An Optimal Sampling Technique for Distinguishing Random S-boxes

Paul Stankovski and Martin Hell
Department of Electrical and Information Technology
Lund University, Sweden
E-mail: {paul,martin}@eit.lth.se

***Abstract*—The nonrandom behavior of the outputs of a random S-box can be exploited when constructing distinguishers for cryptographic primitives. Different methods of constructing samples from the outputs have been used in the literature. However, it has been unclear exactly how these methods differ and which method is optimal. We analyze four different sampling techniques. We prove that two of these sampling techniques result in dependent samples. We further show one sampling technique that is optimal in terms of error probabilities in the resulting distinguisher. However, this sampling technique is quite impractical as it requires very large storage. We further show a fourth sampling technique that is much more practical, and we prove that it is equivalent to the optimal one. We also show an improved algorithm for calculating the associated probability distributions that are required for the attack.**

I. Introduction

Random S-boxes can appear in cryptanalysis when observations, e.g., linear sums of keystream bits in stream ciphers, can be regarded as outputs of a large table. In this paper we study such random S-boxes. More specifically, we study how to perform an optimal distinguisher from the observations. A random S-box is an a-to-b-bit mapping which can be seen as a table containing $n = 2^a$ random entries of b bits each. Our work is motivated by the analysis of the HC-128 stream cipher as performed in [2], [5], but the results are applicable to all random S-boxes. HC-128 is a stream cipher in the eSTREAM portfolio, and is thus considered to be one of the most promising stream ciphers today. Indeed, it is very fast in software and it has been shown to resist cryptanalytic attacks very well. There are no attacks (not relying on side-channel information) that are more efficient than exhaustive key search. The distinguishing attack given in [2] is currently the most efficient non-generic attack, and that attack is based on the attack given in [5]. The improvement comes from a more efficient sampling technique, reducing the number of keystream bits required by the distinguisher.

We analyze different sampling techniques. We show that the sampling technique used in [5] significantly underestimates the number of samples needed by the distinguisher as the samples are not independent. We further prove that it is not possible to take two independent biased samples at all, unless the S-box is reinitialized. The optimal sampling technique is thus to take one sample containing all information, i.e., to consider a vector of all outputs. The problem with this vector is that, for large vector sizes, its probability distribution is infeasible both to compute and to store. Due to this, a shortcut was used in [2], namely to consider only the *weight* of the vector. We show that this weight probability distribution is equivalent to the optimal probability distribution and, as a result, it is not possible to further improve the sampling used in [2]. Finally, we give a new algorithm for computing the vector weight distribution that improves the one given in [2]. Our new algorithm uses much less memory (optimal) and saves 80-85% in time.

The final publication is available at IEEE Xplore, Copyright IEEE, DOI: 10.1109/ISIT.2012.6284680. This work was sponsored in part by the Swedish Research Council (Vetenskapsrådet) under grant 621-2006-5249.

II. Preliminaries

Two outputs from an S-box of size n are equal with probability (at least) $\frac{1}{n}$ since the same entry may have been used twice. This simple observation can be used to construct a distinguisher for random S-boxes.

We consider an a-to-b-bit random S-box that allows ℓ observations before it is rerandomized. A set of ℓ such S-box observations will be referred to as a *chunk*, and the observations themselves are denoted $s_1, \ldots, s_\ell$. The S-box may be regarded as a table with $n = 2^a$ entries of b bits each, and we will restrict ourselves to the case when $b = 1$, i.e., we observe bits, but the results carry on to the general case. The number of ℓ-bit chunks is denoted k, and we denote the total number of observations by $N = k \cdot \ell$.

