A New Mode of Encryption Providing A Tweakable Strong Pseudo-Random Permutation

Debrup Chakraborty and Palash Sarkar*

Computer Science Department,
CINVESTAV-IPN
Av. IPN No. 2508 Col. San Pedro Zacatenco
Mexico, D.F. 07360. Mexico
email: debrup@cs.cinvestav.mx

*Applied Statistics Unit
Indian Statistical Institute
203 B.T. Road
Kolkata 700108, India
email: palash@isical.ac.in

Abstract

We present PEP, which is a new construction of a tweakable strong pseudo-random permutation. PEP uses a hash-encrypt-hash approach which has recently been used in the construction of HCTR. This approach is different from the encrypt-mask-encrypt approach of constructions such as CMC, EME and EME*. The general hash-encrypt-hash approach was earlier used by Naor-Reingold to provide a generic construction technique for an SPRP (but not a tweakable SPRP). PEP can be seen as the development of the Naor-Reingold approach into a fully specified mode of operation with a concrete security reduction for a tweakable strong pseudo-random permutation. The security bound of HCTR which is also based on the Naor-Reingold approach is weaker than that of PEP. Compared to previous known constructions, PEP is the only construction of tweakable SPRP which uses a single key, is efficiently parallelizable and can handle an arbitrary number of blocks.

Keywords: mode of operation, tweakable encryption, strong pseudo-random permutation.

1 Introduction

A block cipher is a fundamental primitive in cryptography. A block cipher by itself can encrypt only fixed length strings. Applications in general require encryption of long and arbitrary length strings. A mode of operation of a block cipher is used to extend the domain of applicability from fixed length strings to long and variable length strings. The mode of operation must be secure in the sense that if the underlying block cipher satisfies a certain notion of security, then the extended domain mode of operation also satisfies an appropriate notion of security.

A formal model of security for a block cipher is a pseudo-random permutation [8] which is formalized as a keyed family of permutations. Pseudo-randomness of the permutation family requires a computationally bounded adversary to be unable to distinguish between a random permutation and a permutation picked at random from the family. Strong pseudo-random permutations (SPRPs) require computational indistinguishability even when the adversary has access to the inverse permutations.

A mode of encryption usually provides two security assurances – privacy and authenticity. For example, OCB [13] is a mode of operation (providing both privacy and autheticity) which extends the domain of a block-cipher to arbitrary strings. An SPRP which can encrypt arbitrary length strings, can be viewed as a mode of operation with a somewhat different goal. Such a mode of operation is length preserving and no tag is produced. Hence, authentication is of limited nature. A change in the ciphertext cannot be detected but the decryption of the tampered ciphertext will result in a plaintext which is indistinguishable from a random string. Additional redundancy in the message introduced by higher level applications might even allow the detection of the tampering. This point is discussed in details by Bellare and Rogaway [1].

Tweakable encryption was introduced by Liskov, Rivest and Wagner [7] which added an extra input called tweak to a block cipher. This allows simplification of several applications. The paper [7] introduced both tweakable PRP and SPRP. In the adversarial model for tweakable SPRP, the adversary queries the encryption (resp. decryption) oracle with a tweak and the plaintext (resp. ciphertext). The adversary is allowed to repeat the tweaks in its queries to the encryption and the decryption oracles. A tweakable SPRP provides a mode of operation having all the advantages of an SPRP with the additional flexibility of having a tweak. The constructions CMC [3], EME [4], EME* [2] and HCTR [16] are proved to be tweakable SPRPs under the assumption that the underlying block cipher is an SPRP. As mentioned in [3], such a primitive is well suited for disk encryption where the tweak can be considered to be the sector address.

Our Contributions: We present a new construction of a tweakable SPRP called PEP (for Polynomial hash-Encrypt-Polynomial hash). PEP uses a block cipher which can encrypt an n-bit string to construct an encryption algorithm which can encrypt mn-bit strings for any $m \geq 1$. It uses two Wegman-Carter [15] style polynomial hashes over the binary field $GF(2^n)$. (Similar hashes have been earlier used in GCM [9] and HCTR.) The new construction is proved to be a tweakable SPRP assuming that the underlying block cipher is an SPRP. Below we mention some interesting features of PEP.

1. PEP is a fully specified mode of operation providing a tweakable SPRP. The security proof for PEP provides a concrete security bound which is the usual quadratic birthday bound earlier obtained for CMC, EME and EME*. The security bound of HCTR is weaker and the security degradation is cubic.

2. The total computation cost of PEP for encrypting an m-block message consists of $m + 5$ block cipher calls for $m \geq 2$ ($m + 3$ for $m = 1$), and $(4m - 6)$ multiplications in $GF(2^n)$. In contrast, CMC requires $2m + 1$ block cipher calls; EME* requires $(2m + m/n + 1)$ block cipher calls; and HCTR requires m block cipher calls and $2m$ many $GF(2^n)$ multiplications. The exact comparison of the computation costs depends upon several factors such as the implementation platform (hardware or software), availability of suitable co-processors (for software implementation), availability of parallel encryption blocks (for hardware implementation) and most importantly on the actual block cipher being used *and* the efficiency of its implementation.

3. Currently, PEP fulfills a gap in the known constructions. It is the only known construction of

tweakable SPRP which uses a single key, is efficiently parallelizable and can handle arbitrary number of message blocks. Table 1 in Section 3 provides a detailed comparison of PEP with the other tweakable SPRPs.

Related Constructions: To the best of our knowledge, the first suggestion for constructing an SPRP was made by Naor and Reingold in [10]. They suggested the hash-encrypt-hash approach used in PEP. However, as discussed in [3], the description in [10] is at a top level and also the later work [11] does not fully specify a mode of operation. Perhaps more importantly, the work [11] does not consider *tweakable* SPRP since it predates the introduction of tweakable primitives in [7].

The NR approach was rejected in [3] as not being capable of efficient instantiation. The work in [3] and also the later constructions [4, 2] follow a encrypt-mask-encrypt strategy, i.e., there are two layers of encryptions with a layer of masking in between. CMC [3] provides the first efficient, fully specified construction of a tweakable SPRP. Parallel versions of the encrypt-mask-encrypt strategy have been proposed as EME and EME*.

Interestingly, the NR approach made a recent comeback in the HCTR construction. The HCTR construction combines the NR type invertible hash functions with the counter mode of operation. This results in an efficient tweakable SPRP which can handle any message having length $\geq n$ bits. The drawback of HCTR is its weaker security bound and the requirement of having two keys. Currently, PEP can be viewed as the development of the NR approach to the construction of tweakable SPRP.

2 Specification of PEP

We construct the tweakable enciphering scheme PEP from a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and call it PEP$[E]$. The key space of PEP$[E]$ is same as that of the underlying block cipher E and the tweak space is $\mathcal{T} = \{0,1\}^n$. The message space consists of all binary strings of size mn where $m \geq 1$.

