secure when the adversary can observe at most t internal wires. The security proofs demonstrate that the knowledge of t probes cannot give the attacker any information about the secret, by assuming that their leakages are independent and sufficiently noisy. However, this assumption may not hold for parallel implementations which can possibly leak all the shares that are manipulated at the same time [BDF$^+$17]. Thus, the probing model is more suitable to prove the security of the serial implementations.

In 2014, Duc et al. [DDF14a] show that the security in the noisy leakage model of [PR13] can be reduced to the security in the t-threshold probing model of [ISW03]. Hence, it is necessary to prove a masked implementation secure in the t-probing model since insecurity in the probing model usually leads to attacks in other models.

Definition 3. t-probing security [ISW03]. A circuit C is a private implementation of f using encoder I and decoder O is t-secure against probing attacks if for any pair of inputs $x, x' \in \{0,1\}^k$, and let $\mathrm{T} \subset C(I(x), A)$ and $\mathrm{T}' \subset C(I(x'), A)$ be the set of at most t observations anywhere in the circuit implementation, then the distribution of T and T' are identical.

Definition 4. Non-Interference (t-NI) Security [BBD$^+$16]. Let a gadget G take an n-sharing of x as input and output an n-sharing of y, where $y = f(x)$. Let a set of t_1

Algorithm 3: Locality Refresh (LR) [CGZ20].

```
Input  : $x_i$, $1 \le i \le n$.
Output : $y_i, 1 \le i \le n$ such that $\bigoplus_{1 \le i \le n} y_i = \bigoplus_{1 \le i \le n} x_i$.
$y_1 \longleftarrow x_1$                          ▷ $y_{1,1}$
for $j \leftarrow 2$ to $n$ do
    $s \xleftarrow{\$} \{0,1\}^{k'}$                ▷ $s_{j-1}$
    $y_1 \longleftarrow y_1 \oplus (s \oplus x_j)$   ▷ $y_{1,j} = x_1 \bigoplus_{1 \le k \le j} (s_k \oplus x_k)$
    $y_j \longleftarrow s$
end
return $y_1, \ldots, y_n$
```

input/intermediate variables are observed along with t_o output shares where $(t_1 + t_o) \le t < n$. Then, G is called t-NI secure gadget if the observed values can be perfectly simulated by using a maximum of $t_1 + t_o$ input shares of x.

A gadget is t-SNI secure if the number of inputs required for simulation are bounded only by the number of observations on the input/intermediate variables and not on the output shares. Formally,

Definition 5. Strong Non-Interference (t-SNI) Security [BBD+16]. Let a gadget G take an n-sharing of x as input and output an n-sharing of y, where $y = f(x)$. Let a set of t_1 input/intermediate variables are observed along with o output shares where $t_1 + t_o \le t < n$. Then, G is called t-SNI secure gadget if the total set of observations can be perfectly simulated by using a maximum of t_1 input shares of x.

We need to ensure that replacing the *refreshmasks* by the LR procedure does not affect the security guarantees of the higher-order lookup table scheme. Hence, Lemmas 1 and 2 present the security properties of the LR procedure that are instrumental in the t-SNI security proof of the scheme. Even though the security proofs are very similar to that of the *refreshmasks* discussed in [CRZ18], we present them for the sake of completeness. Precisely, Lemma 1 is to show that it is possible to assign any t output shares uniform random values whereas Lemma 2 is to show the simulation of the LR procedure.

Lemma 1. *Any $n-1$ output shares of LR procedure in Algorithm 3 are independent of the unshared input.*

Proof. It is trivial to prove the claim if y_1 is not probed. This is due to the fact that the output shares $y_i, 2 \le i \le n$ in Algorithm 3 are nothing but random values. Hence, these output shares can be simulated independent of the secret, x. If y_1 is probed, there exists at least one unprobed index i, $i \notin I$ such that $y_1 = x \oplus s_i \bigoplus_{1 \le k \le j, k \ne i} s_k$. Since s_i acts as a one-time pad to mask x, assign a uniform random value to y_1. □

Lemma 2. *The LR gadget in Algorithm 3 is t-NI secure.*

Proof. To prove the t-NI security of a gadget (see Definition 4), we construct an index set I such that the observed values can be simulated using input share indices from I. The index set I is constructed as follows: when the variables x_i or y_i, $1 \le i \le n$, are probed, add i to I. Also, when $s_i, 1 \le i \le n-1$ is probed, add i to I. For the intermediate variables $y_{1,j}$, $2 \le j \le n$, add j to I.

After constructing the set I, it is straightforward to show the simulation of probed variables using the input share indices from I. Each random variable $s_i, 1 \le i \le n-1$ is assigned a uniform random value as this would have happened in the actual execution. Any probed output shares $y_i, 2 \leq i \leq n$ can be simulated since $y_i = s_i$, for $2 \leq i \leq n$. First output share y_1 can be assigned a uniform random value using Lemma 1. The simulation of the intermediate variable of the form $y_{1,j}$, $1 \leq j \leq n-1$ has two cases. If there exist an index j' such that $1 \leq j' \leq j$ and $j \notin I$, then we assign an uniform random value to $y_{1,j}$. This is because the random variable $s_{j'}$ is not probed and acts as a one-time pad. We compute the value of $y_{1,j}$ using the indices $i \in I$ such that $1 \leq i \leq j$, otherwise. Since only one input index per observation is added to I and at most $t = n-1$ values only can be observed, $|I| < n$. Hence, we show the simulation using $t_1 + t_o \leq t$ input shares of x. □

Even though reducing the randomness locality helps to reduce the number of true random values required for the implementation, there is still an overhead associated with the amount of *RAM memory* required to store the randomised lookup table (see Table 1). The amount of memory will be a bottleneck for higher-order implementations, particularly for resource-constrained devices. Therefore, there is a need to optimise not only the randomness complexity but also the memory complexity of lookup table-based countermeasures to achieve the practicality of this class of side-channel countermeasures.

4 RAM Memory Optimisation of Masked Lookup Table

After minimising the *randomness locality* of the higher-order lookup table scheme as described in the previous section, the next step will be to attain the randomised lookup table with a single column, thus achieving an almost constant RAM memory (up to logarithmic factor in the size of the S-box) for the overall implementation of the higher-order lookup table-based scheme [CRZ18]. This section discusses our approach to arriving at the desired result by adapting robust and multiple PRG techniques [IKL+13, CGZ20].

4.1 No. of Columns of Lookup Table Independent of Masking Order

It is easy to see from the higher-order lookup table scheme (see Algorithm 2) that the only step which uses random values is re-sharing (see Algorithm 1), which is to be replaced by LR in our scheme (see Algorithm 4). The idea is to generate the random masks required by the LR algorithm (see Algorithm 3) from a PRG. Moreover, we will store only the first column of the lookup table while the remaining $n-1$ columns are computed *on-the-fly* by the PRG. There are a total of $n-1$ shifts by the input shares where each shift requires $(n-1) \cdot 2^k$ pseudorandom values, each of size k'-bits. Hence, the total number of pseudorandom bits for implementing the higher-order lookup table-based scheme is given by:

$$k' \cdot 2^k \cdot (n-1)^2. \tag{4}$$

We would like to highlight a few observations here. Algorithm 3 takes an n-sharing of the input and returns a re-sharing of the same input of which $n-1$ shares are nothing but random values. As part of the higher-order lookup table scheme, these re-shared outputs are the inputs to the next shift (Step 7 of Algorithm 2). When the output masks are computed *on-the-fly* i.e. the values are not explicitly stored, random masks of the previous shift are to be *recomputed* during further shift (see Remark 1). Precisely, the LR procedure after shift by x_{i+1} requires the output masks of LR after shift by x_i (see Remark 2). One more point to observe is when the table T is shifted by x_{i+1}, the value at the table index u in i^{th}-shift will be at $u \oplus x_{i+1}$ in $(i+1)^{\text{th}}$-shift (Step 7 of Algorithm 2).

To facilitate the recomputation of pseudorandom values on-the-fly, we need to keep track of the shift, row, and column indices and pass on these indices as inputs to the PRG construction. Further, assign a unique index, say α, per every random mask that is generated as part of Algorithm 3. Needless to say, the PRG construction will return the same pseudorandom value if called on the same value of α. The upper bound on the range of unique index α is the number of pseudorandom values required for the construction of randomised lookup table (4). As explained above, the input share x_{i+1} is also required along with the shift and the row indices to recompute the output masks of the previous shift. Also, the input share x_n is passed as input to mask refresh in the final table lookup (Step 19 of Algorithm 4).

Remark 1. The idea of the lookup table scheme using PRG is to discard the output masks generated by PRG without storing them. As explained in Section 3, the randomised lookup table to be computed during the shift by x_{i+1} depends on the lookup table constructed in the shift by x_i. Concretely, inside the LR procedure, entries of the randomised lookup table after the $(i+1)^{\text{th}}$-shift are to be xored with the masks generated during the i^{th}-shift, $i \geq 2$. Hence, there is a need to recompute the random masks of the previous shift.

Remark 2. One more observation here is that using LR instead of *refreshmasks* not only reduces the randomness locality of the scheme, but also reduces the number of calls to PRG to *recompute* the random values of previous shifts. This is possible since LR limits the dependence of output masks in shift $i+1$ only to shift i, unlike *refreshmasks*. Therefore, we need to compute output masks of i^{th}-shift in $(i+1)^{\text{th}}$-shift using LR vs. masks of $(1,\ldots,i)$ shifts in $(i+1)^{\text{th}}$-shift using *refreshmasks*. Concretely, the number of calls to PRG to recompute random values with *refreshmasks* and LR are $(n-1)^2 \cdot (n-2) \cdot 2^{k-1}$ and $(n-1) \cdot (n-2) \cdot 2^k$, respectively. Hence, the number of calls to PRG in turn depends on the locality.

4.2 Masking Lookup Tables using *robust* PRG

Ishai *et al.* [IKL+13] introduced the concept of a *robust* PRG where the randomness required for a secure implementation of a cryptographic circuit, say C, can be obtained from a robust PRG instead of a TRNG, without degrading the security. This substitution indeed helps to reduce the randomness complexity of the circuit C. Precisely, the robust PRG is associated with a parameter r such that the outputs of PRG construction remain r-wise independent even if the adversary can leak a set of intermediate values, say A, using at most t probes anywhere in the circuit C. We follow the weaker notion of strong robustness as explained in [CGZ20] since it leads to slightly more efficient construction in terms of the number of true random inputs to PRG. We recall the definition of a strong robust PRG.

Definition 6. Strong robust PRG ([IKL+13, CGZ20]). A circuit implementation C of a PRG $G : \{0,1\}^m \rightarrow \{0,1\}^p$ is strong (r,t,q)-robust if given $Y = G(X)$ where $X \leftarrow \{0,1\}^m$ and q is a parameter, for any set R of at most t-probes in C, there is a set A of at most $q \cdot t$ output bits, such that by fixing the values C_R of the wires in R and of the Y_A output bits, the values $Y_{\overline{A}}$ of the output bits not in A are $(r - q|R|)$-wise independent and uniformly distributed.

Trivial construction [CGZ20]. We would like to implement a strong (r,t,q)-robust PRG using *trivial construction*. We recall this construction in Lemma 3. This is obtained by xoring the outputs of $t+1$ non-robust r-wise independent PRGs. It can be observed from the robust PRG definition from [IKL+13] that the parameter q indicates the impact of observing the input wires of PRG on the outputs.The parameter $q = 1$ for the robust PRG constructed using trivial construction. The reason being out of the $t+1$ outputs from (non-robust) r-wise independent PRGs, at least one of the output remains unprobed. Hence, that unprobed value acts as a one-time-pad.

Definition 7. r-wise independent PRG. A PRG $g : \{0,1\}^m \rightarrow \{0,1\}^p$ is said to be r-wise independent if any set of at most r output bits of $g(x)$ are uniform and independent of x when $x \xleftarrow{\$} \{0,1\}^m$.