As noted above, it is known (see [5]) that the xor sum of a pair of output bits is biased, and this bias stems from the fact that the same S-box entry may have been probed for both outputs. More specifically, for $i \neq j$,

$$\Pr(s_i = s_j) = \frac{1}{2}\frac{n-1}{n} + \frac{1}{n} = \frac{1}{2}(1 + \frac{1}{n}) = \frac{1}{2} + 2^{-(a+1)}, \quad (1)$$

and the bias in (1) can be used to construct a distinguisher. The main problem we study in this paper is exactly *how* to construct this distinguisher when the number of S-box observations is more than two. That is, how should a cryptanalyst use the observations to construct an optimal distinguisher?

The empirical probability distribution as defined by the sampling is denoted P^*. The corresponding (theoretical) probability distribution of the S-box is denoted P_1 while its uniform distribution is denoted P_2. The optimal hypothesis test is given by the Neyman-Pearson lemma, see e.g., [1].

Lemma 1 (Neyman-Pearson): Let $X_1, X_2, \ldots, X_t$ be iid random variables according to P^*. Consider the decision problem corresponding to the hypotheses $P^* = P_1$ vs. $P^* = P_2$. For $Q \geq 0$ define a region

$$\mathcal{A}_t(Q) = \left\{ \frac{P_1(x_1, x_2, \ldots, x_t)}{P_2(x_1, x_2, \ldots, x_t)} > Q \right\}.$$

Let $\alpha_t = P_1^t(\mathcal{A}_t^c(Q))$ and $\beta_t = P_2^t(\mathcal{A}_t(Q))$ be the error probabilities corresponding to the decision region $\mathcal{A}_t$. Let $\mathcal{B}_t$ be any other decision region with associated error probabilities α^* and β^*. If $\alpha^* \leq \alpha$, then $\beta^* \geq \beta$.

If we want the error probabilities to be equal we set $Q = 1$. In other words, we decide $P^* = P_1$ if

$$\frac{P_1(x_1, \ldots, x_t)}{P_2(x_1, \ldots, x_t)} > 1 \underset{\text{indep.}}{\Leftrightarrow} \sum_{i=1}^{t} \log \frac{P_1(x_i)}{P_2(x_i)} > 0, \tag{2}$$

and $P^* = P_2$ otherwise. The equivalence in (2) is valid when the samples $x_1, \ldots, x_t$ are independent.

In our case, the samples x_i will be constructed from the observations s_j. Note that the Neyman-Pearson lemma, which gives the optimal distinguisher, requires that the samples x_i are independent. By sampling technique we mean how to use the observations to build the samples used in the distinguisher.

If the samples are very easy to construct from the observations, we can say that the online computational complexity of the attack is given by the number of terms t in the sum (2). The offline complexity is the time needed to compute P_1.

III. Sampling from a Random S-box

We will consider the following four sampling techniques:

- **All-Pairs Sampling (APS)** Take all pairs of observations $(s_i, s_j), 1 \leq i < j \leq \ell$ as samples. Let P_1 be the distribution corresponding to (1), i.e., $\Pr(s_i = s_j) = \frac{1}{2}(1 + \frac{1}{n})$ and $\Pr(s_i \neq s_j) = \frac{1}{2}(1 - \frac{1}{n})$. P_2 is the uniform distribution, $\Pr(s_i = s_j) = \Pr(s_i \neq s_j) = \frac{1}{2}$.
- **Linear-Pairs Sampling (LPS)** Take the pairs of observations $(s_i, s_{i+1}), 1 \leq i < \ell$ as samples and let P_1 and P_2 be as for APS above.
- **Vector Sampling (VS)** Take the vectors $(s_1, s_2, \ldots, s_\ell)$ as samples and perform the hypothesis test with the corresponding vector probability distributions as P_1 and P_2.
- **Weight Sampling (WS)** Take the vector weights $\|(s_1, s_2, \ldots, s_\ell)\| = \sum_{i=1}^{\ell} s_i$ as samples and perform the hypothesis test with the corresponding vector weight probability distributions as P_1 and P_2.