Finite Field Arithmetic: An n-bit string can also be viewed as an element in $GF(2^n)$. Thus, we will consider each n-bit string in the specification of PEP as a polynomial over $GF(2)$ modulo a fixed primitive polynomial $\tau(x)$ of degree n. For each n-bit string Z that occur in the description of PEP, we will use $Z(x)$ to denote the corresponding polynomial in $GF(2^n)$. The expressions $p(x)M_1$ (resp. xEN), represent the n-bit string obtained by multiplying the polynomials $p(x)$ and M_1 (resp. x and EN) modulo $\tau(x)$. Also for two n-bit strings Z_1 and Z_2, the expression $Z_1(x)Z_2(x)$ denotes the n-bit string obtained by multiplying $Z_1(x)$ and $Z_2(x)$ modulo $\tau(x)$. Finally, by $R^{-1}(x)$ we will denote the multiplicative inverse of $R(x)$ modulo $\tau(x)$ when $R(x) \neq 0$.

Definition 1 *Let $m \geq 3$ and $p_{m,1}(x), \ldots, p_{m,m}(x)$ be a sequence of polynomials over $GF(2)$ each having degree at most $n-1$. We call such a sequence an* allowed sequence *with respect to a primitive polynomial $\tau(x)$ of degree n if the following two conditions hold.*

1. *$\bigoplus_{i=1}^{m} p_{m,i}(x) \equiv 0 \bmod \tau(x)$.*
2. *For $1 \leq i < j \leq m$, $(p_{m,i}(x) \oplus p_{m,j}(x)) \not\equiv 0 \bmod \tau(x)$.*

From the definition, it is clear that for an allowed sequence to exist, we must have $m \geq 3$. The parameter m in the specification of PEP will represent the number of blocks to be encrypted (or decrypted). Later we show how to easily define such an allowed sequence. Since $\tau(x)$ is fixed, we will simply write "allowed sequence" instead of "allowed sequence with respect to $\tau(x)$".

The notation $\mathsf{bin}(m)$ denotes the n-bit binary representation of the integer m. For example, $\mathsf{bin}(1) = 0^{n-1}1$ and $\mathsf{bin}(2) = 0^{n-2}10$. The specification of PEP consists of three cases: $m = 1$; $m = 2$; and $m \geq 3$. The cases $m = 1$ and $m = 2$ are shown in Figure 2. Figure 3 shows the encryption of a 4-block message. The complete encryption and decryption algorithms are shown in Figure 1.

Remark: For decryption to be possible, we need $R(x)$ to have a multiplicative inverse modulo $\tau(x)$. Since $R(x)$ is a polynomial of degree at most $n-1$, the only value for which $R(x)$ does not have such an inverse is $R(x) = 0$. For such an R, the protocol is not defined. Note that $R = E_K(T)$. Assuming $E_K()$ to be a random permutation, the probability $R = 0$ is $1/2^n$. Since $n \geq 128$, the probability of getting a T for which the protocol is not defined is negligible.

Basic intuition behind the construction: The basic idea of the construction is to compute a polynomial hash (Wegman-Carter [15]) of the message. (Similar hashes are used in the GCM mode of operation [9] and HCTR.) This hash is the element MPP whose expression is the following.

$$MPP \quad = \quad EN \oplus \bigoplus_{i=1}^{m} P_i(x)R^{i-1}(x). \tag{1}$$

The mask M_1 is obtained by encrypting MPP. Since we are aiming at an SPRP, we should ensure that each ciphertext bit depends upon all the plaintext bits. To do this, the mask M_1 is "mixed" to the message blocks to obtain the PPP_i's. While doing this we must be careful. During decryption, we will obtain the PPP_i's after the decryption layer. Thus, we should be able to compute MPP from the PPP_i's. To ensure that this can be done, we do two things. First we convert the P_i's to PP_i's by multiplying with R^{i-1}. The second thing is to "distribute" M_1 among the PP_i's while obtaining the PPP_i's so as to ensure that

$$\bigoplus_{i=1}^{m} PPP_i = \bigoplus_{i=1}^{m} PP_i = \bigoplus_{i=1}^{m} P_i(x)R^{i-1}(x) = MPP \oplus EN. \tag{2}$$

This follows from the first property of allowed sequences, namely, $\bigoplus_{i=1}^{m} p_{m,i}(x) \equiv 0 \bmod \tau(x)$ and hence MPP can be computed either as $\bigoplus_{i=1}^{m} PP_i \oplus EN$ or as $\bigoplus_{i=1}^{m} PPP_i \oplus EN$. Since allowed sequences exist only for $m \geq 3$, we need to tackle the cases $m = 1$ and $m = 2$ separately. In the case $m = 1$, there is no requirement to "distribute" M_1 among the message blocks while in the case of $m = 2$, this is a bit tricky to do. In the last case, we distribute M_1 as $M_1 \oplus EN$ and $M_1 \oplus EEN$. Note that the XOR of these two elements is $EN \oplus EEN$ (and not 0). However, both EN and EEN can be computed from the tweak and the length of the message and hence $MPP = PP_1 \oplus PP_2 \oplus EN = PPP_1 \oplus PPP_2 \oplus EEN$ and can be computed both during encryption and decryption.

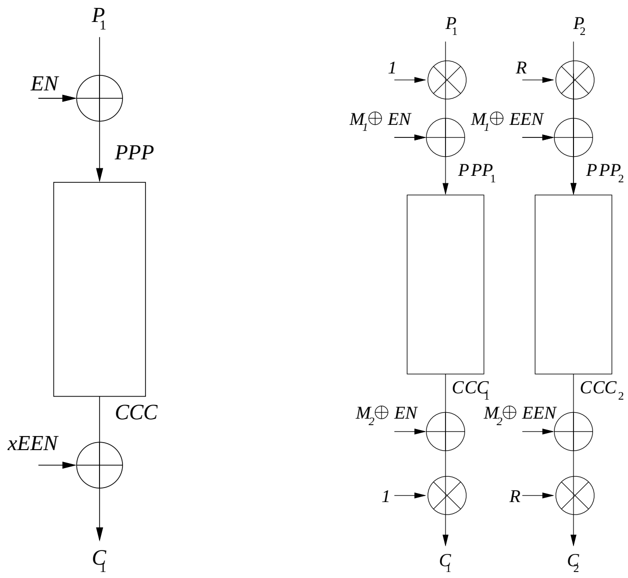
The computation after the encryption layer is similar to the computation before the encryption layer. This is because we are constructing an SPRP and the view from the decryption end will be similar to the view from the encryption end. In the decryption query, we work with $L(x) = R^{-1}(x)$ while computing the polynomial hash. This is required to ensure the consistency of decryption. This leads to decryption requiring one extra inversion operation making it slightly more costlier than encryption.