Lemma 3. *[CGZ20, Lemma 2] Let $g : \{0,1\}^m \rightarrow \{0,1\}^p$ be a r-wise independent PRG. A strong $(r,t,1)$-robust PRG $G : \{0,1\}^{m\cdot(t+1)} \rightarrow \{0,1\}^p$ can be obtained such that: $G(x_1,\ldots,x_{t+1}) = g(x_1) \oplus \ldots \oplus g(x_{t+1})$.*

One way to implement each of these r-wise independent (non-robust) PRGs is by evaluating a random $r-1$ degree polynomial over a finite field. Let $\mathbb{F}$ be a field of order 2^β such that the representation of each element needs β bits. The higher-order lookup table scheme contains $n-1$ shift and LR operations of the table, and each such LR requires $(n-1)\cdot 2^k \cdot k'$ bits. Moreover, the final LR requires $k' \cdot (n-1)$ random bits. Overall randomness per S-box is given by,

$$\Big(k' \cdot (n-1) \cdot \big(2^k \cdot (n-1) + 1\big)\Big).$$

The following inequality should hold,

$$\Big(k' \cdot (n-1) \cdot \big(2^k \cdot (n-1) + 1\big)\Big) \leq (\beta \cdot 2^\beta). \tag{5}$$

The seed to PRG is nothing but the coefficients of this $r-1$ degree polynomial. It turns out that the alternative construction of robust PRG based on expander graphs [IKL$^+$13] becomes expensive for our approach (see Remark 3).

Remark 3. As mentioned in [CGZ20], the expander graph robust PRG construction based on unbalanced bipartite expander graphs become better than the trivial construction only for $r \geq 2^{18}$ and at least 2^{36} random input bits. For the majority of the practical implementations of higher-order lookup table-based masking schemes on resource-constrained devices, the amount of randomness will be much lower than 2^{36}. Even though the expander graphs can optimise the time generation of each pseudorandom value, our primary focus is to reduce the memory required to store the true random input seed to PRG. Therefore, we choose to go with the trivial construction instead of expander graph-based construction.

We summarise the steps involved in the higher-order masked lookup table-based computation of an S-box using a robust PRG in Algorithm 4. The *LR-n-r* procedure to compute pseudorandom outputs using robust PRG is described in Algorithm 5. We would like to mention that the input and the auxiliary tables are each of size $k' \cdot 2^k$ bits only.

Lemma 4. *The randomness locality of the (k,k') S-box gadget in Algorithm 4 is $k' \cdot 2 \cdot (n-1)$ bits.*

Proof. The S-box gadget takes an n-sharing as input and outputs the masked S-box computation such that $\bigoplus_{1\leq i\leq n} y_i = \mathrm{S}(x)$. The masked table construction contains a sequence of $n-1$ shifts by input shares followed by LR. During each shift by x_i, the lookup table T depends on input shares $x_1,\ldots,x_i$. Also, the LR process re-shares the table row with the help of $n-1$ random output masks. As part of this process, add input share along with a random mask, s_i to the first column. Also, assign the same random mask s_i to the output share. At step i of LR, $i-1$ random masks would have been added to the first index. By the end of step $n-1$, the first and the remaining output shares depend on $k' \cdot (n-1)$ bits and k' bits, respectively. One interesting property of the LR procedure is that, after any given LR, the output shares depend only on $n-1$ random values. Therefore, by the end of the shift by x_{n-1}, the first row of table T depends only on $n-1$ input shares and $n-1$ random output masks from LR, each of size k' bits. Therefore, the randomness locality of Algorithm 4 is $k' \cdot (n-1+n-1) = k' \cdot 2 \cdot (n-1)$ bits. □

We need to set the value of the parameter r for the robust PRG construction such that the implementation of the robust PRG remains secure against t probes. We recall the following theorem from [IKL$^+$13] which states that the randomness required for the implementation can be generated from a robust PRG that remains secure under probing attacks.

Algorithm 4: Higher-order masked lookup table computing output masks on-the-fly using *robust* PRG.

```
Input :
    • $x_i$, $1 \le i \le n \in \{0,1\}^k$.
    • A $(k, k')$ S-box lookup table.
Output: $y_i, 1 \le i \le n-1$ and $y_n := \mathrm{S}(x) \oplus y_1 \oplus \ldots y_{n-1} \in \{0,1\}^{k'}$.
for $u \leftarrow 0$ to $2^k - 1$ do
    $T(u) \longleftarrow S(u)$
end
for $i \leftarrow 1$ to $n-1$ do
    for $u \leftarrow 0$ to $2^k - 1$ do
        $T_{\mathrm{aux}}(u) \longleftarrow T(u \oplus x_i)$
    end
    for $u \leftarrow 0$ to $2^k - 1$ do
        $T(u) \longleftarrow \big(\mathit{LR\text{-}n\text{-}r}\ (T_{\mathrm{aux}}(u), i, u, x_i)\big)[0]$
        // LR-n-r is described in Algorithm 5
        // Copy only the first entry from LR-n-r output
    end
end
$y_1, \ldots, y_n = \mathit{LR\text{-}n\text{-}r}\ (T(x_n), n-1, x_n, 0)$
```

Algorithm 5: *LR-n-r*: LR procedure for normal variant using the robust PRG.

```
Input : $y_1, i, u, x_i$.
// $y_1$ is the $u$-th table entry after the $i$-th shift by $x_i$.
Output: $z_1, \ldots, z_n$.
$z_1 \longleftarrow y_1$
for $j \leftarrow 2$ to $n$ do
    $s^* \longleftarrow 0$
    // recomputation of the $(i-1)^{\mathrm{th}}$ shift randoms.
    if $(2 \le i \le n-1)$ then
        $\alpha \longleftarrow \Big(\big((i-2)\cdot(n-1)\cdot 2^k\big) + \big((u \oplus x_i)\cdot(n-1)\big)\Big) + (j-1)$
        $s^* \xleftarrow{\mathbb{F}_{2^k}} PRG_r(\alpha)$
        // output of the strong $(r,t,1)$-robust PRG at $\alpha$
    end
    $\alpha \longleftarrow \Big(\big((i-1)\cdot(n-1)\cdot 2^k\big) + \big(u\cdot(n-1)\big)\Big) + (j-1)$
    $s \xleftarrow{\mathbb{F}_{2^k}} PRG_r(\alpha)$
    $z_j \longleftarrow s$
    $z_1 \longleftarrow z_1 \oplus (s^* \oplus z_j)$
end
return $z_1, \ldots, z_n$
```

Theorem 2. *[IKL*$^+$*13, Theorem 30] Suppose C is a t-secure implementation of a function f that requires p random bits for implementation such that the randomness locality of the circuit is l. Let $G : \{0,1\}^m \rightarrow \{0,1\}^p$ be a strong (r,t,q)-robust linear PRG with a parameter $r \geq t \cdot max(l,q)$. Then the circuit $\hat{C}$ is a t-secure implementation of f constructed using G that requires m random bits.*

From Theorem 2, set the value of r-wise independent parameter to $t \cdot l$. Since we opt to use trivial construction to obtain strong $(r,t,1)$-robust PRG by combining outputs of $n = t+1$ polynomial evaluations over a finite field $\mathbb{F}$ of size $\beta = O(k)$ (from (5)), the overall randomness complexity of the masked lookup table scheme is

$$(t+1) \cdot r \cdot \beta = n \cdot \left(k' \cdot 2 \cdot (n-1)^2\right) \cdot O(k) = O(k' \cdot n^3 \cdot k).$$

It takes $O(k' \cdot n^3 \cdot k^2)$ bit operations to generate a pseudorandom value. Needless to say, each output of PRG is of length β bits. As already mentioned, the PRG construction outputs the β-bit pseudorandom value based on the unique index α that we compute using the lookup table indices. Overall, the total memory required for the scheme in Algorithm 4 is nothing but the size of a single column of the lookup table plus the size of the input seed to each of the $t+1$ PRGs. We summarise in Table 2 the formulas for α computation and recomputation along with the total memory required for the masked S-box computation using the higher-order lookup table-based scheme.

Since the scheme in Algorithm 4 is analogous to [CRZ18, Algorithm 1, Section 3], even the t-SNI security proof of our approach is almost identical to the proofs discussed in [Cor14, CRZ18]. Of course, one immediate difference here in our case is that the random values used for masking are the outputs of a robust PRG instead of the outputs of a TRNG. To bridge this gap between a TRNG and a robust PRG, we need the following lemma from [CGZ20] to show that any $(r - q \cdot |U|)$ outputs of PRG can be assigned uniform random values and any t internal observations from a PRG can be simulated using a subset of at most $q \cdot |U|$ output bits of PRG. Since we are using trivial construction to implement the robust PRG, the parameter $q = 1$. Hence, probing any t internal wires of PRG construction gives no more advantage than probing t outputs of the TRNG.

Lemma 5. *(Robust PRG)[CGZ20, Lemma 3] Let $G : \{0,1\}^m \rightarrow \{0,1\}^p$ be a strong (r,t,q)-robust linear PRG with $r \geq t \cdot q$. Let U be a set of at most t probes in G and L be any subset of $(r - q|U|)$ output bits of PRG. There exist a subset T with $|T| \leq q|U|$ such that the distribution of $V = G_{L \cup T}(X)$ is uniform in $\{0,1\}^{|L \cup T|}$ when $X \leftarrow \{0,1\}^m$. Moreover, $G_U(X)$ can be efficiently simulated given V_T only.*

Theorem 3. *The higher-order masked lookup table-based S-box computation in Algorithm 4 locality refreshed using Algorithm 5 (with the strong $(r,t,1)$-robust PRG) where the adversary observe t_1 intermediate variables and t_o output variables such that $t_1 + t_o \leq t < n$ is t-SNI secure.*

Proof idea. The idea behind the proof is as follows: table T is shifted by $x_i, 1 \leq i \leq n-1$ and each shift is followed by the LR of each row of T. After this sequence of operations, a table lookup of T at x_n-th row will give the n-sharing of $\mathrm{S}(x)$ and a final LR is performed on the shares of $\mathrm{S}(x)$. Hence, there are a total of $n-1$ shifts and LRs followed by a final table lookup and LR. When the internal variables of the LR are not probed, then any $n-1$ output shares of that LR can be assigned uniform random values, without requiring the knowledge of input shares. Whereas if the intermediate variables of i^{th} shift and LR are probed, then we need the knowledge of the input share x_i for simulation. Essentially, the number of input shares required for the simulation depends on the number of intermediate observations made by the adversary. Not only that, since the PRG construction outputs are r-wise independent, we must use at most r bits of randomness while showing the simulation of the observed variables.

Refer to Appendix B.1 for the detailed t-SNI security proof.

4.3 Lookup Table Construction using Multiple PRGs

Coron *et al.* [CGZ20] introduced the notion of *multiple independent PRGs* wherein replace a single robust PRG with multiple PRGs, such that the PRG construction no more requires the robustness property. Adapting this notion for lookup table setting, even though we could not achieve any asymptotic improvement in randomness complexity compared to robust PRG setting, we still can reduce the time to generate a pseudorandom to $\tilde{O}(n)$ from $\tilde{O}(n^3)$. This reduction is possible since the randomness locality of each of these multiple PRGs is defined only w.r.t. to a subset of random values, unlike the entire circuit in the case of robust PRG. For completeness, we recall the definition of locality w.r.t. a subset of randomness.

Definition 8. l-local gadget with randomness subset ([CGZ20]). Let G be a gadget and let A be a subset of the randomness used by G. The gadget G makes an l-local use of its randomness if any intermediate variable of G depends on at most l-random bits from the set A.

To implement Algorithm 4 with multiple PRGs approach, one interesting observation is that while generating random values using PRG for output masks $j = 2, \ldots, n$, the values generated for two distinct row indices, say j_1 and j_2, $j_1 \neq j_2$, are never combined as part of the algorithm. Intuitively, we can use an independent PRG per column of the S-box computation. But there is an issue with this approach. Since the adversary can potentially leak all the output values of a non-robust PRG with a single probe, the column values across multiple LR operations are no longer independent which is contrary to the sole purpose of using *refreshmasks* in [Cor14]. Therefore, we need to use an independent non-robust PRG per column per shift by input share. Also, $n-1$ PRGs in the final LR after the table lookup at x_n. A total of $n \cdot (n-1)$ independent PRGs are required for implementing a higher-order lookup table-based scheme using [CRZ18] algorithm. In spite of $n \cdot (n-1)$ PRGs required for the implementation, we need to store *only the true random input seeds of* $n-1$ *PRGs* after the offline phase. This reduction in memory is possible because the masked lookup table after the shift and LR by x_{n-1} depends only on the random masks generated during the $(n-1)^{\text{th}}$ LR. Hence, after pre-processing the masked lookup table, discard the input seeds of PRGs used prior to shift by x_{n-1}. Of course, we still need to store the input seed to PRGs for the final LR. Overall, reducing the total memory required for the scheme further by a factor of (approximately) $(n-1)$.

The randomness locality is now defined only with respect to the set of random values per column, $y_{i,j}$ where i represents the index of input share x_i and j represents the column index, $2 \leq j \leq n$. As mentioned earlier, since these values belonging to a subset of random masks used in the lookup table are never combined, the locality for each PRG is set to

$$l = 1.$$

Now, to determine the r-wise independence parameter for each PRG, we recollect the following theorem from [CGZ20].