It is clear that Vector Sampling (VS) is optimal since it preserves all information in the samples. The drawbacks are that the distribution is very large in storage (2^ℓ) and that it is demanding to compute. APS was applied in [4], [5]. It uses the easily computed bias in (1) and produces many samples. For ℓ observations, $\binom{\ell}{2}$ samples are produced. Due to the dependency between samples, LPS was suggested in [2] and WS was also applied as an improvement. However, it was an open question whether it was possible to improve over WS as it appears that not all sample information is retained in the vector weight samples.

The rest of the paper is organized as follows. In Sections III-A and III-B we prove that APS and LPS are faulty. In Sections III-C and III-D we give algorithms for computing the required distributions for VS and WS, respectively. We also prove that VS and WS are equivalent in terms of the performance of the resulting distinguisher. Section IV explicitly compares APS, LPS and WS. The paper is concluded in Section V.

A. All-Pairs Sampling (APS)

The Neyman-Pearson lemma assumes that all samples are independent and identically distributed. In APS sampling, all possible bit pairs in an ℓ-bit chunk are taken as samples, producing in total $k\binom{\ell}{2}$ samples. It is very easy to prove that these samples are not independent. Consider a chunk with $\ell = 3$, where we take the samples $(s_1, s_2), (s_1, s_3)$ and (s_2, s_3). If we know the first two samples, then we also know the last sample, i.e.,

$$H(S_2 \oplus S_3 | S_1 \oplus S_2, S_1 \oplus S_3) = 0,$$

where $H(\cdot)$ denotes the entropy function, $\oplus$ denotes xor and S_1, S_2 and S_3 are random variables corresponding to the three observations. This argument is easily extended to the general case with arbitrary ℓ, which also serves as a direct motivation for defining and using LPS sampling.

Even though the samples are dependent, APS is very easy to apply. Computing and storing the P_1 requires negligible memory and can be trivially done by hand, see (1). However, the large number of samples gives an online computational complexity of $k\binom{\ell}{2} = O(k\ell^2)$.

B. Linear-Pairs Sampling (LPS)

In LPS sampling we take (s_1, s_2) as the first sample and then only take one new sample for each new observation. This procedure produces $\ell - 1$ samples per chunk for a total of $k(\ell - 1)$ samples. In order to show that this sampling technique also gives dependent samples, for P_1 we calculate and compare $\Pr(s_3 = s_2 | s_2 = s_1)$ and $\Pr(s_3 = s_2 | s_2 \neq s_1)$ to see that the probability of pair equality in one sample depends on the pair equality of the previous one.

We regard the S-box as a table with n entries. The first time we read a specific entry in the table, we say that we "open" the entry. Consider $\Pr(s_3 = s_2 | s_2 \neq s_1)$ first. Given that $s_2 \neq s_1$, we must have opened precisely two entries in the table, one 0 and one 1. We can now have $s_3 = s_2$ in two different ways, by reading s_3 from either an "old" entry (same as s_2) or a "new" previously unopened one. Thus, we have

$$\Pr(s_3 = s_2 | s_2 \neq s_1) = 1 \cdot \frac{1}{n} + \frac{1}{2} \cdot \frac{n-2}{n} = \frac{1}{2}.$$

Calculating $\Pr(s_3 = s_2 | s_2 = s_1)$ divides into two cases.

Case A: s_1 and s_2 were read from the same entry.

Case B: s_1 and s_2 were read from different entries.

The probability of case A is $p = \frac{1}{n}$, while that of case B is $q = \frac{n-1}{2n}$. Given case A, the probability that $s_3 = s_2$ is

$$a = \frac{1}{n} + \frac{n-1}{2n} = \frac{n+1}{2n}.$$

Given case B, the probability that $s_3 = s_2$ is

$$b = \frac{2}{n} + \frac{n-2}{2n} = \frac{n+2}{2n}.$$

In total we get $\Pr(s_3 = s_2 | s_2 = s_1) =$

$$\frac{p}{p+q} \cdot a + \frac{q}{p+q} \cdot b = \frac{1}{2}(1 + \frac{2}{n+1}) > \frac{1}{2},$$

from which we conclude that

$$\Pr(s_3 = s_2 | s_2 \neq s_1) \neq \Pr(s_3 = s_2 | s_2 = s_1).$$

This proves that LPS is also erroneous in assuming independence between samples.