Figure 1: Encryption and Decryption using PEP

Algorithm $\mathbf{E}_K^T(P_1, P_2, \ldots, P_m)$
$R = E_K(T)$;
$EN = E_K(R \oplus \mathsf{bin}(m))$;
$EEN = E_K(xEN)$;
if $m == 1$, **then**
$PPP_1 = P_1 \oplus EN$;
$CCC_1 = E_K(PPP_1)$;
$C_1 = CCC_1 \oplus xEEN$;
return C_1;
endif

if $m == 2$, **then**
$PP_1 = P_1$; $PP_2 = R(x)P_2(x)$;
$MPP = PP_1 \oplus PP_2 \oplus EN$;
$M_1 = E_K(MPP)$;
$PPP_1 = PP_1 \oplus M_1 \oplus EN$;
$PPP_2 = PP_2 \oplus M_1 \oplus EEN$;
$CCC_1 = E_K(PPP_1)$; $CCC_2 = E_K(PPP_2)$;
$MCC = CCC_1 \oplus CCC_2 \oplus EN$;
$M_2 = E_K(MCC)$;
$CC_1 = CCC_1 \oplus M_2 \oplus EN$;
$CC_2 = CCC_2 \oplus M_2 \oplus EEN$;
$C_1 = CC_1$; $C_2 = R(x)CC_2(x)$;
return C_1, C_2;
endif
if $m \geq 3$, **then**
$R_1 = 1$; $PP_1 = P_1$; $MPP = PP_1$;
for $i = 2$ to m do
$R_i(x) = R(x)R_{i-1}(x)$;
$PP_i(x) = R_i(x)P_i(x)$;
$MPP = MPP \oplus PP_i$;
end for
$MPP = MPP \oplus EN$;
$M_1 = E_K(MPP)$; $MCC = 0^n$;
for $i = 1$ to m do
$PPP_i = PP_i \oplus p_{m,i}(x)M_1(x)$;
$CCC_i = E_K(PPP_i)$;
$MCC = MCC \oplus CCC_i$;
end for
$MCC = MCC \oplus EEN$;
$M_2 = E_K(MCC)$;
$CC_1 = CCC_1 \oplus p_{m,1}(x)M_2(x)$;
$R_1 = 1$; $C_1 = CC_1$;
for $i = 2$ to m do
$CC_i = CCC_i \oplus p_{m,i}(x)M_2(x)$;
$R_i(x) = R(x)R_{i-1}(x)$;
$C_i(x) = R_i(x)CC_i(x)$;
end for
return $C_1, C_2, \ldots, C_m$;
endif

Algorithm $\mathbf{D}_K^T(C_1, C_2, \ldots, C_m)$
$R = E_K(T)$;
$EN = E_K(R \oplus \mathsf{bin}(m))$;
$EEN = E_K(xEN)$;
if $m == 1$, **then**
$CCC_1 = C_1 \oplus xEEN$;
$PPP_1 = D_K(CCC_1)$;
$P_1 = PPP_1 \oplus EN$;
return P_1;
endif
$L(x) = R(x)^{-1}$;
if $m == 2$, **then**
$CC_1 = C_1$; $CC_2 = L(x)C_2(x)$;
$MCC = CC_1 \oplus CC_2 \oplus EEN$;
$M_2 = E_K(MCC)$;
$CCC_1 = CC_1 \oplus M_2 \oplus EN$;
$CCC_2 = CC_2 \oplus M_2 \oplus EEN$;
$PPP_1 = D_K(CCC_1)$; $PPP_2 = D_K(CCC_2)$;
$MPP = PPP_1 \oplus PPP_2 \oplus EEN$;
$M_1 = E_K(MPP)$;
$PP_1 = PPP_1 \oplus M_1 \oplus EN$;
$PP_2 = PPP_2 \oplus M_1 \oplus EEN$;
$P_1 = PP_1$; $P_2 = L(x)PP_2(x)$;
return P_1, P_2;
endif
if $m \geq 3$, **then**
$L_1 = 1$; $CC_1 = C_1$; $MCC = CC_1$;
for $i = 2$ to m do
$L_i(x) = L(x)L_{i-1}(x)$;
$CC_i = L_i(x)C_i(x)$;
$MCC = MCC \oplus CC_i$;
end for
$MCC = MCC \oplus EEN$;
$M_2 = E_K(MCC)$; $MPP = 0^n$;
for $i = 1$ to m do
$CCC_i = CC_i \oplus p_{m,i}(x)M_2(x)$;
$PPP_i = D_K(CCC_i)$;
$MPP = MPP \oplus PPP_i$;
end for
$MPP = MPP \oplus EN$;
$M_1 = E_K(MPP)$;
$PP_1 = PPP_1 \oplus p_{m,1}(x)M_1(x)$;
$L_1 = 1$; $P_1 = PP_1$;
for $i = 2$ to m do
$PP_i = PPP_i \oplus p_{m,i}(x)M_1$;
$L_i(x) = L(x)L_{i-1}(x)$;
$P_i(x) = L_i(x)PP_i(x)$;
end for
return $P_1, P_2, \ldots, P_m$;
endif



(a) (b)

Figure 2: (a) Enciphering one plaintext block with PEP, (b) Enciphering two plaintext blocks with PEP.

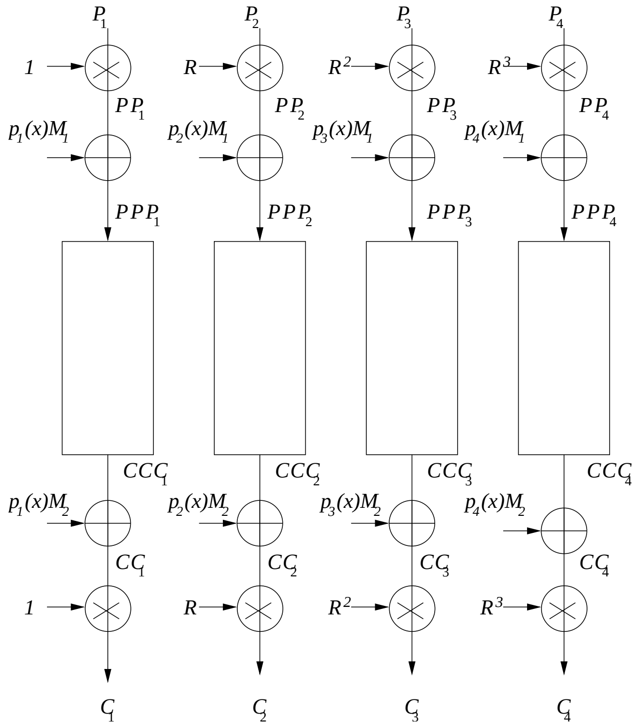


Figure 3: Enciphering four plaintext blocks with PEP. $R = E_K(T)$, $EN = E_K(R \oplus \mathsf{bin}_n(4))$, $EEN = E_K(xEN)$, $M_1 = E_K(\bigoplus_{i=1}^{4} R^{i-1}(x)PP_i(x) \oplus EN)$, $M_2 = E_K(\bigoplus_{i=1}^{4} CCC_i \oplus EEN)$.

2.1 Construction of Allowed Sequence of Polynomials

In this section, we provide one construction of allowed sequence. We do not claim this to be the only possibility; there may be others.

Let $\tau(x)$ be a primitive polynomial of degree n. Let $m_{\max}$ be a positive integer such that $\tau(x)$ does not divide any trinomial of degree less than $m_{\max}$. Estimates of $m_{\max}$ have been studied in the context of attacks on the nonlinear combiner model for stream ciphers [6]. (To use a primitive polynomial $\tau(x)$ in such stream ciphers, it is necessary that $\tau(x)$ does not divide a low degree trinomial.) This study indicates that $m_{\max}$ is around $2^{n/3}$. Given $\tau(x)$, there are algorithms for computing sparse multiples of $\tau(x)$. See [14] for a discussion on this issue.