Theorem 4. *[CGZ20, Theorem 7] Let C be a t-secure circuit implementing a function f. The circuit requires the set of random values A_i, $1 \leq i \leq \gamma$ such that each set uses p random bits and has a randomness locality l (see Definition 8). An adversary can get all random bits of any A_i, $i = 1, \ldots, \gamma$ with a single probe. Let $G : \{0,1\}^m \to \{0,1\}^p$ be a $l \cdot t$-wise independent PRG. Then, the circuit $\hat{C}$ implementing the same function f remains t-secure if the random values required by the sets A_i are replaced by PRGs G_i, $i = 1, \ldots, \gamma$, such that the circuit $\hat{C}$ requires $\gamma \cdot m$ random bits for the implementation.*

As per Theorem 4, the corresponding security parameter per PRG is set to $r = t \cdot l = t$. Therefore, to obtain t-wise independence, each such PRG can be implemented using polynomial evaluation over a finite field $\mathbb{F}$ with t coefficients. Table 2 summarises the total memory required for the masked S-box computation using multiple PRG approach along with formulas for unique index α computation and recomputation during LR procedure. It can be observed from Table 2 that there is no asymptotic reduction in the randomness complexity between robust and multiple PRG approaches, we observe that for practical instantiations of masking order n, the use of multiple PRG leads to a slightly lesser number of input random bits.

For the multiple PRG approach, the sub-routine to compute pseudorandom values has to be updated accordingly, even though the remaining steps of the higher-order masked lookup table scheme in Algorithm 4 remain the same. Another change is w.r.t. α computation. The index α is unique per each PRG and not for the entire scheme. Since we use a PRG per shift per column, the column and shift indices together decide the index of the PRG. The multiple PRGs construction receives a pair of values as input where the first input parameter indicates the index of the PRG, then evaluates the PRG with the given index over the second input parameter α. Algorithm 6 describes the way of computing this pair of values during the LR procedure. The procedure calls to *LR-n-r* in the Step 9 of Algorithm 4 will be replaced by the *LR-n-m* which is described in Algorithm 6.

Algorithm 6: *LR-n-m*: LR procedure for normal variant using the multiple PRGs approach.

```
Input  : $y_1, i, u, x_i$.
// $y_1$ represents $T(u)$ after the $i^{\text{th}}$ shift by $x_i$.
Output : $z_1, \dots, z_n$.
$z_1 \longleftarrow y_1$
for $j \leftarrow 2$ to $n$ do
    $s^* \longleftarrow 0$
    // recomputation of the $(i-1)^{\text{th}}$ shift randoms.
    if $(2 \leq i \leq n-1)$ then
        $ind \longleftarrow \big((i-2)\cdot(n-1)\big) + (j-1)$
        $\alpha \longleftarrow u \oplus x_i$
        $s^* \xleftarrow{\mathbb{F}_{2^k}} PRG_m(ind, \alpha)$
        // output of $PRG_{ind}$ evaluated at $\alpha$
    end
    $ind \longleftarrow \big((i-1)\cdot(n-1)\big) + (j-1)$
    $\alpha \longleftarrow u$
    $s \xleftarrow{\mathbb{F}_{2^k}} PRG_m(ind, \alpha)$
    $z_j \longleftarrow s$
    $z_1 \longleftarrow z_1 \oplus (s^* \oplus z_j)$
end
return $z_1, \dots, z_n$
```

To prove the security using the multiple PRGs notion, Coron *et al.* [CGZ20] introduced an extended security notion called t-SNI-R according to which an adversary can obtain any subset of randomness (that is generated using a non-robust PRG) using a single probe. The extended model is to make sure that the adversary has no additional advantage when the intermediate values of the PRG construction are probed since he can get the entire subset with a single probe. Precisely, the impact of observing the internal wires of PRG on the output is given to the adversary. We slightly modified the t-SNI-R security notion to suit our setting and introduced t-SNI-R^* (see Remark 4). We would like to mention that the t-SNI-R^* notion is a generalisation of the prior t-SNI-R security notion.

To prove the security using the multiple PRGs approach, Coron *et al.* [CGZ20] introduced an extended security notion called t-SNI-R according to which an adversary can obtain any subset of randomness (generated from a non-robust PRG) using a single probe. The extended model is to make sure that the adversary has no additional advantage in probing the intermediate values of the PRG construction since he can get the entire subset with a single probe. Precisely, the adversary will obtain the shadow of observing the internal wires of PRG on the output. We adapt the t-SNI-R security notion to suit our setting and introduced t-SNI-R^*. We would like to mention that the t-SNI-R^* notion, is an extension of the prior t-SNI-R security notion, that follows the same security guarantees in addition to the notational changes mentioned in Remark 4. Formally,

Definition 9. t-SNI-R^*. Let G be a gadget taking an n-sharing of input x and outputs an n-sharing of $y = f(x)$. Let the randomness of the gadget G be divided into partitions $A_i, i = 1, \ldots, \gamma$ such that adversary can get all random bits of any A_i, $i = 1, \ldots, \gamma$ with a single probe. Consider any t_1 input/intermediate variables, any subset O of output shares, $t_o = |O|$ and any subset $R^* \subset [1, \gamma]$ such that $t_1 + t_o + |R^*| < n$. Then, there exists a set of input share indices I with $|I| \leq t_1$ such that the t_1 input observations, $y_{O^*} = \{y_i, y_i \in y_O \text{ or } y_i \in A_j, j \in R^*\}$ output shares along with the partition $A_i, i \in R^*$ can be perfectly simulated using $|I \cup R^*|$ input shares of x.

For instance, for AES-128 implementation using our higher-order lookup table-based construction with multiple PRGs, the adversary can get the subset $A_i^{(j)}, \forall j \in [1, 160]$ across the S-box gadgets. Thus, it is possible to generate column i of the randomised lookup table across the S-boxes using the same PRG. The following theorem shows the composition of t-SNI-R^* gadgets according to which the adversary can get the subset $A_i^{(j)}, 1 \leq j \leq M$ across the composed gadget, which is a combination of M such t -SNI-R^* gadgets. We mimic the security of this composition gadget from the t-SNI-R composition proof [CGZ20].

Theorem 5. *Let M represents the set of t-SNI-R^* gadgets, $G_i, i \in M, |M| = m$. The gadget obtained by any composition of m such gadgets is t-SNI-R^* where the randomness is considered as $\bigcup_{i \in M} A_j^{(i)}$ for $1 \leq j \leq \gamma$.*

The proof of Theorem 5 can be found in Appendix B.2.

Theorem 6. *The masked higher-order lookup table-based S-box computation in Algorithm 4 that is locality refreshed using Algorithm 6 (using multiple PRGs each with a locality $l = k'$) where the adversary observe t_1 intermediate variables and t_o output variables and any set $R^* \subset [1, n \cdot (n-1)]$ such that $t_1 + t_o + |R^*| < n$ is t-SNI-R^* secure. Moreover, all values from the randomness partition $A_z, z \in R^*$ can be perfectly simulated using at most $t_1 + |R^*|$ input shares of x.*

Proof idea. The security proof of the higher-order lookup table using the multiple PRGs approach is similar to Theorem 3 with the change that the random values are generated using the multiple PRGs approach instead of a single robust PRG. Moreover, while using the multiple PRGs technique in the memory optimised variant of the table-based construction, we considered the set of random masks per column per shift as one of a partition of the overall randomness. In turn, this subset is generated from a non-robust polynomial-based PRG construction. While proving the security of our construction in the extended t-SNI-R^* security model, we need to show the simulation of the entire subset even if a single random value from the set is probed.

We provide the full security proof in Appendix B.3.

Remark 4. In the t-SNI-R definition introduced by [CGZ20], the security notion considers the case where the randomness of the implementation is divided into n partitions,

Table 2: Summary of the total number of bits of pseudorandom values, true random values, memory required per (k, k') S-box along with formulas for index α computation and recomputation of Algorithms 5 and 6 using robust PRG and multiple PRG approaches, respectively. Here, i represents the shift index, u is the row index and j represents the column index.

Description	**Value**
# pseudorandom	$k' \cdot (n-1) \cdot \big(2^k \cdot (n-1) + 1\big)$.
Algorithm 4 using strong robust PRG	
# input to PRG	$\beta \cdot k' \cdot 2 \cdot n \cdot (n-1)^2$.
Memory required	$k' \cdot \big(2 \cdot 2^k + \beta \cdot 2 \cdot n \cdot (n-1)^2\big)$.
Index α computation	$\big((i-1) \cdot (n-1) \cdot 2^k\big) + \big(u \cdot (n-1)\big) + j - 1$.
Index α recomputation	$\big((i-2) \cdot (n-1) \cdot 2^k\big) + \big((u \oplus x_i) \cdot (n-1)\big) + j - 1$.
Algorithm 4 using multiple PRG	
# input to PRG	$\beta \cdot k' \cdot (n-1)^3$.
Memory required	$k' \cdot \big(2 \cdot 2^k + \beta \cdot 2 \cdot (n-1)^2\big)$.
(index to PRG, α) computation	$\Big(\big((i-1) \cdot (n-1) + j - 1\big), u\Big)$.
(index to PRG, α) recomputation	$\Big(\big((i-2) \cdot (n-1) + j - 1\big), (u \oplus x_i)\Big)$.

$A_1, \ldots, A_n$. The randomness for the masked S-box using lookup table-based implementation is divided into $n \cdot (n-1)$ partitions. Moreover, the indices of PRGs used to generate output shares are $[((n-1)^2 + 1), (n \cdot (n-1))]$, and not $[1, n-1]$. Hence, we can not directly use the same notation suggested by [CGZ20]. We introduce an extended security notion t-SNI-R* where the adversary can still get any partition A_i using a single probe. One more subtle difference between the ISW-based implementation from [CGZ20] and our approach is we use a dedicated PRG per column and the final output shares, $y_i, 2 \leq in$ are nothing but random masks generated from $n-1$ PRGs. This difference makes the our proofs elegant compared to the proofs from [CGZ20].

5 Lookup Table With Increasing Shares Using PRG

This section discusses the implementation of the higher-order masked lookup table scheme with increasing shares [CRZ18, Section 5] following a similar approach as discussed in Section 4. An advantage with the increasing shares approach is that it helps to bring down the total number of pseudorandom bits required by a factor of *two*. This was possible with the observation that only i random masks are sufficient to protect the randomised lookup table entry during the i-th shift, unlike $n-1$ masks as in the normal variant approach. Moreover, the reduction in overall randomness leads to a factor of *two* improvement in the execution time of the masked S-box implementation. Since the number of output masks increases gradually for increasing shares variant, the required number of pseudorandom values per shift also changes. Hence, the upper bound and the computation/recomputation of the unique index α has to be adjusted that will induce changes in the LR procedure accordingly.

5.1 Lookup Table Scheme with Increasing Shares using Robust PRG

Here we discuss the process of computing the masked lookup table with increasing shares using a strong $(r, t, 1)$-robust PRG, where we compute the random output masks on-the-fly.

The structure of the higher-order masked lookup table scheme presented in Algorithm 4 remains the same for increasing shares variant since the masked lookup table construction in Algorithm 4 is *independent* of the number of columns. Since the computation of index α depends on the number of pseudorandom values required per shift, we present the LR variant for increasing shares using a strong $(r, k, 1)$-robust PRG in Algorithm 7. Therefore, the only change in Algorithm 4 is to replace the sub-routine call to Algorithm 5 by Algorithm 7.