One may further note that the same probabilities are valid for any other overlapping pair, i.e., for $\Pr(s_k = s_j | s_j \neq s_i)$ and $\Pr(s_k = s_j | s_j = s_i)$ for all distinct indices i, j and k.

This dependency may seem natural since the two samples are overlapping in one of the observations. Collecting samples in a non-overlapping fashion according to $(s_1, s_2), (s_3, s_4), (s_5, s_6)$, and so on, may at first glance seem better. However, by performing similar calculations we can also prove that

$$\Pr(s_4 = s_3 | s_2 \neq s_1) \neq \Pr(s_4 = s_3 | s_2 = s_1).$$

The corresponding calculations show that

$$\begin{aligned} \Pr(s_4 = s_3 | s_2 \neq s_1) &= \frac{1}{2}\left(1 + \frac{n-2}{n^2}\right) \text{ and} \\ \Pr(s_4 = s_3 | s_2 = s_1) &= \frac{1}{2}\left(1 + \frac{n^2+3n-2}{n^2(n+1)}\right). \end{aligned}$$

This means that the probability of pair equality in one sample depends on the previous one in this case as well. This immediately generalizes to all non-overlapping pairs, i.e., the same holds for $\Pr(s_j = s_i | s_v \neq s_u)$ and $\Pr(s_j = s_i | s_v = s_u)$ for all distinct indices i, j, u and v. Since the overlapping and non-overlapping cases are exhaustive, we can conclude that any two samples will be dependent. An intuitive explanation for this is that a sample leaks information about the entries in the S-box. This information will affect the probability of the next sample since we may read the same entries as before. We summarize this result in Theorem 2.

Theorem 2 (Random S-box Sampling Theorem): It is not possible to extract more than one independent sample from a chunk $s_1, \ldots, s_\ell$ of observations from a random S-box.

The advantage of LPS over APS is that fewer samples are used. The computational complexity of the online phase of LPS is $k(\ell - 1) = O(k\ell)$.

Algorithm I – Vector Distribution (vd)

Input: S-box size n, vector length ℓ, current depth d, current probability p, probability distribution container $dist$ of length 2^ℓ, vector v, number of opened table entries with zeros a_0, number of opened table entries with ones a_1.
Output: probability distribution $dist$.
Initial recursion parameters: $dist$ zeroized,
$(d, p, v, a_0, a_1) = (0, 1, 0, 0, 0)$.

```
if ($d == \ell$) { $dist[v]$ += $p$; return; }
if ($a_0 > 0$) $vd(dist, n, \ell, d+1, p \cdot \frac{a_0}{n}, v\|0, a_0, a_1)$; /* old 0 */
if ($a_1 > 0$) $vd(dist, n, \ell, d+1, p \cdot \frac{a_1}{n}, v\|1, a_0, a_1)$; /* old 1 */
if ($a_0 + a_1 < n$) { /* table not exhausted */
    $vd(dist, n, \ell, d+1, p \cdot \frac{n-(a_0+a_1)}{2n}, v\|0, a_0+1, a_1)$; /* new 0 */
    $vd(dist, n, \ell, d+1, p \cdot \frac{n-(a_0+a_1)}{2n}, v\|1, a_0, a_1+1)$; /* new 1 */
}
```

C. *Vector Sampling (VS)*

In order to correctly apply the Neyman-Pearson lemma, we need to find the probability of the complete chunk. Thus, we collect all observations in one vector $(s_1, s_2, \ldots, s_\ell)$. The vector probability distributions P_1 and P_2 both have a domain of size 2^ℓ, which determines the storage cost for handling P_1 and P_2 with VS.