We will be constructing an allowed sequence of length $m_{\max}$ and hence we will not be able to encrypt a message having more than $m_{\max}$ blocks. For $n = 128$, we have $2^{n/3} \approx 2^{42.6}$ and hence $m_{\max}$ is also around $2^{42.6}$. The ability to encrypt messages having at most $2^{42.6}$ many blocks is sufficient for all practical purposes.

Let $m = 3t \leq m_{\max}$, with $t \geq 1$. We define a sequence of polynomials $\tau_{3t,1}(x), \ldots, \tau_{3t,3t}(x)$ in the following manner.

$$\left.\begin{array}{lcll} \tau_{3t,i}(x) & = & x^i & \text{for } 1 \leq i \leq 2t; \\ \tau_{3t,2t+i}(x) & = & x^{2i-1} \oplus x^{2i} & \text{for } 1 \leq i \leq t. \end{array}\right\} \tag{3}$$

Let $3 \leq m \leq m_{\max}$. We define a sequence of polynomials $p_{m,1}(x), \ldots, p_{m,m}(x)$ in the following manner.

$m = 3t$: Define $p_{m,i}(x) = \tau_{m,i}(x)$.

$m = 3t + 1$: Define $p_{m,1} = 1 \oplus x$, $p_{m,2}(x) = x \oplus x^2$, $p_{m,3}(x) = x^2 \oplus x^3$, $p_{m,4}(x) = x^3 \oplus 1$ and $p_{m,4+i}(x) = x^3 \tau_{3(t-1),i}(x)$, for $1 \leq i \leq m$.

$m = 3t + 2$: Define $p_{m,1} = 1 \oplus x$, $p_{m,2}(x) = x \oplus x^2$, $p_{m,3}(x) = x^2 \oplus x^3$, $p_{m,4}(x) = x^3 \oplus x^4$, $p_{m,5}(x) = x^4 \oplus 1$ and $p_{m,5+i}(x) = x^4 \tau_{3(t-1),i}(x)$, for $1 \leq i \leq m$.

From the definition, note that for $m_1 \neq m_2$, we may have $p_{m_1,i}(x) \neq p_{m_2,i}(x)$. Later we show that due to its simple form, multiplication by $p_{m,i}(x)$ is quite efficient.

Proposition 1 *The sequence of polynomials $p_{m,1}(x), \ldots, p_{m,m}(x)$ is an allowed sequence.*

Proof : From the definition of $\tau_{3t,i}(x)$ it is easy to verify that $\bigoplus_{i=1}^{3t} \tau_{3t,i}(x) = 0$. From this again it is easy to see that $\bigoplus_{i=1}^{m} p_{m,i}(x) = 0$. This establishes the first condition for allowed sequences.

For the second condition, we must show that $p_{m,i}(x) \oplus p_{m,j}(x) \not\equiv 0 \bmod \tau(x)$. There are three cases to consider.

Both $p_{m,i}(x)$ and $p_{m,j}(x)$ are monomials: By construction the degrees of both $p_{m,i}(x)$ and $p_{m,j}(x)$ are at most $m_{\max} < 2^n - 1$. Hence, by the primitivity of $\tau(x)$ we have the required condition.

Both $p_{m,i}(x)$ and $p_{m,j}(x)$ are 2-nomials: By construction $p_{m,i}(x) = x^{i_1} \oplus x^{i_1+1}$ and $p_{m,j}(x) = x^{j_1} \oplus x^{j_1+1}$. Assume without loss of generality that $i_1 < j_1$. Then, $p_{m,i}(x) \oplus p_{m,j}(x) = x^{i_1}(1 \oplus x)(1 \oplus x^{j_1-i_1})$. Again, the primitivity of $\tau(x)$ ensures the required condition.

One of $p_{m,i}(x)$ and $p_{m,j}(x)$ is a monomial and the other is a 2-nomial: In this case, $p_{m,i}(x) \oplus p_{m,j}(x)$ is a trinomial of degree at most $m_{\max}$. The definition of $m_{\max}$ ensures the required condition.

This completes the proof. ∎

2.2 Computation of $p_{m,i}(x)M_1(x)$

In the encryption and decryption algorithms, we need to compute $p_{m,i}(x)M_1(x)$ and $p_{m,i}(x)M_2(x)$. We show how these may be efficiently computed. For this it is sufficient to show how to efficiently compute $\tau_{3t,i}(x)M_1$ and $\tau_{3t,i}(x)M_2$.

The polynomials $\tau_{3t,i}(x)$ satisfy the following recurrences:

$$\begin{array}{rcll} \tau_{3t,i}(x) & = & x\tau_{3t,i-1}(x) & \text{for } 2 \leq i \leq 2t; \\ \tau_{3t,2t+1} & = & \tau_{3t,1}(x) \oplus \tau_{3t,2}(x); & \\ \tau_{3t,i}(x) & = & x^2\tau_{3t,i-1}(x) & \text{for } 2t+2 \leq i \leq 3t. \end{array}$$

Define $M_{1,3t,i} = \tau_{3t,i}(x)M_1$. Then using the above recurrences, we have

$$\begin{array}{rcl} M_{1,3t,i}(x) & = & xM_{1,3t,i-1}\text{for } 2 \leq i \leq 2t; \\ M_{1,3t,2t+1} & = & M_{1,3t,1}(x) \oplus M_{1,3t,2}; \\ M_{1,3t,i}(x) & = & x^2M_{1,3t,i-1}\text{for } 2t+2 \leq i \leq 3t. \end{array}$$

Using these recurrences, it is easy to compute all the $\tau_{3t,i}(x)M_1$'s; the requirement is to multiply by either x or x^2 and perform a bitwise XOR. Multiplying by x and x^2 is much more efficient than a general multiplication modulo $\tau(x)$. A similar computation will yield the products $\tau_{3t,i}(x)M_2$.

3 Features of PEP

Here we discuss some of the important features and limitations of PEP.

Message Length: PEP does not produce any ciphertext expansion as it does not produce any tag. The tweak is not considered to be a part of the ciphertext. This is similar to CMC, EME, EME* and HCTR. The current version of PEP can only handle messages whose length is a multiple of n. This is similar to CMC. EME can handle messages of lengths mn, with $1 \leq m \leq n$, while EME* and HCTR can handle messages of lengths $\geq n$. Modification of PEP to handle messages of lengths $\geq n$ is a future task.

There is a (theoretical) restriction on the number of blocks in a particular message to be encrypted by PEP. The number of blocks in any one message to be encrypted by PEP is at most $m_{\max}$. For $n = 128$, this implies that a single message can contain at most around $2^{42.6}$ blocks, which is sufficient for all practical purposes (see Section 2.1).

Single Block Cipher Key: PEP uses the same key for all the block cipher calls. CMC and EME require a single key. On the other hand, EME* requires three keys and HCTR requires two keys. A single block cipher key saves key storage space and key setup costs.