Algorithm 7: *LR-i-r*: LR procedure for increasing shares variant using the robust PRG.

```
Input  : $y_1, i, u, x_i$.
// $y_1$ represents $T(u)$ after the $i^{\text{th}}$ shift by $x_i$.
Output : $z_1, \ldots, z_n$.
$z_1 \longleftarrow y_1$
for $j \leftarrow 2$ to $i+1$ do
    $s^* \longleftarrow 0$
    // recomputation of the $(i-1)^{\text{th}}$ shift randoms.
    if $((2 \leq i \leq n-1)$ and $j < (i+1))$ then
        $\alpha \longleftarrow \Big(\big((i-2)\cdot(i-1)\cdot 2^{k-1}\big) + \big((u \oplus x_i)\cdot(i-2)\big)\Big) + (j-1)$
        // $j < (i+1)$ excludes last index from the recomputation
        $s^* \overset{\mathbb{F}_{2^k}}{\longleftarrow} PRG_r(\alpha)$
        // output of the strong $(r, t, 1)$-robust PRG at $\alpha$
    end
    $\alpha \longleftarrow \Big(\big((i-1)\cdot i \cdot 2^{k-1}\big) + \big(u\cdot(i-1)\big)\Big) + (j-1)$
    $s \overset{\mathbb{F}_{2^k}}{\longleftarrow} PRG_r(\alpha)$
    $z_j \longleftarrow s$
    $z_1 \longleftarrow z_1 \oplus (s^* \oplus z_j)$
end
return $z_1, \ldots, z_n$
```

The total number of pseudorandom bits required for the increasing shares variant is given by:

$$k' \cdot \big(1 \cdot 2^k + 2 \cdot 2^k + \ldots + (n-1)\cdot 2^k\big) = k' \cdot 2^{k-1} \cdot n \cdot (n-1).$$

To determine the parameters for the strong $(r, k, 1)$-robust PRG, first, we need to compute the *locality* of the implementation using increasing shares. Even though the number of output masks are gradually increasing, it turns out that the *locality* of the implementation remains the same i.e. $2 \cdot (n-1)$. This is due to the fact that after the shift by x_{n-1}, the table still depends on $n-1$ input shares and $n-1$ output masks.

Lemma 6. *The randomness locality of the (k, k') S-box gadget in Algorithm 4 locality refreshed using Algorithm 7 is $k' \cdot 2 \cdot (n-1)$ bits.*

Proof. The proof provided for Lemma 4 can be extended to increasing shares with the only change that the lookup table depends on i random masks instead of $n-1$ masks after shift and LR by x_i. Therefore, by the end of shift by x_{n-1}, the first row of table T depends on $n-1$ random output masks from LR along with $n-1$ input shares, each of size k' bits. Therefore, the randomness locality of Algorithm 4 locality refreshed using Algorithm 7 is $k' \cdot (n-1+n-1) = k' \cdot 2 \cdot (n-1)$ bits. □

Due to the same value of *locality*, the parameter r, the number of true random input bits to PRG and the memory complexity remains unchanged. Hence, the memory complexity is $\beta \cdot k' \cdot (n-1)^2 \cdot n = \tilde{O}(k' \cdot k \cdot n^3)$. Even though the time required to generate a pseudorandom value remains $\tilde{O}(k' \cdot k \cdot n^3)$, there is a factor of *two* reduction in the total number of pseudorandom values required for the increasing shares implementation. Effectively, this improves the overall execution time by ($\approx$) a factor of *two* when compared to the implementation using the approach presented in Section 4. The changes in the unique index α computation using the inputs to LR are summarised in Table 3.

Theorem 7. *The masked higher-order lookup table-based S-box computation in Algorithm 4 locality refreshed using Algorithm 7 (with the strong $(r, t, 1)$-robust PRG) where the adversary observe t_1 intermediate variables and t_o output variables such that $t_1 + t_o \leq t < n$ is t-SNI secure.*

The security proof of *increasing shares with robust PRG* is almost similar to Theorem 3 except for the difference in the simulation of LR. The LR procedure for increasing shares recomputes i random masks from the shift by x_i (in steps 4 - 7 of Algorithm 7) and produces an $i+1$ re-sharing as the output. Essentially, invoke the procedure with the last input share being zero. Since there is an increment of column index by *one*, the number of random values used to mask the randomised lookup table is incremented by one at a time. So, we can not use Lemma 2 as it is to show the simulation of output variables of LR. Hence, we need to prove that the modified LR still respects the t-NI security definition. Precisely, add i to I when all $(i+1) < n$ output shares of the LR after shift by x_i is observed.

We provide the full security proof in Appendix C.1.

Increasing Shares Construction Using Multiple PRGs. Unlike the implementation in Subsection 5.1, this approach reduces the total number of truly random input bits to PRG from $\beta \cdot k' \cdot (n-1)^3$ bits to $(\beta \cdot k' \cdot n \cdot (n-1)^2)/2$ bits. The formulas for the index computation/recomputation to suit the increasing shares variant along with the total memory complexity are summarised in Table 3. Algorithm 8 lists the steps for the LR procedure using multiple PRGs with the modified computation of index α for the increasing shares variant.

Theorem 8. *The masked higher-order lookup table-based S-box computation in Algorithm 4 that is locality refreshed using Algorithm 8 (using multiple PRGs each with a locality $l = k'$) where the adversary observe t_1 intermediate variables and t_o output variables and any set $R^* \subset [1, n \cdot (n-1)]$ such that $t_1 + t_o + |R^*| < n$ is t-SNI-R^* secure. Moreover, Moreover, all values from the randomness partition $A_z, z \in R^*$ can be perfectly simulated using at most $t_1 + |R^*|$ input shares of x.*

The security proof for this increasing shares approach using the multiple PRGs is similar to the proof presented in the Theorem 6 except the change that the LR procedure as explained in previous subsection. Accordingly, the way we construct the index set I will be modified. Also, we show the simulation in the extended model of security, t-SNI-R^*.

The detailed proof can be found in Appendix C.2.

6 Implementation

In this section, we present the implementation results of a masked software implementation of the full block ciphers AES-128 [FIP] and PRESENT 80-bit key variant [BKL+07] using the constructions described in the previous sections. We would like to mention that all our results are based on an 8-bit implementation. As explained in [CGZ20, Section 5], while implementing the full block cipher using PRG constructions discussed so far, we

Algorithm 8: *LR-i-m*: LR procedure for the increasing shares approach using the multiple PRGs.

```
Input  : $y_1, i, u, x_i$.
// $y_1$ represents $T(u)$ after the $i^{\text{th}}$ shift by $x_i$.
Output : $z_1, \ldots, z_n$.
$z_1 \longleftarrow y_1$
for $j \leftarrow 2$ to $i+1$ do
    $s^* \longleftarrow 0$
    // recomputation of the $(i-1)^{\text{th}}$ shift randoms.
    if $((2 \leq i \leq n-1)$ and $j < (i+1))$ then
        $ind \longleftarrow (i-2) \cdot (n-1) + (j-1)$
        $\alpha \longleftarrow u \oplus x_i$
        $s^* \xleftarrow{\mathbb{F}_{2^k}} PRG_m(ind, \alpha)$
        // output of $PRG_{ind}$ evaluated at $\alpha$
    end
    $ind \longleftarrow (i-1) \cdot (n-1) + (j-1)$
    $\alpha \longleftarrow u$
    $s \xleftarrow{\mathbb{F}_{2^k}} PRG_m(ind, \alpha)$
    $z_j \longleftarrow s$
    $z_1 \longleftarrow z_1 \oplus (s^* \oplus s)$
end
return $z_1, \ldots, z_n$
```

Table 3: Summary of the total number of bits of pseudorandom values the true random values, the number memory required per (k, k') S-box along with the formulas for the computation and the recomputation of index α for the increasing shares variant of Algorithms 7 and 8 using the robust PRG and the multiple PRG approaches, respectively. Here, i represents the shift index, u is the row index and j represents the column index.

Description	**Value**
# pseudorandom	$k' \cdot 2^{k-1} \cdot (n-1)^2$.
Algorithm 4 using strong robust PRG	
# input to PRG	$\beta \cdot k' \cdot 2 \cdot n \cdot (n-1)^2$.
Memory required	$k' \cdot (2 \cdot 2^k + \beta \cdot 2 \cdot n \cdot (n-1)^2)$.
Index α computation	$\big(i \cdot (i-1) \cdot 2^{k-1}\big) + \big(u \cdot (i-1)\big) + j - 1$.
Index α recomputation	$\big((i-1) \cdot (i-2) \cdot 2^{k-1}\big) + \big((u \oplus x_i) \cdot (i-2)\big) + j - 1$.
Algorithm 4 using multiple PRG	
# input to PRG	$\big(\beta k' \cdot n \cdot (n-1)^2\big)/2$.
Memory required	$k' \cdot (2 \cdot 2^k + \beta \cdot 2 \cdot (n-1)^2)$.
(index to PRG, α) computation	$\Big(\big((i \cdot (i-1)/2) + j - 1\big), u\Big)$.
(index to PRG, α) recomputation	$\Big(\big(((i-1) \cdot (i-2)/2) + j - 1\big), (u \oplus x_i)\Big)$.

Table 4: Summary of locality, the total number of true and pseudorandom bits required for a full block cipher execution of AES-128 using the robust and the multiple PRGs approaches. The values represent number of bits.

	Normal shares variant [CRZ18, Algorithm 1]	Increasing shares variant [CRZ18, Algorithm 3]
S-box	$(n-1)^2 \cdot 2^k$	$(n-1)^2 \cdot 2^{(k-1)}$
Robust PRG approach		
Locality	$2 \cdot (n-1) \cdot 8$	$2 \cdot (n-1) \cdot 8$
True rand	$48 \cdot n \cdot (n-1)^2$	$48 \cdot n \cdot (n-1)^2$
Multiple PRG approach		
Locality	8	8
PRG seed	$16 \cdot (n-1)^3$	$8n \cdot (n-1)^2$

need to calculate the total number of pseudorandom bits required for the full block cipher implementation. In the following Table 4, we indicate the *number bits* of pseudorandom and true randoms along with the overall locality of the AES-128 block cipher implementation. The first row of Table 4 corresponds to the total number of bits of randomness required for all the S-box function calls (10 rounds · 16 S-box/round=160 S-box calls) for a t-th order secure AES-128 implementation.

Since our aim is to design a masking scheme that is feasible for resource-constrained devices, we choose NXP-FRDM-k64F, an ultra-low-cost development platform as our target architecture. The FRDM-K64 is supported by various open source embedded operating systems. The microcontroller used in the development platform is MK64FN1M0VLL12, a low-power microcontroller based on ARM Cortex-M4 processor having a 256 KB RAM, 1 MB flash memory and a clock frequency of 120 MHz. This device using the in-built TRNG, requires (approximately) 300 clock cycles to generate a 32-bit random number.

Depending on the speed of the in-built TRNG of the target architecture, there is a trade-off between online execution time and generating the random values from PRG/TRNG. Since the chosen target architecture (NXP-FRDM-k64F) has a relatively faster TRNG, we followed a hybrid approach to generate the random values required for the implementation. To reduce the RAM to store the pre-processed tables, generate the random values during the pre-processing phase from PRGs. We made this choice since the *recomputation* of the random values (without explicitly storing them) is possible only with the PRG approach. Hence, we met the primary motivation of optimised RAM memory. Whereas to improve the online execution time of the block cipher implementation, generate the random values during the online phase from the in-built TRNG. After the table lookup using the final share $T(x_n)$, the random masks required for the final LR are generated from a TRNG. Precisely, the final output shares of S-box are masked without PRG. The major advantage of this approach is the random values from the randomised table pre-processing will not enter into the gadgets in the further linear layers of the block-cipher computation. Hence, there is no need to perform the LR after each *xor* operation. On the other hand, on a device having a relatively slower TRNG, one can choose to generate the entire set of random values from the PRG itself. So, the gadgets from the further linear layers of block-cipher have to be locality refreshed. Precisely, use a PRG to optimise the RAM and use a PRG/TRNG for a better online execution time. In either case, our techniques achieve a balance between RAM and online execution time.

To take the pre-processing advantage of lookup table-based masking schemes, pre-compute the lookup tables for all 160 (16/round · 10 rounds) and 496 (16/round · 31 rounds) S-box function calls for AES-128 and PRESENT block ciphers, respectively. This pre-computation happens independent of the secret x. We refer to this time required for pre-processing as the *offline* time. The rest of the computation of the block cipher happens after the availability of the secret. The time required for execution after the availability of secret is referred to as *online* time. Hence, the overall execution time is divided into offline and online time. Note that the offline calculations have to be repeated for every single execution of block cipher. Reusing the same set of pre-processed lookup tables leads to an insecure implementation since the random output masks across two independent block cipher executions remain the same.