For P_2, all vectors are equally likely, resulting in identical probability values $P_2(v) = 2^{-\ell}$ for all vectors v.

The S-box vector probability distribution P_1 can be calculated according to Algorithm I, which is stated recursively for simplicity. The main idea here is simply to keep track of how many entries in the S-box that have revealed zeros and ones, as this information will enable us to compute the associated probabilities at each stage.

The storage requirement for Algorithm I is precisely 2^ℓ (probability entries), and since this amount of memory is necessary to store the resulting probability distribution, no other algorithm can do better in terms of memory. The time complexity of Algorithm I is also exponential in ℓ, at most $4^\ell = 2^{2\ell}$, because every call at depth d results in at most 4 calls at depth $d + 1$. By employing dynamic programming, see e.g., [3], it is possible to improve this time complexity to $O(n^2 2^\ell)$ at the expense of increased storage, $O(n^2 2^\ell)$, but the running time must still necessarily be exponential in ℓ.

For large ℓ, i.e., when many observations are generated before the S-box is reinitialized, the vector sampling technique is infeasible since the distribution P_1 is both too large to store and too demanding to compute.

D. *Weight Sampling (WS)*

Now consider WS, for which we take vector weights $\|(s_1, s_2, \ldots, s_\ell)\| = \sum_{i=1}^{\ell} s_i$ as samples. The corresponding vector probability distributions P_1 and P_2 have domains of size $\ell + 1$, which is much more manageable than those for VS.

For WS we begin with P_2. Every vector is equally likely in the ideal case, so the resulting vector weight probability distribution is combinatorially determined by

$$P_2(w) = \binom{\ell}{w} 2^{-\ell}$$

Algorithm II – Weight Distribution (wd)

Input: S-box size n, vector length ℓ, current depth d, current probability p, probability distribution container $dist$ of length $\ell+1$, weight w, number of opened table entries with zeros a_0, number of opened table entries with ones a_1.
Output: probability distribution $dist$.
Initial recursion parameters: $dist$ zeroized,
$(d, p, w, a_0, a_1) = (0, 1, 0, 0, 0)$.

```
if ($d$ == $\ell$) { $dist[w]$ += $p$; return; }
if ($a_0 > 0$) $wd(dist, n, \ell, d+1, p \cdot \frac{a_0}{n}, w, a_0, a_1)$; /* old 0 */
if ($a_1 > 0$) $wd(dist, n, \ell, d+1, p \cdot \frac{a_1}{n}, w+1, a_0, a_1)$; /* old 1 */
if ($a_0 + a_1 < n$) { /* table not exhausted */
    $wd(dist, n, \ell, d+1, p \cdot \frac{n-(a_0+a_1)}{2n}, w, a_0+1, a_1)$; /* new 0 */
    $wd(dist, n, \ell, d+1, p \cdot \frac{n-(a_0+a_1)}{2n}, w+1, a_0, a_1+1)$; /* new 1 */
}
```

Algorithm III – Vector Probability (vp)

Input: vector probability $prob$ (accumulated), current probability p, S-box size n, vector v (s_ℓ to s_1 from MSB to LSB), vector length t, number of opened table entries with zeros a_0, number of opened table entries with ones a_1.
Output: vector probability $prob$.
Initial recursion parameters:
$(prob, p, t, a_0, a_1) = (0, 1, \ell, 0, 0)$.

```
if ($t$ == 0) { $prob$ += $p$; return; }
if ($v$ & 1) {/* next output bit is 1 */
    if ($a_1 > 0$) $vp(prob, p \cdot \frac{a_1}{n}, n, v \gg 1, t-1, a_0, a_1)$; /* old 1 */
    if ($a_0 + a_1 < n$) /* table not exhausted */
        $vp(prob, p \cdot \frac{n-(a_0+a_1)}{2n}, n, v \gg 1, t-1, a_0, a_1+1)$; /* new 1 */
} else {/* next output bit is 0 */
    if ($a_0 > 0$) $vp(prob, p \cdot \frac{a_0}{n}, n, v \gg 1, t-1, a_0, a_1)$; /* old 0 */
    if ($a_0 + a_1 < n$) /* table not exhausted */
        $vp(prob, p \cdot \frac{n-(a_0+a_1)}{2n}, n, v \gg 1, t-1, a_0+1, a_1)$; /* new 0 */
}
```