Tweak: Encryption under PEP requires an n-bit tweak. The tweak need not be random, unpredictable or secret. The adversary is allowed to repeat a tweak in queries to the encryption and decryption oracles. The tweak is required for decryption and hence has to be available at both the receiver and the sender ends. This may be achieved by maintaining a shared counter between the two parties or it may be such that it can be understood from the context. An example of the later is the sector address in disk encryption applications.

Online/Offline: An encryption scheme is called online if it can output a stream of ciphertext bits as a stream of plaintext bits arrive. PEP is not online. PEP incorporates the effect of the whole plaintext on each ciphertext bit. Hence, construction of PEP does not allow it to output a single block of ciphertext unless it has seen the total message. We note that no construction of SPRP (tweakable or otherwise) can be online for the same reason as PEP.

Consider the encryption algorithm of PEP for $m \geq 3$. The algorithm consists of three separate **for** loops. The first loop computes the PP_i's and the quantity $\bigoplus_{i=1}^{m} R_i(x)P_i(x)$. The values of the PP_i's need to be stored for use by the second loop. The second loop computes the PPP_i's and the CCC_i's and also the quantity $\bigoplus_{i=1}^{m} CCC_i$. The PPP_i's do not need to be stored but the value of the CCC_i's need to be stored for use by the third loop. The third loop produces the CC_i's and the C_i's which completes the encryption. (Decryption also has a similar structure).

To summarize, the algorithm makes a pass over the P_i's to produce the PP_i's which are stored; makes a pass over the PP_i's to produce the CCC_i's which are also stored; and finally makes a pass over the CCC_i's to produce the C_i's. This makes it a three pass algorithm. Note that the PP_i's can be written over the P_i's and the CCC_i's can be written over the PP_i's. Thus, the intermediate quantities PP_i's and the CCC_i's do not require any extra storage.

HCTR is also a hash-encrypt-hash type construction and requires three passes for reasons similar to that of PEP. The algorithmic descriptions of CMC, EME and EME* as given in the respective papers suggest these algorithms to be three-pass algorithms. On the other hand, a careful consideration of the algorithms show that all of these algorithms can be implemented using two passes over the data. Basically, these algorithms are of the form encrypt-mask-encrypt. The first encryption layer needs to be completed in one pass over the data. Then the mask is computed. The actual masking of the intermediate values and the second encryption layer can be combined in a single pass.

Note that any algorithm requiring more than one pass cannot be online and also at least one set of intermediate variables need to be stored. If we overwrite the P_i's then no extra storage is required for either two or three pass algorithms. On the other hand, if we wish to preserve the P_i's, then the same amount of extra space is required by both two and three pass algorithms. Further, a two pass algorithm is not necessarily more efficient than a three pass algorithm. We have to compare the total amount of computation done by the two algorithms to determine relative efficiency. We consider this issue next.

Computation Cost: PEP performs two polynomial hashes and one layer of block cipher encryption. The total number of block cipher encryptions for an m-block message is 4 if $m = 1$; and is $m + 5$ if $m \geq 2$. The polynomial hashes are used to compute MPP and MCC for $m \geq 2$, the two computations being similar.

To compute MPP we have to compute $\bigoplus_{i=1}^{m} P_i(x)R^{i-1}(x)$. Using Horner's rule this can be computed using $(m-1)$ multiplications over $GF(2^n)$. But Horner's rule does not compute the values $PP_i = P_i(x)R^{i-1}(x)$. Since PP_i's are also required, using Horner's rule does not help. We have to compute $R^2(x), \ldots, R^{m-1}(x)$ and $P_2(x)R(x), \ldots, P_m R^{m-1}(x)$. These require a total of $2m-3$ multiplications in $GF(2^n)$. Similarly, a total of $2m-3$ multiplications are required for computing MCC and the CC_i's. Hence, for $m \geq 2$, PEP requires a total of $4m-6$ finite field multiplications. In addition, there are the multiplications of the type $p_{m,i}(x)M_1$ and $p_{m,i}(x)M_2$. But as discussed in Section 2.2, these can be computed very efficiently.

If sufficient memory is available, then the values $R^2(x), \ldots, R^{m-1}(x)$ computed during the computation of MPP and the PP_i's can be stored and used during the computation of MCC and the CC_i's. (This can also be combined with the parallel computation strategy discussed below.) This brings down the total number of multiplications to $3m-4$.

We note that decryption requires the computation of one finite field inverse. The cost of this is amortized over the entire computation and will not reflect on the overall cost if m is moderately large. The main cost will be that for hardware implementation since we will have to implement a finite field inverter requiring more chip area.

Parallelism: The encryption layer is fully parallelizable though the computations of R, EN, EEN, M_1 and M_2 has to be sequential. Thus, for $m \geq 2$, we require at least six parallel encryption rounds irrespective of the number of available block cipher units – five for computing the above quantities and at least one for encrypting the PPP_i's.

The computation of the PP_i's and MPP can be parallelized in the following manner. We illustrate by an example. Suppose there are 4 finite field multipliers available. In the first step, the quantity R^2 is computed. Now consider the following parallel schedule for the four multipliers.

	multiplier 1	multiplier 2	multiplier 3	multiplier 4
Round 1	$(P_2(x) * R(x))$	$(P_3(x) * R^2(x))$	$(R(x) * R^2(x))$	$(R^2(x) * R^2(x))$
Round 2	$(P_4(x) * R^3(x))$	$(P_5(x) * R^4(x))$	$(R^3(x) * R^2(x))$	$(R^4(x) * R^2(x))$
Round 3	$(P_6(x) * R^5(x))$	$(P_7(x) * R^6(x))$	$(R^5(x) * R^2(x))$	$(R^6(x) * R^2(x))$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

Using this schedule, all the four multipliers can be kept busy in all the rounds (except for the last round and the initial computation of R^2). A similar schedule can be built for computing MCC and the CC_i's. In general using κ many multipliers, all the multiplications can be completed in approximately $\lceil (4m-6)/\kappa \rceil$ many parallel rounds which is optimal for κ many multipliers.

HCTR uses a polynomial hash which can be evaluated by Horner's rule. On the other hand, there is no straightforward way of parallelizing the polynomial computation *without* increasing the total number of multiplications. The approach described above can be used to obtain parallel implementation of polynomial evaluation and this would double the number of multiplications required in evaluating using Horner's rule.

Provable Security: PEP is provably secure. We state a theorem related to the security of PEP in Section 4 and provide the proof in Section A. The concrete security bound that we obtain for PEP is similar to that obtained for CMC, EME and EME* and is as expected for a mode of operation. Loosely speaking, the theorem shows that the advantage of an adversary in attacking PEP[E] as a tweakable SPRP is bounded above by the advantage of an adversary in attacking E as an SPRP plus an additive factor which is approximately equal to $c\sigma_n^2/2^n$, where c is a constant and σ_n is the total number of blocks (plaintext or ciphertext) provided by the adversary in its queries to the encryption and the decryption oracles. The security bound of HCTR is considerably weaker than the other modes of operations including PEP. For HCTR, the quantity $c\sigma_n^2/2^n$ is replaced by $c\sigma_n^3/2^n$, i.e., there is a cubic security degradation rather than the usual quadratic degradation. One consequence of a weaker security bound is that the secret keys need to changed much earlier compared to the other modes.