AES-128 using the robust PRG approach: In order to implement AES-128 using a robust PRG, the first step is to initialise the input seed with true random values. The number of true random bits required for initialisation of the seed in turn depends on the locality of the circuit. As proved in Lemma 4, the locality of the S-box gadget described in Algorithm 4 is $2 \cdot (n-1) \cdot k'$. Since the random values generated from PRG are being used only in the pre-processing phase of the randomised lookup table construction, the locality of the overall implementation is nothing but the locality of the S-box gadget i.e. $2 \cdot (n-1) \cdot 8$ (for AES-128, $k' = 8$). Now the task is to identify the size of the finite field to obtain the random values mentioned in Row 3 of Table 4. Let R represent the total number of random values such that $R \leq \beta \cdot |\mathbb{F}_{2^{8 \cdot \beta}}|$. To observe the results of implementations secure against t-th order where $t \leq 10$, we have $\beta = 3$. Needless to say that each evaluation of polynomial over $\mathbb{F}_{2^{8 \cdot \beta}}$ generates $8 \cdot \beta$-bit random values. Since the robust PRG is constructed by combining outputs of $n = t + 1$ PRGs (refer Lemma 3), the number of bytes of true random inputs to robust PRG is given by:

$$\beta \cdot n \cdot r = 48 \cdot n \cdot (n-1)^2.$$

Along with the masked lookup table, in order to generate the outputs of PRG *on-the-fly*, we also need to store the inputs to PRG. Therefore, the number of bytes of memory required per S-box for the implementation of AES-128 using robust PRG is:

$$2 \cdot 8 \cdot 2^8 + 48 \cdot n \cdot (n-1)^2.$$

We summarise the values for concrete instantiations for $n = 3, 5$ in Table 5.

AES-128 using the multiple PRG approach: As discussed in Section 4.3, the locality of each of these multiple PRGs is *k'*. There is an independent PRG per column per shift. For the case of AES-128, each PRG has to generate $160 \cdot 2^8 = 40,960$ bytes of random values which implies $\beta = 2$. For the scheme presented in Subsection 4.3, at the end of shift by x_{n-1}, we need to store only true random inputs of $n-1$ PRGs. Precisely, the memory required in bytes is given by:

$$2 \cdot 2^8 + 16 \cdot n \cdot (n-1)^2.$$

We have built our code for the memory optimised variant of the masked implementation of AES-128 implemented using higher-order lookup table-based scheme on top of the publicly available masked implementations from [Cor]. We referred the publicly available unmasked implementation of PRESENT block cipher from [Klo]. Our implementation code is available at [VV]. For the experiments, we assume that the masked implementation will receive shares of the subkeys. However, the masked lookup table techniques can be extended to subbyte operation of key expansion. But, storing the pre-processed lookup tables of subbyte of key expansion requires an additional RAM memory. For instance, for AES-128, this requires an additional 10 KB ($40 \cdot 256$ bytes) of memory for subbyte of key expansion.

Table 5 lists the offline and the online execution times along with the true random values and the memory required for AES-128 implemented using the robust PRG technique

Table 5: Higher-order masked S-box computation using our variant of the table-based scheme with full pre-processing (Algorithm 4) using robust PRG technique for the *normal shares* (Algorithm 5) and the *increasing shares* variants (Algorithm 7). The true random PRG seed and the total memory required for AES-128 are given in bytes and KB, respectively. The offline, online and total execution are represented in millions of clock cycles.

n	PRG seed	Total Memory (KB)	Offline (M)	Online (M)	Total (M)
Normal shares variant using robust PRG approach					
3	72	40.14	2535.303	3.973	2539.276
5	480	40.94	66330.46	39.086	66369.546
Increasing shares using robust PRG approach					
3	72	40.14	2021.685	3.955	2025.64
5	480	40.94	46360.51	39.024	46399.534

for $n = 3, 5$. Similarly Table 6 lists the execution times and the corresponding values for multiple PRGs approach. We compared the implementation results of our approach with circuit-based implementation of AES-128 using [RP10] with *FullRefresh* [DDF14b] that is proven t-SNI secure in [BBD$^+$16]. Also, with masked bitsliced variant of AES-128 [RSD06, GR17] adapted to 8-bit implementation and the original implementation described in [GR17] along with the optimisation suggested for 32-bit architecture. It is evident from Table 7 that the online execution of our approach is approximately 2 and 1.5 times faster compared to [RP10] and the 8-bit bitsliced implementations, respectively. However, the online execution time of 16-bit bitsliced approach is slightly better than our approach, particularly at higher orders. The reason being the recomputation of random masks using polynomial-based PRG during the online phase is computationally heavy compared to the bitsliced implementation (see Remark 5). We also implemented our approach for light-weight block cipher PRESENT. The experimental results for the PRESENT block cipher implemented using the multiple PRGs technique are tabulated in Table 8 along with a comparison of the results with the circuit-based implementation of PRESENT using [CRV15].

Remark 5. We present the total amount of computation required to implement AES-128 using our approach and the bitsliced approach. Let us consider the lookup table-based implementation of the S-box. There are 16 S-box operations per round. Further, each S-box performs *recomputation* of t output masks using PRG. Inturn each output mask is an evaluation of a t degree polynomial over $\mathbb{F}_{2^{16}}$. We implement each field multiplication over $\mathbb{F}_{2^{16}}$ using 5 xors and 20 table lookups. Also, refresh the inputs to and outputs from S-box using LR. So, the total number of operations required per round is $16 \cdot t^2 \cdot (5\,xor + 20\,TL) + 32 \cdot t$. Even though the random masks during shift and LR by x_{n-1} are 8-bit values, as part of the *reccomputation*, we still need to compute the 16-bit output from the polynomial-based PRG evaluation over $\mathbb{F}_{2^{16}}$.

In the 16-bit bitsliced implementation of AES-128, there is one bitsliced S-box computation per round, and it involves 16 32-bit ISW-AND gates followed by fullrefresh, 84 XOR gates, and 8 move operations. Since our target device has a faster TRNG, the number of random values for fullrefresh will not impact the execution time of the bitsliced variant.

For a concrete instantiation of $t = 10$, the *online phase* of the lookup table computation per round requires: $\big($8000 xors, 32,000 table lookups$\big)$ for *recomputation of masks* and 320 xors per LR. Also, 320 bytes from TRNG. Similarly, for $t = 10$, the bitsliced computation per round needs: 3872 logical *and*, 7890 xors and 3520 bytes from TRNG.

Table 6: Higher-order masked S-box computation using our variant of the table-based scheme with full pre-processing (Algorithm 4) using the multiple PRGs technique for the *normal shares* (Algorithm 6) and the *increasing shares* variants (Algorithm 8). The true random PRG seed and the total memory required for AES-128 are given in bytes and KB, respectively. The offline, online and total execution are represented in millions of clock cycles.

n	PRG seed	Total Memory (KB)	Offline (M)	Online (M)	Total (M)
Normal shares variant using the multiple PRGs approach					
3	16	40.1	111.618	0.429	112.047
5	128	40.4	832.862	0.968	833.83
7	432	40.7	2675.283	1.786	2677.069
9	1024	41.3	6219.613	2.893	6222.506
11	2000	42.1	11945.748	4.238	11949.986
Increasing shares variant using the multiple PRGs approach					
3	12	40.1	72.59	0.423	73.013
5	80	40.5	465.558	0.968	466.526
7	252	41.4	1439.652	1.765	1441.417
9	576	40.8	3265.303	2.873	3268.176
11	1100	41.2	6267.878	4.197	6272.075

Table 7: Higher-order masked S-box computation using [RP10] with FullRefresh and 8-bit bitslicing. Also, 16-bit bitslicing using 32-bit ISW-AND [GR17]. The true randoms and the total memory required for AES-128 are given in bytes and KB, respectively. The total execution is represented in millions of clock cycles.

n	True rand	Memory (KB)	Total(M)	True rand	Memory (KB)	Total(M) 8-bit bitslicing	Total(M) 16-bit bitslicing
	[RP10]			Bitslicing			
3	960	0.02	1.612	20480	1.055	0.825	0.477
5	1920	0.03	2.61	40960	1.758	1.76	0.869
7	2880	0.04	4.078	61440	2.461	3.272	1.437
9	3840	0.05	6.027	81920	3.164	4.89	1.846
11	4800	0.06	8.458	102400	3.867	6.938	2.825

Table 8: Higher-order masked S-box computation using our variant of the table-based scheme with full pre-processing (Algorithm 4) using the multiple PRG technique for *increasing shares* variant (Algorithm 8) and the circuit-based implementation of PRESENT using [CRV15]. The first entry represents the number of input shares. The true random PRG seed and the total memory required for PRESENT are given in bytes and KB, respectively. The offline, online and total execution are represented in millions of clock cycles.

n	PRG seed	Total Memory (KB)	Offline (M)	Online (M)	Total (M)
Increasing shares variant using the multiple PRGs approach					
3	6	7.8	8.586	0.996	9.582
5	40	7.85	51.735	0.968	52.703
7	126	7.93	156.924	2.239	159.163
9	288	8.13	351.086	4.761	355.847
11	550	8.4	661.489	7.166	668.655
Circuit-based implementation using [CRV15]					
n	True rand	Total Memory (KB)	Offline (M)	Online (M)	Total (M)
3	1488	0.042	–	2.056	2.056
5	2976	0.063	–	3.173	3.173
7	4464	0.086	–	4.686	4.686
9	5952	0.11	–	6.755	6.755
11	7440	0.133	–	9.24	9.24

7 Conclusion

The amount of RAM memory required to store the pre-processed tables for a full block cipher execution makes the table-based schemes infeasible at higher orders. In our work, we demonstrate that the higher-order masked lookup table schemes with full pre-processing can be implemented on resource-constrained devices. Concretely, the amount of RAM memory required to implement the masked lookup table-based scheme from [CRZ18] is now essentially made independent of the masking order. It will be an interesting future work to achieve RAM compression for the masked higher-order lookup table-based implementation of block ciphers with modes of operation, instead of a single block cipher execution as done currently.

Acknowledgements

This work was funded by the INSPIRE Faculty Award (DST, Govt. of India) for Srinivas Vivek. We would like to thank anonymous reviewers for their valuable inputs.

References

[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.

[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.

[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.

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 342–375. Springer, 2020.

[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 April 15, 2021.

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

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic Decomposition for Probing Security. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Proc., Part I*, volume 9215 of *LNCS*, pages 742–763. Springer, 2015.

[CRV15] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast Evaluation of Polynomials over Binary Finite Fields and Application to Side-channel Countermeasures. *J. Cryptographic Engineering*, 5(2):73–83, 2015.

[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.

[DDF14a] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In Nguyen and Oswald [NO14], pages 423–440.

[DDF14b] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

[FH17] 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*. Springer, 2017.

[FIP] 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.

[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 Fischer and Homma [FH17], pages 154–171.

[GTP$^+$20] Zhipeng Guo, Ming Tang, Emmanuel Prouff, Maixing Luo, and Fei Yan. Table Recomputation-Based Higher-Order Masking Against Horizontal Attacks. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 39(1):34–44, 2020.

[IKL$^+$13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 576–588. Springer, 2013.

[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.

[JS17] Anthony Journault and François-Xavier Standaert. Very High Order Masking: Efficient Implementation and Security Evaluation. In Fischer and Homma [FH17], pages 623–643.

[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 April 15, 2021.

[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.

[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.

[RRST02] Josyula R. Rao, Pankaj Rohatgi, Helmut Scherzer, and Stephane Tinguely. Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 31–41. IEEE Computer Society, 2002.

[RSD06] Chester Rebeiro, A. David Selvakumar, and A. S. L. Devi. Bitslice implementation of AES. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *Cryptology and Network Security, 5th International Conference, CANS 2006, Suzhou, China, December 8-10, 2006, Proceedings*, volume 4301 of *Lecture Notes in Computer Science*, pages 203–212. Springer, 2006.

[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.

[Viv17] Srinivas Vivek. Revisiting a Masked Lookup-Table Compression Scheme. In Arpita Patra and Nigel P. Smart, editors, *Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2017, Proceedings*, volume 10698 of *Lecture Notes in Computer Science*, pages 369–383. 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 July 14, 2021.

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

[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 A Variant Scheme for Large Registers

In this section, we extend the scheme to improve the computations for devices with large registers, as previously done by [RDP08, Cor14, CRZ18]. The idea is to *pack*, say w S-box outputs each of $\{0,1\}^{k'}$-bits in a single m-bit register where $m = w \cdot k'$. Let us assume that the target architecture register size is always a power of 2. Therefore, a set of w values can be shifted at the same time which not only helps to reduce the size of lookup table but also improves the running time of the scheme. Formally, the packing proceeds as follows. Treat the k-bit input as:

$$k := a^{(1)} \parallel a^{(2)}, \tag{6}$$

where $a^{(1)} \in \{0,1\}^{k_1}$ and $a^{(2)} \in \{0,1\}^{k_2}$ such that $k = k_1 + k_2$ and $k_2 = \log_2 w$.