for all vector weights $0 \le w \le \ell$.

P_1 can be calculated according to Algorithm II, which is just a simple modification of Algorithm I. Instead of passing on a (partial) vector we now pass on the (accumulated) vector weight. The algorithm is, again, stated recursively for simplicity, but it can also be implemented in a dynamic programming fashion as detailed in [2]. Upper bound formulas for memory and computational complexity for handling vectors of size ℓ derived from an S-box of size n are given by $\frac{n^2\ell}{2}$ and $\frac{n^2\ell^2}{4}$, respectively.

We now explicitly prove that VS and WS are equivalent in terms of keystream complexity of the resulting distinguisher. We first present Algorithm III which calculates the probability of an S-box outputting a specific vector – the vector probability. The correctness of Algorithm III follows from its relationship to Algorithm I.

Theorem 3 (WS is optimal): WS is equivalent to VS in terms of the Neyman-Pearson test (Lemma 1).

Proof: The proof follows if we can show that all vectors of equal weight are equiprobable, because the probability distributions P_1 and P_2 for WS can then be derived from those of VS by grouping all probabilities for vectors of equal weight. In such a case the Neyman-Pearson test is equal for both sampling techniques, showing that no information is lost when considering WS over VS.

It is sufficient to show that the vector probability is invariant under pairwise bit flips. That is, we need to show that the vector probability does not change if a neighboring pair of bits in a vector are flipped from 10 to 01 (or from 01 to 10).

Let $v = (s_1, s_2, \ldots, s_\ell)$ be a vector for which $s_i = 0$ and $s_{i+1} = 1$ for some i, and let v' be the corresponding vector with $s'_i = 1$ and $s'_{i+1} = 0$. Let v_j denote the vector $(s_j, s_{j+1}, \ldots, s_\ell)$. We need to show that $vp(p, v, a_0, a_1) = vp(p, v', a_0, a_1)$ (we omit some of the less interesting parameters).

All recursive calls to $vp(p, v, a_0, a_1)$ and $vp(p, v', a_0, a_1)$ are identical up to depth i, so it is enough to consider any two such calls $vp(p, v_i, a_0, a_1)$ and $vp(p, v'_i, a_0, a_1)$ at depth i. We need to show that both these calls give rise to the same quadruple of function calls at depth $i+2$, two levels deeper.

$vp(p, v_i, a_0, a_1)$ splits into

$$vp(p \cdot \frac{a_0}{n}, v_{i+1}, a_0, a_1) \text{ and}$$
$$vp(p \cdot \frac{n - a_0 - a_1}{2n}, v_{i+1}, a_0 + 1, a_1)$$

at level $i+1$, and then into

$$vp(p \cdot \frac{a_0 a_1}{n^2}, v_{i+2}, a_0, a_1),$$
$$vp(p \cdot \frac{a_0(n - a_0 - a_1)}{2n^2}, v_{i+2}, a_0, a_1 + 1),$$
$$vp(p \cdot \frac{(n - a_0 - a_1)a_1}{2n^2}, v_{i+2}, a_0 + 1, a_1) \text{ and}$$
$$vp(p \cdot \frac{(n - a_0 - a_1)(n - (a_0 + 1) - a_1)}{(2n)^2}, v_{i+2}, \ldots)$$