Comparison: Table 1 provides a comparison of the various features of PEP and other modes of operations which are SPRPs. We make the following points based on Table 1.

HCTR is the only previously known fully specified tweakable SPRP which is of the hash-encrypt-hash type. The NR construction is too incomplete (and also not tweakable) to permit a proper comparison to other constructions. The main drawback of HCTR is its weaker security bound and the requirement of two keys. It has other good features such as ability to handle all message lengths $\geq n$, and lower computation cost. The encryption layer of HCTR is fully parallel. The two hash layers can be implemented in parallel with computation cost similar to that of PEP.

PEP is currently the only known single key, efficiently parallelizable algorithm which can handle arbitrary number of n-bit blocks. CMC, EME* and HCTR can handle such messages, but CMC is strictly sequential; EME* requires one key from $\mathcal{K}$ plus two other n-bit keys; and HCTR requires one key from $\mathcal{K}$ and one n-bit key. On the other hand, EME uses a single key and is fully parallel but can handle at most n many n-bit blocks.

A general comparison of the computation cost of PEP and HCTR with the other modes (CMC, EME and EME*) is difficult. This is because in one case we have less block cipher invocations but $GF(2^n)$ multiplications whereas in the other case we have more block cipher invocations and no $GF(2^n)$ multiplications. Consequently, the comparison depends upon several factors such as:

1. The implementation platform – hardware or software. For software implementation, we need to consider the target architecture and its support for $GF(2^n)$ multiplication. Availability of cryptographic co-processors may provide substantial support for such operation. For hardware implementation, the number of available parallel encryption units and $GF(2^n)$ multiplication units need to be considered.

2. The most important consideration is the design of the actual block cipher being used with the mode of operation *and* its efficient implementation.

Note that PEP approximately trades one block cipher call for four multiplications (HCTR trades one [BC] for two [M]). Implementation results from [9] suggest that using a look-up table it is possible to complete a few $GF(2^n)$ multiplications in the time required for one AES-128 invocation. A more detailed software speed comparison requires efficient implementation of the various modes of operation and can be a topic of future study.

A mode of operation is not designed for use with any particular block cipher. As a side remark, we would like to note that the FIPS 197 specifies AES for protecting "sensitive (unclassified) information". So each government might be having its own block cipher for protecting classified information. The computation cost of PEP vis-a-vis the other modes with respect to such ciphers has to be determined on a case-to-case basis.

The number of passes made by PEP and HCTR is one more than the other modes (though the number of encryption layers is one less). We do not consider this to be a serious problem since all the modes require at least two passes and cannot perform online encryption and decryption. The additional pass of PEP and HCTR by itself does not by itself lead to any efficiency degradation.

Table 1: Comparison of SPRPs using an n-bit block cipher, an n-bit tweak and for m message blocks. (We assume $m \geq 3$.) [BC]: one block cipher invocation; [M]: one $GF(2^n)$ multiplication.

Mode	sec. bnd.	computation cost	keys	msg. len.	passes	enc. layers	parallel?
CMC	$\sigma_n^2/2^n$	$(2m+1)$[BC]	1	mn, $m \geq 1$	2	2	No
EME	$\sigma_n^2/2^n$	$(2m+2)$[BC]	1	mn, $1 \leq m \leq n$	2	2	Yes
EME*	$\sigma_n^2/2^n$	$(2m+\frac{m}{n}+1)$[BC]	3	$\geq n$	2	2	Yes
HCTR	$\sigma_n^3/2^n$	m[BC] $+2(m+1)$[M]	2	$\geq n$	3	1	partial
PEP	$\sigma_n^2/2^n$	$(m+5)$[BC] $+(4m-6)$[M]	1	mn, $m \geq 1$	3	1	Yes

4 Security of PEP

4.1 Definitions and Notation

As mentioned earlier, an n-bit block cipher is a function $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, where $\mathcal{K} \neq \emptyset$ is called the key space and for any $K \in \mathcal{K}$, $E(K,.)$ is a permutation. We will usually write $E_K(\,)$ instead of $E(K,.)$.

An adversary A is a probabilistic algorithm which has access to some oracles and which outputs either 0 or 1. Oracles are written as superscripts. The notation $A^{\mathcal{O}_1,\mathcal{O}_2} \Rightarrow 1$ denotes the event that the adversary A, interacting with the oracles $\mathcal{O}_1, \mathcal{O}_2$, finally outputs the bit 1.

Let $\mathrm{Perm}(n)$ denote the set of all permutations on $\{0,1\}^n$. We require $E(,)$ to be a strong pseudorandom permutation. The advantage of an adversary in breaking the strong pseudorandomness of $E(,)$ is defined in the following manner.

$$\mathbf{Adv}_E^{\pm\mathrm{prp}}(A) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{E_K(),E_K^{-1}()} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\pi(),\pi^{-1}()} \Rightarrow 1\right].$$

Formally, a tweakable enciphering scheme is a function $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$, where $\mathcal{K} \neq \emptyset$ and $\mathcal{T} \neq \emptyset$ are the key space and the tweak space respectively and $\mathcal{M} = \cup_{i \geq 1}\{0,1\}^{ni}$, where n is the length of a message block. We shall often write $\mathbf{E}_K^T(.)$ instead of $\mathbf{E}(K,T,.)$. The inverse of an enciphering scheme is $\mathbf{D} = \mathbf{E}^{-1}$ where $X = \mathbf{D}_K^T(Y)$ if and only if $\mathbf{E}_K^T(X) = Y$.

Let $\mathrm{Perm}^{\mathcal{T}}(\mathcal{M})$ denote the set of all functions $\boldsymbol{\pi} : \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ where $\boldsymbol{\pi}(\mathcal{T},.)$ is a length preserving permutation. Such a $\boldsymbol{\pi} \in \mathrm{Perm}^{\mathcal{T}}(\mathcal{M})$ is called a tweak indexed permutation. For a tweakable enciphering scheme $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$, we define the advantage an adversary A has in distinguishing $\mathbf{E}$ and its inverse from a random tweak indexed permutation and its inverse in the following manner.

$$\mathbf{Adv}_{\mathbf{E}}^{\pm\widetilde{\mathrm{prp}}}(A) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right].$$

Pointless queries: An adversary never queries its deciphering oracle with (T,C) if it got C in response to an encipher query (T,M) for some M. Similarly, the adversary never queries its enciphering oracle with (T,M) if it got M as a response to a decipher query of (T,C) for some C.

These queries are called *pointless* as the adversary knows what it would get as response for such queries.