We group w values having the same most significant k_1-bits and $a^{(2)}$ ranges from $0 \leq a^{(2)} \leq w - 1$. For instance, consider packing the (8, 8)-bit AES S-box into a 32-bit register, we can pack $w = 32/8 = 4$ S-box values into a single 32-bit register. Essentially, this variant improves the computation speed by a factor of w since a set of w values each of k'-bits are processed in parallel. Precisely, let each row of T_1 contains a pack of w S-box outputs, such that each row of T_1 can be represented as:

$$T_1(u) \coloneqq \mathrm{S}(u \mid\mid 0)|| \ldots \mathrm{S}(u \mid\mid (w-1)), \tag{7}$$

where $u \in \{0,1\}^{k_1}$. This table T_1 is shifted using $x_i^{(1)}$ which represents k_1 bits of the i^{th} input share x_i. Similar steps as in Algorithm 4 are followed except the fact that only k_1-bits of the input shares are used for shifting, and T_1 is shifted by $x_1^{(1)}, \ldots, x_{n-1}^{(1)}$. But there is a corresponding change in the way LR works. We need to sample an m-bit random value instead of a k'-bit value (see Remark 6). The rows of the table T_1 are masked using the m-bit words. The steps of this process are described in Algorithm 9. We present the randomised lookup table construction using the robust PRG approach. Similar steps can be followed in the multiple PRGs approach as well.

Remark 6. One way of generating the m-bit value is to pass the value of $u, u \in \{0,1\}^{k_1}$, to the robust PRG construction. Then it generates the values for $u||z$, $\forall z \in \{0,1\}^{k_2}$. Then, concatenate these values generated to form a m-bit value as the output. But, we need k'-bit random values during the construction of T_2. Hence, the input parameters to the robust PRG construction inside T_2 to be passed in such a way that the value of the unique index α needs to be distinct from the values generated during T_1 construction. We need to follow similar steps for the multiple PRGs construction as well.

After the construction of T_1 as explained in Algorithm 9, the most significant k_1-bits of the final share is used to lookup T_1 that returns an m-bit value:

$$\mathrm{LR}(T_1(x_n^{(1)}), x_n^{(1)}) = y_{m_1}, \ldots y_{m_n}, \tag{8}$$

where $y_{m_i} \in \{0,1\}^m$ and

$$y_{m_n} = \mathrm{S}(x^{(1)} \mid\mid 0^{k_2})|| \ldots || \mathrm{S}(u \mid\mid 1^{k_2}) \oplus y_{m_1} \oplus \cdots \oplus y_{m_{n-1}}. \tag{9}$$

Since each of the values in the n-dimensional vector obtained in Equation 7 is a concatenation of w words, parsing the values to obtain the k'-bits at position $x^{(2)}$ would result in the shared evaluation of $\mathrm{S}(x)$. But, we cannot use $x^{(2)}$ as it is since it is going to leak k_2-bits of the secret x. Therefore, we construct another Table T_2 of w rows each of size k'-bits.

The constructions of T_2 proceeds as follows. The vector obtained in Equation 7 is processed and the w words of k'-bits are extracted into w rows of T_2. Then, the same procedure of shift by $x_1^{(2)}, \ldots, x_{n-1}^{(2)}$, each shift followed by a refresh is repeated. Finally, a table lookup of $T_2(x_n^{(2)})$ is going to return the shared evaluation of $\mathrm{S}(x)$. This procedure is

Algorithm 9: Construction of T_1 for larger register variant using *robust* PRG.

Input :

- x_i, $1 \le i \le n \in \{0,1\}^k$.
- Register size, m.
- An (k, k') S-box lookup table.

Output : Table T_1.

```
w ⟵ $m/k'$
$k_2 \longleftarrow \log_2 w$
$k_1 \longleftarrow k - k_2$ for $u \leftarrow 0$ to $2^{k_1} - 1$ do
    $T_1(u) \longleftarrow \mathrm{S}(u \mid\mid 0) \mid\mid \dots \mathrm{S}(u \mid\mid (w-1))$
end
for $i \leftarrow 1$ to $n-1$ do
    for $u \leftarrow 0$ to $2^{k_1} - 1$ do
        $T_{\mathrm{aux}}(u) \longleftarrow T_1(u \oplus x_i^{(1)})$
    end
    for $u \leftarrow 0$ to $2^{k_1} - 1$ do
        $(t_1, \dots, t_n) \longleftarrow (T_{\mathrm{aux}}(u), 0, \dots, 0)$
        foreach $i > 1$ do
            for $j \leftarrow 2$ to $n$ do
                $t_j \longleftarrow \mathrm{PRG}_m(i-1, j, u \oplus x_i^{(1)})$ // PRG
                 returns an $m$-bit random value (see
                 Remark )
            end
        end
        $(t_1, \dots, t_n) \longleftarrow \mathrm{LR}_m(t_1, \dots, t_n)$
        $T_1(u) \longleftarrow t_1$
    end
end
```

summarised in Algorithm 10. One subtle point to observe here is that, unlike the scheme presented in Algorithm 4, it is not possible to preprocess the entire table offline. Hence, compute tables T_1 and T_2 during offline and online, respectively.

Since this approach for larger register variant require the construction of two tables T_1 and T_2 where the amount of randomness required for T_1 is the same as Algorithm 4 and T_2 requires additional random values to mask, the total number of k'-bit random values required are:

$$(n-1) \cdot \left(2^k \cdot (n-1) + 1\right) + (n-1) \cdot \left(2^{k_2} \cdot (n-1) + 1\right). \tag{10}$$

Algorithm 10: Larger register variant using *robust* PRG (continued).

Input :

- x_i, $1 \leq i \leq n \in \{0,1\}^k$.
- Register size, m.
- Table T_1.

Output : $y_i, 1 \leq i \leq n{-}1$ and $y_n := \mathrm{S}(x) \oplus y_1 \oplus \ldots y_{n-1} \in \{0,1\}^{k'}$.

```
$w \longleftarrow m/k'$
$k_2 \longleftarrow \log_2 w$
$k_1 \longleftarrow k - k_2$
$t_{m_1} = \mathrm{LR}(T_1(x_n^{(1)}))$
for $j \leftarrow 2$ to $n$ do
    $t_{m_j} \longleftarrow \mathrm{PRG}(n-1, j, x_n^{(1)})$
end
$y_{m_1}, \ldots y_{m_n} = \mathrm{LR}(t_{m_1}, \ldots, t_{m_n})$
for $u \leftarrow 0$ to $2^{k_2} - 1$ do
    $T_2(u) \longleftarrow \mathrm{extract}(y_{m_1}, u+1)$
     // extract the $u + 1^{\mathrm{th}} k'$−bit chunk from $m$-bit
     register
end
for $i \leftarrow 1$ to $n-1$ do
    for $u \leftarrow 0$ to $2^{k_2} - 1$ do
        $T_{\mathrm{aux}}(u) \longleftarrow T_2(u \oplus x_i^{(2)})$
    end
    for $u \leftarrow 0$ to $2^{k_2} - 1$ do
        if $i == 1$ then
            for $j \leftarrow 2$ to $n$ do
                $t_j \longleftarrow \mathrm{extract}(y_{m_j}, u+1)$
            end
        end
        foreach $i > 1$ do
            for $j \leftarrow 2$ to $n$ do
                $t_j \longleftarrow \mathrm{PRG}(i-1, j, u \oplus x_i^{(2)})$
            end
        end
        $t_1, \ldots, t_n \longleftarrow \mathrm{LR}(t_1, \ldots, t_n)$
        $T_2(u) \longleftarrow t_1$
    end
end
$t_1 \longleftarrow T_2(x_n^{(2)})$
for $j \leftarrow 2$ to $n$ do
    $t_j \longleftarrow \mathrm{PRG}(n-1, j, x_n^{(2)})$
end
$y_1, \ldots, y_n = \mathrm{LR}(t_1, \ldots, t_n)$
```

B Security Proof of Normal Variant

B.1 Proof of Theorem 3 (Normal variant with robust PRG)

Proof. The security proof of Theorem 3 heavily depends on the proof strategies from [CRZ18, CGZ20]. The evident reason being the two modifications to the original lookup table-based construction from [CRZ18]. The first one is the replacement of *refreshmasks* with *LR* and the second change is w.r.t. the way of generating the random values. Concretely, generate the random values from a strong $(r, t, 1)$-robust PRG construction instead of a TRNG. We have proved the Lemmas 1 and 2 in Section 3 which follows the same security properties of *refreshmasks* to show that the higher-order lookup table-based scheme remains t-SNI secure, after the replacement.

Before proceeding with the t-SNI proof of Theorem 3 to demonstrate the replacement of a TRNG with a a strong $(r, t, 1)$-robust PRG, let us recollect the steps of Theorem 2. This theorem is instrumental in demonstrating that the number of input bits required for the simulation of the robust PRG construction is always bounded by the number of true random inputs to robust PRG (represented by the parameter r).

Recap of proof of Theorem 2. We recall the proof of Theorem 2 to illustrate the $(r, t, 1)$-robust PRG construction is t-probing secure. As mentioned in Section 4, for the robust PRG implemented using trivial construction (see Lemma 3), the parameter

$$q = 1.$$

Moreover, each of the $t + 1$ PRGs of the trivial construction are polynomials with r coefficients evaluated over a field $\mathbb{F}_{2^\beta}$. So, each input/output of the polynomial-based PRG is a β-bit value.

Let the adversary put $t^{PRG} \leq t$ probes in the robust PRG circuit and the remaining $(t - t^{PRG})$ probes in the lookup table computation. We recollect the steps of Theorem 2 to show the simulation of the $t^{PRG} \leq t$ internal observations in the PRG circuit requires at most $\beta \cdot t^{PRG}$ output bits of the robust PRG and any other subset of $r - (\beta \cdot t^{PRG})$ output bits can be assigned uniform random values.

Using Lemma 5, the number of output bits of G that can be assigned true random values is given by

$$r - q \cdot \beta \cdot t^{PRG} = (t \cdot max(q \cdot \beta, l)) - \beta \cdot t^{PRG}. \tag{11}$$

Moreover, the simulation of t^{PRG} variables require $(t^{PRG} \cdot \beta)$ random bits. The remaining $(t - t^{PRG})$ probes in the randomised lookup table computation depends on $(t - t^{PRG}) \cdot l$ bits of randomness. According to Lemma 4, the locality of the (k, k') S-box gadget is $k' \cdot 2 \cdot (n - 1)$ bits. Hence, we need the following inequality to hold:

$$\begin{aligned}(t - t^{PRG}) \cdot l \leq & (r - t^{PRG} \cdot \beta) \\ \leq & (t \cdot max(\beta, l) - t^{PRG} \cdot \beta).\end{aligned}$$

We have two cases here.

Case I:($max(\beta, l) = \beta$): note that both β, l are positive integers. In this case,

$$\begin{aligned}(t - t^{PRG}) \cdot l \leq & (t \cdot \beta - t^{PRG} \cdot \beta) \\ (t - t^{PRG}) \cdot l \leq & \beta \cdot (t - t^{PRG}) \\ l \leq & \beta.\end{aligned}$$

Case II:($max(\beta, l) = l$): in this case,

$$\begin{aligned}(t - t^{PRG}) \cdot l \leq& (t \cdot max(\beta, l) - t^{PRG} \cdot \beta)\\ (t - t^{PRG}) \cdot l \leq& (t \cdot l - t^{PRG} \cdot \beta)\\ (t \cdot l - t^{PRG} \cdot l) \leq& (t \cdot l - t^{PRG} \cdot \beta)\\ t^{PRG} \cdot \beta \leq& t^{PRG} \cdot l\\ \beta \leq& l.\end{aligned}$$

The t^{PRG} variables in G are simulated as follows: let $R = \{A \cup B, |A \cup B| \leq r\}$. There exist a set $B, |B| \leq \beta \cdot t^{PRG}$ whereas the set A is any subset of $(r - \beta \cdot |A|)$ output bits of G such that $R \leftarrow \{0,1\}^{|A \cup B|}$ is assigned uniform and random bits. This is possible since $|R| \leq r$. Since the PRG construction is linear, it is possible to simulate the t^{PRG} observations using only the output bits from the set B. Hence, using at most $(t - t^{PRG}) \cdot l$ random bits we need to show the simulation of the observed variables in the higher-order lookup table implementation.