at level $i+2$. Similarly, $vp(p, v'_i, a_0, a_1)$ splits into

$$vp(p \cdot \frac{a_1 a_0}{n^2}, v'_{i+2}, a_0, a_1),$$
$$vp(p \cdot \frac{a_1(n - a_0 - a_1)}{2n^2}, v'_{i+2}, a_0 + 1, a_1),$$
$$vp(p \cdot \frac{(n - a_0 - a_1)a_0}{2n^2}, v'_{i+2}, a_0, a_1 + 1) \text{ and}$$
$$vp(p \cdot \frac{(n - a_0 - a_1)(n - a_0 - (a_1 + 1))}{(2n)^2}, v'_{i+2}, \ldots).$$

Here we have $v_{i+2} = v'_{i+2}$, so the sets of calls are identical. ■

A direct consequence of Theorem 3 is that, although VS is highly impractical to use due to its exponential time- and memory complexities, WS will provide the same result as VS at a much lower cost, allowing much larger ℓ-values. The computational complexity of the distinguisher is given by $O(k)$.

We can also use Theorem 3 to improve the efficiency of computing P_1 with Algorithm II. This is also true for the dynamic programming variant of the algorithm presented in [2]. Since all vectors of equal weight are equiprobable, we need only consider vectors of type $00\ldots011\ldots1$. The improved algorithm is to calculate the probabilities for all $\ell+1$ such vectors by using a dynamic programming version of Algorithm III. Recall that the time and memory complexities

TABLE I

A COMPARISON OF ONLINE AND OFFLINE COMPUTATIONAL AND MEMORY COMPLEXITIES FOR A DISTINGUISHING ATTACK WHEN USING EACH SAMPLING TECHNIQUE.

	online comp	online mem	offline comp	offline mem
APS	$O(k\ell^2)$	$O(1)$	$O(1)$	$O(1)$
LPS	$O(k\ell)$	$O(1)$	$O(1)$	$O(1)$
VS	$O(k)$	$O(2^\ell)$	$O(n^2 2^\ell)$	$O(n^2 2^\ell)$
WS	$O(k)$	$O(\ell)$	$O(n^2\ell^2)$	$O(\ell)$

given in [2] are $O(n^2\ell^2)$ and $O(n^2\ell)$, respectively, so memory usage is limiting in practice. For the new algorithm we need only $O(\min(n,\ell))$ memory for storing intermediate probability values and $O(\ell)$ storage to hold the resulting probability distribution. An additional improvement is to recognize that the distribution is symmetric, so we need only compute half of it.

While the time required is still $O(n^2\ell^2)$, the constants are better. Our simulations show that we save 80-85% in time, and the memory usage is $O(\ell)$, i.e., it no longer depends on the size of the S-box. This is optimal since it equals the length of the vector.

IV. COMPARING SAMPLING TECHNIQUES

We have shown above that both APS and LPS are erroneous as the corresponding samples are not taken independently. Still, both techniques are very simple to apply. The distribution P_1 is very easy to compute in each case, and checking if $(s_i = s_j)$ is trivial, but the drawback is that the resulting distinguisher will not be optimal. For optimality, WS (or VS) must be used. This optimality comes at the cost of a larger precomputational complexity, i.e., for computing P_1. Table I summarizes the important parameters corresponding to each sampling technique. Note that we assume that the best dynamic programming variant is used to compute the probability distributions P_1 for VS and WS. The actual performance of the attack using each of the sampling techniques has been simulated. As VS and WS give the exact same distinguisher performance, only WS is included in the simulations. For a fair comparison, we assume that the number of chunks is the same for all variants, i.e., APS and LPS are allowed to use many more samples than WS, but all use the same number of observations. The two plots in Fig. 1 show the error probabilities for APS, LPS and WS when the size of the S-box is $n = 64$ and the number of observations in each chunk is $\ell = 20$ and $\ell = 500$, respectively. Similarly, Fig. 2 shows the error probabilities when the size of the S-box is $n = 512$.