Following [4], we define the query complexity σ_n of an adversary as follows. A string X contributes $\max(|X|/n, 1)$ to the query complexity. A tuple of strings $(X_1, X_2, \ldots)$ contributes the sum of the contributions from all oracle queries plus the contribution from the adversary's output. Suppose an adversary makes q queries where the number of n-bit blocks in the ith query is ℓ_i. Then, $\sigma_n = 1 + \sum_{i=1}^{q}(1+\ell_i) \geq 2q$. Let ρ be a list of resources used by the adversary A and suppose $\mathbf{Adv}_{\pi}^{\pm\mathrm{xxx}}(A)$ has been defined where π is either a block cipher or a tweakable encipering scheme. $\mathbf{Adv}_{\pi}^{\pm\mathrm{xxx}}(\rho)$ denotes the maximal value of $\mathbf{Adv}_{\pi}^{\pm\mathrm{xxx}}(A)$ over all adversaries A using resources at most ρ. Usual resources of interest are the running time t of the adversary, the number of oracle queries q made by the adversary and the query complexity σ_n ($n \geq 1$).

The notation PEP[E] denotes a tweakable enciphering scheme, where the n-bit block cipher E is used in the manner specified by PEP. Our purpose is to show that PEP[E] is secure if E is secure. The notation PEP[Perm(n)] denotes a tweakable enciphering scheme obtained by plugging in a random permutation from Perm(n) into the structure of PEP. For an adversary attacking PEP[Perm(n)], we do not put any bound on the running time of the adversary, though we still put a bound on the query complexity σ_n. We show the information theoretic security of PEP[Perm(n)] by obtaining an upper bound on $\mathbf{Adv}_{\mathrm{PEP}[\mathrm{Perm}(n)]}^{\pm\widetilde{\mathrm{prp}}}(q, \sigma_n)$. The upper bound is obtained in terms of n and σ_n. For a fixed block cipher E, we bound $\mathbf{Adv}_{\mathrm{PEP}[E]}^{\pm\widetilde{\mathrm{prp}}}(q, \sigma_n, t)$ in terms of $\mathbf{Adv}_{E}^{\pm\mathrm{prp}}(q, \sigma_n, t')$, where $t' = t + O(\sigma_n)$. We will use the notation $\mathbf{E}_\pi$ as a shorthand for PEP[Perm(n)] and $\mathbf{D}_\pi$ will denote the inverse of $\mathbf{E}_\pi$. Thus, the notation $A^{\mathbf{E}_\pi, \mathbf{D}_\pi}$ will denote an adversary interacting with the oracles $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$.

4.2 Statement of Result

The following theorem specifies the security of PEP.

Theorem 2 *Fix n, q and $\sigma_n \geq q$ to be positive integers and an n-bit block cipher $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$. Then*

$$\mathbf{Adv}_{\mathrm{PEP}[\mathrm{Perm}(n)]}^{\pm\widetilde{\mathrm{prp}}}(q, \sigma_n) \leq \frac{1}{2^{n+1}} \times \left(q^2 + 6(5q + \sigma_n)^2\right) \tag{4}$$

$$\mathbf{Adv}_{\mathrm{PEP}[E]}^{\pm\widetilde{\mathrm{prp}}}(q, \sigma_n, t) \leq \frac{1}{2^{n+1}} \times \left(q^2 + 6(5q + \sigma_n)^2\right) + \mathbf{Adv}_{E}^{\pm\mathrm{prp}}(q, \sigma_n, t') \tag{5}$$

where $t' = t + O(\sigma_n)$.

Since each query consists of at least one n-bit block, we have $q \leq \sigma_n$ and hence we could write $(q^2 + 6(5q + \sigma_n)^2) \leq c\sigma_n^2$ for some constant c. Upper bounding q by σ_n is proper when σ_n and q are comparable, i.e., when each query consists of a few blocks. On the other hand, if each query consists of a large number of blocks, the bound $q \leq \sigma_n$ is very loose and replacing q by σ_n makes the bound appear worse than what it really is. Hence, we choose to present the bound in terms of both q and σ_n.

The above result and its proof is similar to previous work (see for example [3, 4, 13]). As mentioned in [3], Equation (5) embodies a standard way to pass from the information theoretic setting to the complexity theoretic setting. Let $\mathbf{E}(.,.,.)$ denote PEP[E]. For any adversary A, we have the following.

$$\mathbf{Adv}_{\mathrm{PEP}[E]}^{\pm\widetilde{\mathrm{prp}}}(A) \quad = \quad \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]$$

$$
\begin{aligned}
&= \left(\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right) \\
&\quad + \left(\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right. \\
&\quad \left. - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right) \\
&= X + \mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(A)
\end{aligned}
$$

where $X = \left(\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathbf{E}_K(.,.),\mathbf{E}_K^{-1}(.,.)} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right)$. The quantity X represents an adversary's advantage in distinguishing PEP[E] from PEP[π], where π is a randomly chosen permutation from Perm(n). Clearly, such an adversary A can also distinguish E from a random permutation and hence $X \le \mathbf{Adv}^{\pm\mathrm{prp}}_{E}(A)$. This argument shows how (4) is obtained from (5).

We need to consider an adversary's advantage in distinguishing a tweakable enciphering scheme $\mathbf{E}$ from an oracle which simply returns random bit strings. This advantage is defined in the following manner.

$$
\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{MEM}[\mathrm{Perm}(n)]}(A) = \Pr[\pi \xleftarrow{\$} \mathrm{Perm}[n] : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1] - \Pr[A^{\$(.,.),\$(.,.)} \Rightarrow 1]
$$

where $\$(.,.)$ returns random bits of length $|M|$. The basic idea of proving (4) is as follows.

$$
\begin{aligned}
\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(A) &= \left(\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right]\right. \\
&\quad \left. - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right) \\
&= \left(\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1\right] - \Pr\left[A^{\$(.,.),\$(.,.)} \Rightarrow 1\right]\right) \\
&\quad + \left(\Pr\left[A^{\$(.,.),\$(.,.)} \Rightarrow 1\right] - \Pr\left[\boldsymbol{\pi} \xleftarrow{\$} \mathrm{Perm}^{\mathcal{T}}(\mathcal{M}) : A^{\boldsymbol{\pi}(.,.),\boldsymbol{\pi}^{-1}(.,.)} \Rightarrow 1\right]\right) \\
&\le \mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(A) + \binom{q}{2}\frac{1}{2^n} \qquad (6)
\end{aligned}
$$

where q is the number of queries made by the adversary. For a proof of the last inequality see [4].

The main task of the proof now reduces to obtaining an upper bound on $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(\sigma_n)$. This proof is provided in Section A. The proof uses the standard technique of sequence of games between an adversary and the mode of operation PEP. The proof is similar to the corresponding proofs of CMC [3] and EME [4]. By a sequence of games we show that if in response to any valid query of the adversary, random strings of appropriate lengths are returned then the internal computations of PEP can be performed consistently under the assumption that the block cipher and its inverse are random permutations. The crux of the proof lies in showing that there would seldom be any collisions in the range and domain sets of the block cipher if the adversary queries PEP with valid queries and PEP responds to them by producing random strings. In a later part we remove the randomness associated with the adversary and the game runs on a fixed transcript consisting of the queries and their responses. We show that in such a situation also the internal computations of PEP can be performed consistently.