Simulation of variables from Algorithm 4 refreshed using *LR-n-r*. As explained in Case I and Case II above, the internal probes of PRG along with the probes in the lookup table scheme can be simulated using r random bits. For simplicity, we show the proof when $t^{PRG} = 0$, i.e., the adversary using all t probes in the lookup table circuit. The proof still holds for any $t^{PRG} > 0$. We now proceed to show the simulation using at most r random bits. We follow an ISW way of simulation to prove Theorem 3. Initially, we construct an index set I to hold the set of input shares depending on the observations made by the adversary. Let the adversary observe the input/intermediate variables using t_1 probes and put t_o probes on output variables such that

$$t_1 + t_o \leq t < n.$$

To prove that the S-box gadget is t-SNI, we need to show that the variables (including input/intermediate/output variables) observed using t probes can be simulated using at most t_1 input shares, $|I| \leq t_1$. Since the table T is shifted and refreshed $n-1$ times, we use $T_i, T_{aux,i}$ to represent the lookup and the auxiliary tables, respectively. Let SLR_i represent the sequence of shift by x_i and the subsequent LR operation. Also, we maintain a list of table indices probed across shifts i, $1 \leq i \leq n-1$. Let $V = \{T_i(u,j)$, $1 \leq i \leq n-1, u \in \{0,1\}^k, 1 \leq j \leq n\}$, $|V| \leq t_1$, where $T_i(u,j)$ represents the j^{th} element in u^{th} vector of table T during i^{th} shift. As mentioned in Remark 7, we will show the simulation of the observed table entries. We will not show the simulation of the entire column as in the t-SNI proof from [CRZ18].

We construct a set of input share indices required to show the simulation of the t observations. As described above, the steps involved are $SLR_i, 1 \leq i \leq n-1$ and final table lookup. The construction of I proceeds as follows:

1. Initialise $I = \{\}$.
2. Add i to I if x_i or $u \oplus x_i$ $(1 \leq i \leq n)$ are probed.
3. Add i to I if any of the intermediate variables of the form $T_i(u,j)$, $T_{aux,i}(u,j)$ or $T_i(u \oplus x_i, j)$ where $1 \leq i \leq n-1$ are probed.
4. Add n to I if $T_{n-1}(x_n, j)$ is probed.

5. Add i to I if the internal variables $y_{1,1}$ or $y_{1,j}$, $2 \leq j \leq n-1$ of SLR_i are probed.

6. No index is added to I when probing the outputs of SLR_i. The same holds for the final sharing of $\mathrm{S}(x)$ in Step 12 of Algorithm 4.

Since we are adding at most one index per internal probe and no index is added to I while probing the output shares, $|I| \leq t_1$. The simulation of the probed variables using the input share indices from I proceeds as follows. We prove the simulation using induction. Assume that SLR_i receives the simulated inputs.

1. Needless to say, the simulation of all the loop variables u, i and public values of $\mathrm{S}(x)$ requires no knowledge of input shares.

2. **Base case**: the induction hypothesis is true for $i = 1$ since $T_1(u) = \mathrm{S}(u)$.

3. We show the simulation of variables observed during SLR_i operation is divided further into two subcases.

   - $i \notin I$**:** according to the construction of I, the only possibility is by probing the outputs of LR. Using Lemma 1, any $n-1$ outputs can be assigned uniform random values without knowledge of x_i. In particular, we can assign the outputs of LR $y_i, 1 \leq i \leq n-1$ uniform random values since $t \cdot l \leq r$. The same argument holds for simulating the final outputs of $\mathrm{S}(x)$ after the final LR.
   - $i \in I$**:** since SLR_i receives simulated inputs, we can use x_i to simulate any intermediate variable observed including $T_i(u \oplus x_i, j)$, $T_{aux,i}(u, j)$ in Step 6. These values will be the inputs to Step 9 of of Algorithm 4, *LR-n-r* (see Algorithm 5). Now we move on to the simulation of the probed variables of *LR-n-r*. As mentioned in the proof of Lemma 2, for every $1 \leq j \leq n$, if the adversary probes j^{th} input to LR, then we need to simulate the random value s_j as well. Assign a uniform random value to s_j. Hence, using the inputs to LR and the random values s_j, we can simulate the intermediate variables $y_{1,1}$ or $y_{1,j}$ or the output shares y_j of LR.

4. It can be observed that the table entries added to set V are nothing but the output shares from the LR, which we show the simulation in Step 3.

5. In either case, we show the simulation of input/intermediate/output variables using the input share indices from the index set I. Hence, the induction hypothesis holds true for SLR_{i+1} as well.

6. The index set I would contain the index n when observing the final lookup or internal variables of LR in Step 12 of Algorithm 4. Hence, the simulation of these observations using x_n is similar to Step 3.

This concludes the t-SNI security proof of Theorem 3.

□

Remark 7. In the t-SNI security proof from [CRZ18], the whole table column is treated as a single share and simulated. This approach would have followed in [CRZ18] to view the table in each shift as a gadget for the ease of security proof. In our security proof, we only show the simulation of the individual entries of the table (not the entire column) due to the following. Since the random values from the same column having two distinct indices $T(u, j)$ and $T(u+1, j)$ are never combined as part of the scheme, the locality of the robust PRG is $l = k' \cdot 2 \cdot (n-1)$ bits. Hence, it is possible to assign uniform random values to at most $t \cdot l$ bits. To show the simulation of the entire column, we would need a

PRG with 2^k-independence that requires 2^k coefficients from TRNG, which would increase the memory/randomness complexity of the proposed scheme. Hence, we opt to provide iterative proof rather than recursive proof. Accordingly, we consider the entire S-box as a single gadget.

B.2 Proof of Theorem 5 (t-SNI-R^* Gadget composition)

Proof. Consider M gadgets $G_1, \ldots G_M$ that are t-SNI-R^* secure. Arrange these gadgets in a directed acyclic graph and organise the nodes in the graph in reverse topological sort ordering. Let each gadget G_i have $t_{1,i}$ input/intermediate probes in G_i such that $t_{1,1} + \cdots + t_{1,M} \leq t_1$. The randomness partitions for gadget G_i are denoted by $A_j^{(i)}$ for $1 \leq j \leq \gamma$. We use induction on i to prove that the composed gadget is t-SNI-R^*.

For the base case $|M| = 1$, the composed gadget has a single gadget. Since the gadgets G_i are t-SNI-R^*, we can infer that the final gadget is also t-SNI-R^*. Now consider the induction hypothesis that the composition of l gadgets $G_1, \ldots, G_l$ is t-SNI-R^* holds true for $i = l$. Now we need to prove that the composed gadget remains t-SNI-R^* for $i = l + 1$.

For the case $i = l$, since the composition of $G_1, \ldots, G_l$ gadgets is t-SNI-R^*, all intermediate variables $t_{1,1} + \cdots + t_{1,l} \leq t$, output shares O^* and any subset of indices $R_l^* \subset [1, \gamma]$ along with random bits $\bigcup\limits_{i=1}^{l} A_j^{(i)}, j \in R_l^*$ can be simulated using $|I_l \cup R_l^*|$ input shares of x. Since the gadgets are arranged in reverse topological sort, the outputs of G_{l+1} are the inputs to the composed gadget $G_1, \ldots, G_l$. Since each individual gadget is t-SNI-R^*, the intermediate variables $t_{l+1,1} \leq t$, output shares O^* and any subset of indices $R_{l+1}^* \subset [1, \gamma]$ along with random bits $\bigcup\limits_{i=1}^{l} A_j^{(l+1)}$ can be simulated using the input shares from the set $I' \cup R_{l+1}^* \cup (I_l \cup R_l^*)$ where $|I'| \leq t_{1,l+1}$. The t-SNI-R^* property holds for any subset $R^* \subset [1, \gamma]$. Hence, in the composed gadget obtained using $G_1, \ldots, G_{l+1}$, the intermediate variables $t_{1,1} + \cdots + t_{1,l+1}$, output shares O^* and any subset of indices $R^* \subset [1, \gamma]$ along with random bits $\bigcup\limits_{i=1}^{l+1} A_j^{(i)}, j \in R_l^*$ can be simulated using the input shares x_i where $i \in (I_{l+1} \cup R_{l+1}^*)$ and $I_{l+1} = I' \cup I_l$. We have,

$$|I_{l+1}| = |I' \cup I_l| \leq |I'| + |I_l| \leq t_{1,l+1} + (t_{1,1} + \cdots + t_{1,l}) \leq t,$$

which proves that the composed gadget of $G_1, \ldots, G_{l+1}$ is t-SNI-R^*. This even holds when $l + 1 = m = |M|$, which terminates the proof.

□

B.3 Proof of Theorem 6 (Normal Variant with Multiple PRGs)

The security proof of Theorem 6 is similar to the one presented in Subsection B.1. One immediate difference being the extended security model t-SNI-R^* where we need to show the simulation of the entire randomness partition by even probing a single value from this partition. Moreover, unlike a robust PRG, the adversary has no advantage in probing the internal wires of non-robust PRG implementation. The index set construction of Theorem 6 is the same as that of Theorem 3. We still provide the details for completeness.

Proof. We follow an ISW way of simulation to prove Theorem 6. Initially, we construct an index set I to hold the set of input shares depending on the observations made by the adversary. Let the adversary observe the input/intermediate variables using t_1 probes and put t_o probes on output variables such that

$$t_1 + t_o \leq t < n.$$

To prove that the S-box gadget is t-SNI-R^* (see definition 9), we need to show that the variables (including input/intermediate variables) observed using t_1 probes and O^* output shares can be simulated using at most $t_1 + |R^*|$ input shares, $|I| \leq t_1$. Since the table T is shifted and refreshed $n-1$ times, we use $T_i, T_{aux,i}$ to represent the lookup and the auxiliary tables, respectively. Let SLR_i represent the sequence of shift by x_i and the subsequent LR operation. Also, we maintain a list of table indices probed across shifts i, $1 \leq i \leq n-1$. Let $V = \{T_i(u,j),\, 1 \leq i \leq n-1, u \in \{0,1\}^k, 1 \leq j \leq n\}$, $|V| \leq t_1$, where $T_i(u,j)$ represents the j^{th} element in u^{th} vector of table T during i^{th} shift. As mentioned in Remark 7, we will show the simulation of the observed table entries. We will not show the simulation of the entire column as in the case of the proof from [CRZ18].

We construct a set of input share indices required to show the simulation of the t observations. As described, the steps involved are $SLR_i, 1 \leq i \leq n-1$ and final table lookup. Let I represent the set of input share indices and the construction of I proceeds as follows:

1. Initialise $I = \{\}$.
2. Add i to I if x_i or $u \oplus x_i$ $(1 \leq i \leq n)$ are probed.
3. Add i to I if any of the intermediate variables of the form $T_i(u,j)$, $T_{aux,i}(u,j)$ or $T_i(u \oplus x_i, j)$ where $1 \leq i \leq n-1$ are probed.
4. Add n to I if $T_{n-1}(x_n, j)$ is probed.
5. Add i to I if the internal variables s_j or $y_{1,1}$ or $y_{1,j}$, $2 \leq j \leq n$ of SLR_i are probed.
6. No index is added to I while probing the outputs of SLR_i. The same holds for the final sharing of $\mathrm{S}(x)$ in Step 12 of Algorithm 4.

Since we are adding at most one index per internal probe and no index is added to I while probing the output shares, $|I| \leq t_1$.

In the first step, we show the simulation of the partitions $A_j, j \in R^*$.

- When the internal variables s_j or the outputs, y_j of the LR or the output shares $T_i(j)$ of SLR_i with indices j, $2 \leq j \leq n$ are probed, then we need to simulate the corresponding A_{ind} where $ind = \big(i \cdot (n-1) + j - 1\big)$.
- Also, if the final shares $y_j, 2 \leq j \leq n$ of $\mathrm{S}(x)$ are probed, then $ind = n \cdot (n-1) + j - 1$.
- As explained in Section 4.3, each of these subsets are output pseudorandom values of size 2^k from a non-robust linear lt-wise independent PRG with $l = 1$. Assign uniform random values to t outputs of the non-robust PRG having the index ind.
- Since the polynomial-based PRG construction is linear and since only lt outputs are assigned, sample the PRG input seed following the same distribution as the output values.
- Then, using the sampled input seed, compute the remaining outputs of the partition A_{ind}.

Once we show the simulation of $A_{ind}, ind \in R^*$, we proceed with the simulation of SLR_i. Now we continue with the observed variables. Assume that SLR_i receives the simulated inputs.

1. Needless to say, simulation of the loop variables u, i and public values of $\mathrm{S}(x)$ requires no knowledge of input shares.

2. **Base case**: the induction hypothesis is true for $i = 1$ since $T_1(u) = \mathrm{S}(u)$.