From the simulations we can see that both LPS and APS are outperformed by WS. It is interesting to see that APS is not very much worse when the same number of chunks is considered. However, we stress again that APS uses a factor $\binom{\ell}{2}$ more samples than WS. This clearly shows the problem of assuming independent samples when they are in fact very

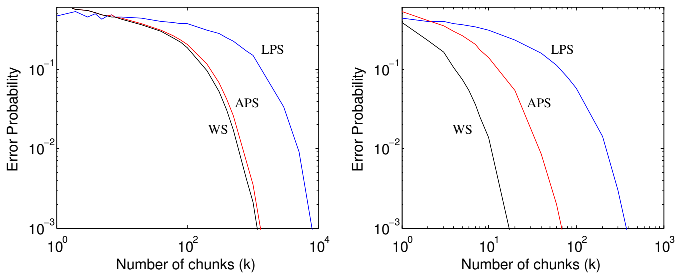

Fig. 1. Success probability as a function of the number of chunks k. Size of S-box is $n = 64$ and ℓ is 20 (left) and 500 (right)

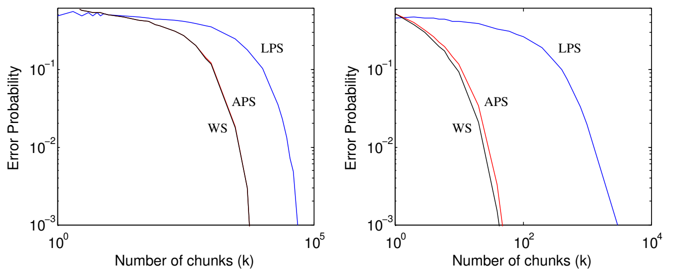

Fig. 2. Success probability as a function of the number of chunks k. Size of S-box is $n = 512$ and ℓ is 20 (left) and 500 (right). Note that WS and APS almost coincide for $\ell = 20$.

dependent. Also, the factor $\binom{\ell}{2}$ makes APS impractical for large ℓ.

Specifically for HC-128, each observation is a linear combination of keystream bits. In this case, the comparison assumes an equal number of keystream bits for all sampling techniques.

Looking at Fig. 1 and Fig. 2 it seems that the performance of APS approaches that of WS when the S-box size n increases and when the chunk size ℓ decreases. Thus, for large n and small ℓ their performances are practically equal, while for small n and large ℓ, WS clearly outperforms APS. We have simulated many other choices of n and ℓ, and all simulations show this same tendency.

V. CONCLUSIONS

Four different sampling techniques for random S-box outputs have been considered and analyzed. We have proved that it is not possible to take two independent samples from one chunk of a random S-box, which implies that APS and LPS are sub-optimal as they impose a higher error probability in the resulting distinguisher. We have also proved that WS is equivalent to the optimal VS, and that WS is much more practical than VS. WS is thus the preferred sampling technique. We have also presented an improved algoritm for the offline computation of P_1 for WS.

Even though APS and LPS are not optimal, they are very simple to apply. For large S-boxes that are frequently rerandomized, the APS technique is very close to optimal.

REFERENCES

[1] T. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley series in Telecommunication. Wiley, 1991.

[2] P. Stankovski, S. Ruj, M. Hell, and T. Johansson. Improved Distinguishers for HC-128. *Designs, Codes and Cryptography*, pages 1–16. *http://dx.doi.org/10.1007/s10623-011-9550-9*.

[3] R. Rivest T. Cormen, C. Leiserson and C. Stein. *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

[4] H. Wu. A New Stream Cipher HC-256. In *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 226–244. Springer-Verlag, 2004.

[5] H. Wu. The Stream Cipher HC-128. In *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 39–47. Springer-Verlag, 2008.