3. We now show the simulation of variables observed during SLR_i operation.

    - $i \notin I$**:** according to the construction of I, the only possibility is by probing the outputs of LR. If the probed outputs of LR are $y_j, 2 \leq j \leq n-1$, then assign the value from the corresponding partition $A_{ind}, ind \in R^*$ which is already simulated. If the probed output is first share y_1, then there exist at least one unprobed $y_{j^*}, 2 \leq j^* \leq n$ which acts as a one-time pad. Hence, assign a uniform random value to y_1. Follow the exact steps to simulate the output shares after the final LR.
    - $i \in I$**:** since SLR_i receives simulated inputs, we can use x_i to simulate any intermediate variable observed including $T_i(u \oplus x_i, j)$, $T_{aux,i}(u, j)$ in Step 6. These values will be the inputs to Step 9 of of Algorithm 4, *LR-n-m* (see Algorithm 6). Now we move on to the simulation of the probed variables of *LR-n-m*. As explained in the previous step, assign the values to s_i from the partition A_{ind}. Using the simulated random masks s_j and input share x_i, compute the intermediate variables $y_{1,j}$ or the output shares y_j of SLR_i.

4. It can be observed that the table entries added to set V are nothing but the output shares from LR, which we show the simulation in Step 3.

5. In either case, we show the simulation of input/intermediate/output variables using the input share indices added to index set I. Hence, the induction hypothesis holds true for SLR_{i+1} as well.

6. The index set I would contain the index n while observing the final lookup or internal variables of LR in Step 12. Hence, simulate the observations using x_n as explained above.

Thus, we can conclude the t-SNI security proof of Theorem 6.

□

C Increasing Shares Variant Proofs

C.1 Proof of Theorem 7 (Increasing shares using Robust PRG)

As mentioned in the security proof of Theorem 3, simulate the internal observations of the PRG circuit. Hence, we proceed with the t-SNI security proof of increasing shares variant in Algorithm 4 locality refreshed using the procedure *LR-n-i* presented in Algorithm 7. Let SLR_i represent the sequence of shift by x_i and the subsequent LR operation.

The proof for higher-order lookup table-based construction using robust PRG is similar to the Theorem 3, except for the following differences. We use the Lemma 1 to assign uniform random values to output shares when the intermediate variables of LR remain unprobed. But, the same can not be applied directly in case of increasing shares variant. The reason being during the i-th shift and LR, the output table has $i+1 < n$ shares for every $i < n-1$. So, the adversary can observe all $(i+1) < n$ shares, in which case it is not possible to simulate the output shares with the knowledge of the input share x_i. Hence, we need to add i to I when all $(i+1) < n$ output shares of SLR_i are observed. But, this is not going to affect the security of our construction. Since there a total of n LRs and one of the LR remains unprobed, which still obeys the t-SNI condition $|I| \leq t_1$. Moreover, we can still use Lemma 1 for the LR after $(n-1)$-th shift and the final LR as long as the intermediate variables are unprobed. This is possible since the output shares for these cases are equal to $n > t$. The rest of the proof, along with the simulation of the probed observations, remains the same as Theorem 3. We provide detailed security proof for completeness.

Proof. We follow an ISW way of simulation to prove Theorem 7. Initially, we construct an index set I to hold the set of input shares based on the observations made by the adversary. Let the adversary observe the input/intermediate variables using t_1 probes and put t_o probes on output variables such that

$$t_1 + t_o \le t < n.$$

To prove that the S-box gadget is t-SNI, we need to show that the simulation of variables (including input/intermediate/output variables) using t probes requires at most t_1 input shares. Since the table T is shifted and refreshed $n-1$ times, we use $T_i, T_{aux,i}$ to represent the lookup and the auxiliary tables, respectively. Let SLR_i represent the sequence of shift by x_i and the subsequent LR operation. Also, we maintain a list of table indices probed across shifts i, $1 \le i \le n-1$. Let $V = \{T_i(u,j),\ 1 \le i \le n-1, u \in \{0,1\}^k, 1 \le j \le n\}$, $|V| \le t_1$, where $T_i(u,j)$ represents the j^{th} element in u^{th} vector of table T during i^{th} shift.

We construct a set of input share indices required to show the simulation of the t observations. As described, the steps involved are $SLR_i, 1 \le i \le n-1$ and final table lookup. Let I represent the set of input share indices and the construction of I proceeds as follows:

1. Initialise $I = \{\}$.
2. Add i to I if x_i or $u \oplus x_i$ $(1 \le i \le n)$ are probed.
3. Add i to I if any of the intermediate variables of the form $T_i(u,j)$, $T_{aux,i}(u,j)$ or $T_i(u \oplus x_i, j)$ where $1 \le i \le n-1$ are probed.
4. Add n to I if $T_{n-1}(x_n, j)$ is probed.
5. Add i to I if the internal variables y_1 or $y_{1,j}$, $2 \le j \le n$ of SLR_i are probed.
6. Add i to I only if all the $i+1$ outputs of $SLR_i, 1 \le i \le n-2$ are probed.
7. No index is added to I while probing the outputs of SLR_{n-1}. The same argument holds for the final sharing of $\mathrm{S}(x)$ in Step 12 of Algorithm 4.

Since we are adding at most one index per internal probe and no index is added to I while probing the output shares, $|I| \le t_1$. The simulation of the probed variables using the input share indices from I proceeds as follows. We prove the simulation using induction. Assume that SLR_i receives the simulated inputs.

1. Needless to say, the simulation of the loop variables u, i and public values of $\mathrm{S}(x)$ needs no knowledge of input shares.
2. **Base case**: the induction hypothesis is true for $i = 1$ since $T_1(u) = \mathrm{S}(u)$.
3. We now describe the simulation of variables observed during SLR_i operation in the following two sub-cases.
   - $i \notin I$**:** according to the construction of I, the possibilities are either by probing the LR outputs of SLR_{n-1} or at least one of the $i+1$ outputs of SLR_i remains unprobed. For all other cases, we would have added the input index i to I when observed. So, none of them come under this case. Hence, we can use Lemma 1 since one of the output shares of SLR_i acts as a one-time pad. In particular, we can assign the outputs of LR $y_i, 1 \le i \le n-1$ uniform random values as in the original circuit since $t \cdot l \le r$. The same argument holds for simulating the final outputs of $\mathrm{S}(x)$ after the final LR.

- $i \in I$**:** since SLR_i receives simulated inputs, we can use x_i to simulate any intermediate variable observed including $T_i(u \oplus x_i, j)$, $T_{aux,i}(u, j)$ in Step 6. These values will be the inputs to Step 9 of of Algorithm 4, *LR-i-r* (see Algorithm 7). Now we move on to the simulation of the probed variables of *LR-i-r*. As mentioned in the proof of Lemma 2, for every $1 \leq j \leq n$, if the adversary probes j^{th} input to LR, then we need to simulate the random value s_j as well. Assign a random value to s_j. Hence, using the inputs to LR and the random values s_j, we can simulate the intermediate variables $y_{1,1}$ or $y_{1,j}$ or the output shares y_j of the LR.

4. It can be observed that the table entries added to set V are nothing but the output shares from the LR, which we show the simulation in Step 3.

5. In either case, we show the simulation of input/intermediate/output variables using the input share indices added to index set I. Hence, the induction hypothesis holds true for SLR_{i+1} as well.

6. The index set I would contain the index n by probing the final lookup or internal variables of LR in Step 12. Hence, simulate the observations using x_n, as explained above.

This proves the t-SNI security of Theorem 7.

□

C.2 Proof of Theorem 8 (Increasing shares using multiple PRG)

As explained in Section C.1, this proof is the same as the arguments provided for other constructions, except for the change that we need x_i to simulate the outputs of $SLR_i, i < (n-1)$. We provide detailed proof for completeness.

Proof. We follow an ISW way of simulation to prove Theorem 8. Initially, we construct an index set I to hold the set of input shares based on the observations made by the adversary. Let the adversary observe the input/intermediate variables using t_1 probes and put t_o probes on output variables such that

$$t_1 + t_o \leq t < n.$$

To prove that the S-box gadget is t-SNI, we need to show that the simulation of variables (including input/intermediate/output variables) observed using t probes requires at most t_1 input shares. Since the table T is shifted and refreshed $n-1$ times, we use $T_i, T_{aux,i}$ to represent the lookup and the auxiliary tables, respectively. Let SLR_i represent the sequence of shift by x_i and the subsequent LR operation. Also, we maintain a list of table indices probed across shifts i, $1 \leq i \leq n-1$. Let $V = \{T_i(u, j),\ 1 \leq i \leq n-1, u \in \{0,1\}^k, 1 \leq j \leq n\}$, $|V| \leq t_1$, where $T_i(u, j)$ represents the j^{th} element in u^{th} vector of table T during i^{th} shift.

We construct a set of input share indices required to show the simulation of the t observations. As described, the steps involved are $SLR_i, 1 \leq i \leq n-1$ and final table lookup. Let I represent the set of input share indices and the construction of I proceeds as follows:

1. Initialise $I = \{\}$.

2. Add i to I if x_i or $u \oplus x_i$ $(1 \leq i \leq n)$ are probed.

3. Add i to I if any of the intermediate variables of the form $T_i(u, j)$, $T_{aux,i}(u, j)$ or $T_i(u \oplus x_i, j)$ where $1 \leq i \leq n-1$ are probed.

4. Add n to I if $T_{n-1}(x_n, j)$ is probed.

5. Add i to I if the internal variables y_1 or $y_{1,j}$, $2 \leq j \leq n$ of SLR_i are probed.

6. Add i to I only if all the $i+1$ outputs of $SLR_i, 1 \leq i \leq n-2$ are probed.

7. Add no index to I by probing the outputs of SLR_{n-1}. The same argument holds for the final sharing of $\mathrm{S}(x)$ in Step 12 of Algorithm 4.

Since we are adding at most one index per internal probe and no index is added to I while probing the output shares, $|I| \leq t_1$. In the first step, we show the simulation of the partitions $A_{ind}, ind \in R^*$.

- When the internal variables s_j or the outputs, y_j of the LR or the output shares $T_i(j)$ of SLR_i with indices j, $2 \leq j \leq n$ are probed, then we need to simulate the corresponding A_{ind} where $ind = \big(i \cdot (i-1)/2\big) + j - 1\big)$.

- Also, if the final shares $y_j, 2 \leq j \leq n$ of $\mathrm{S}(x)$ are probed, then $ind = \big((n-1) \cdot (n-2)/2\big) + j - 1$.

- As explained in Section 4.3, each of these subsets are output pseudorandom values of size 2^k from a non-robust linear lt-wise independent PRG with $l = 1$. Assign uniform random values to t outputs of the non-robust PRG having the index *ind*.

- Since the polynomial-based PRG construction is linear and since only lt outputs are assigned, sample the PRG input seed following the same distribution as the output values.

- Then, using the sampled input seed, compute the remaining outputs of the partition A_{ind}.

Once we show the simulation of $A_{ind}, ind \in R^*$, we proceed with the simulation of SLR_i. We show the simulation using induction. Assume that SLR_i receives the simulated inputs.

1. Needless to say, the simulation of the loop variables u, i and public values of $\mathrm{S}(x)$ requires no knowledge of input shares.

2. **Base case**: the induction hypothesis is true for $i = 1$ since $T_1(u) = \mathrm{S}(u)$.

3. We now show the simulation of variables observed during SLR_i operation.

   - $i \notin I$**:** according to the construction of I, the possibilities are either by probing the LR outputs of SLR_{n-1} or at least one of the $i+1$ outputs of SLR_i remains unprobed. For all other cases, we would have added the input index i to I when observed. So, none of them fall under this case. If the probed outputs of LR are $y_j, 2 \leq j \leq n-1$, then assign the value from the corresponding partition A_{ind} which is already simulated. If the probed output is first share y_1, then there exist at least one unprobed $y_{j^*}, 2 \leq j^* \leq n$ such that y_{j^*} acts as a one-time pad. Hence, assign a uniform random value to y_1. Follow the same steps for simulating the output shares after the final LR.
   - $i \in I$**:** since SLR_i receives simulated inputs, we can use x_i to simulate any intermediate variable observed including $T_i(u \oplus x_i, j)$, $T_{aux,i}(u, j)$ in Step 6. These values will be the inputs to Step 9 of of Algorithm 4, *LR-i-m* (see Algorithm 8). Now we move on to the simulation of the probed variables of *LR-i-r*. As explained in the previous step, assign the values to s_i from the partition $A_{ind}, ind \in R^*$. Using the simulated random masks s_j and input share x_i, compute the intermediate variables $y_{1,j}$ or the output shares y_j of SLR_i.

4. It can be observed that the table entries added to set V are nothing but the output shares from the LR, which we show the simulation in Step 3.

5. In either case, we show the simulation of input/intermediate/output variables using the input share indices added to index set I. Hence, the induction hypothesis holds true for SLR_{i+1} as well.

6. The index set I would contain the index n by observing the final lookup or internal variables of LR in Step 12. Hence, simulate the observations using x_n.

This proves the t-SNI security of Theorem 8.

□