We later prove (see (18)) that for any adversary making q queries and having query complexity σ_n

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(q, \sigma_n) \leq \frac{3(5q + \sigma_n)^2}{2^n}.$$

Using this and (6), we obtain

$$\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(q, \sigma_n) \quad \leq \quad \frac{1}{2^{n+1}} \times \left(q^2 + 6(5q + \sigma_n)^2\right).$$

5 Conclusion

We have presented a new construction for a tweakable SPRP called PEP. Our approach is to use polynomial hash, followed by an encryption layer and again followed by a polynomial hash. This is different from the other constructions of tweakable SPRPs, namely CMC, EME and EME* and is similar to the approach for constructing SPRP (not tweakable SPRP) given in [10]; this approach has also been used in constructing HCTR. PEP offers certain advantages over the known tweakable SPRPs – it is the only know construction which uses a single key, is efficiently parallelizable and can handle an arbitrary number of blocks. We make a detailed comparison between the known constructions of tweakable SPRPs which show that PEP compares quite favourably

References

[1] M. Bellare and P. Rogaway, Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography, Advances in Cryptology - Asiacrypt 2000, LNCS 1976, pp. 317-330, Springer, 2000.

[2] S. Halevi, EME*. Extending EME to handle arbitrary-length messages with associated data. INDOCRYPT 2004, pp. 315-327, Springer 2004

[3] S. Halevi and P. Rogaway. A tweakable enciphering mode, Advances in Cryptology - CRYPTO 2003, LNCS, vol. 2729, pp. 482-499, Springer, 2003.

[4] S. Halevi and P. Rogaway. A parallelizable enciphering mode, Topics in Cryptology, CT-RSA 2004, LNCS, vol. 2964, pp. 292-304, Springer, 2004

[5] C. S. Jutla: Encryption modes with almost free message integrity. EUROCRYPT 2001: 529-544.

[6] S. Maitra, K. C. Gupta and A. Venkateswarlu. Results on multiples of primitive polynomials and their products over $GF(2)$. *Theoretical Computer Science* 341(1-3): 311-343 (2005).

[7] M. Liskov, R. L. Rivest and D. Wagner. Tweakable block ciphers. CRYPTO 2002: 31-46.

[8] M. Luby and C. Rackoff, How to construct pseudo-random permutations and pseudo-random functions, *SIAM Journal of Computing*, vol. 17, pp. 373-386, 1988.

[9] D. A. McGrew and J. Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. Proceedings of Indocrypt 2004, 343-355.

[10] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited, *J. of Cryptology*, vol 12, pp. 29-66, 1999.

[11] M. Naor and O. Reingold. A pseudo-random encryption mode. Manuscript available from www.wisdom.weizmann.ac.il/̃naor.

[12] P. Rogaway. Nonce-based symmetric encryption, Fast Software Encryption (FSE) 2004, LNCS 3017, pp. 348-359, Springer, 2004.

[13] P. Rogaway, M. Bellare and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Conference on Computer and Communication Security 2001: 196-205.

[14] D. Wagner. A generalized birthday problem. CRYPTO 2002, LNCS 2442, pp. 288–303, Springer 2002.

[15] M. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, vol. 22, 1981, pp. 265–279.

[16] P. Wang, D. Feng and W. Wu. HCTR: A variable-input-length enciphering mode. CISC 2005, LNCS 3822, pp. 175–188, 2005.

A Upper Bound on $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(q, \sigma_n)$

We model the adversary's interaction with the oracles $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$ as a game. In the usual game, which we call PEP1, the adversary submits queries to $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$ and gets appropriate answers. Starting from this game, we modify it in successive steps to obtain games where the adversary is provided random bit strings of appropriate lengths. This results in a sequence of games: PEP1, RAND1, RAND2, RAND3 and NON.

By an abuse of notation, we will use A^{PEP1} to denote an adversary A's interaction with the oracles while playing game PEP1. We will use similar notations for the other games. In what follows, by $X \xleftarrow{\$} \{0,1\}^n$, we will denote the act of choosing a random n-bit string and assigning it to the variable X.

A.1 The Game Sequence

Game PEP1**:** We describe the attack scenario of the adversary A through a probabilistic game which we call PEP1. In PEP1, the adversary interacts with $\mathbf{E}_\pi$ and $\mathbf{D}_\pi$ where π is a randomly chosen permutation from $\mathrm{Perm}(n)$. Instead of initially choosing π, we build up π in the following manner.

Initially π is assumed to be undefined everywhere. When $\pi(X)$ is needed, but the value of π is not yet defined at X, then a random value is chosen among the available range values. Similarly when $\pi^{-1}(Y)$ is required and there is no X yet defined for which $\pi(X) = Y$, we choose a random value for $\pi^{-1}(Y)$ from the available domain values.

The domain and range of π are maintained in two sets *Domain* and *Range*, and $\overline{Domain}$ and $\overline{Range}$ are the complements of *Domain* and *Range* relative to $\{0,1\}^n$. The game PEP1 is shown in Figure 4. The figure shows the subroutines Ch-π, Ch-π^{-1}, the initialization steps and how the game responds to a encipher/decipher query of the adversary. The i^{th} query of the adversary depends on its previous queries, the responses to those queries and on some coins of the adversary.

The game PEP1 accurately represents the attack scenario, and by our choice of notation, we can write

$$\Pr[A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1] = \Pr[A^{\mathrm{PEP1}} \Rightarrow 1]. \tag{7}$$

Game RAND1**:** We modify PEP1 by deleting the boxed entries in PEP1 and call the modified game as RAND1. By deleting the boxed entries it cannot be guaranteed that π would be a permutation as though we do the consistency checks but we do not reset the values of Y (in Ch-π) and X (in Ch-π^{-1}). Thus, the games PEP1 and RAND1 are identical apart from what happens when the bad flag is set. So,

$$\Pr[A^{\mathrm{PEP1}} \Rightarrow 1] - \Pr[A^{\mathrm{RAND1}} \Rightarrow 1] \leq \Pr[A^{\mathrm{RAND1}} \text{ sets } \mathsf{bad}] \tag{8}$$

Game RAND2**:** We make certain changes to the game RAND1 which are invisible to the adversary. In RAND1 as the permutation π is not maintained, thus the subroutine Ch-π and Ch-π^{-1} are no more needed. Instead we add a new subroutine called Check-Domain-Range(X, Y). In Check-Domain-Range(X, Y), X is inserted into *Domain* and Y is inserted into *Range*; also, if $X \in \mathit{Domain}$ or $Y \in \mathit{Range}$ then the bad flag is set.

In this game, for an encryption query, we choose the ciphertext blocks to be random n-bit strings and return to the adversary. Then we adjust the internal variables so as to ensure that the particular choice of ciphertext blocks is consistent as per the protocol. Similarly, for a decryption query, we choose the plaintext blocks to be random n-bit strings and return to the adversary and then adjust the internal variables. This does not alter the adversary's view of the game since for each such change the adversary obtains a random n-bit string both before and after the change. Thus,

$$\Pr[A^{\mathrm{RAND1}} \Rightarrow 1] = \Pr[A^{\mathrm{RAND2}} \Rightarrow 1] \tag{9}$$

also,

$$\Pr[A^{\mathrm{RAND1}} \text{ sets } \mathsf{bad}] = \Pr[A^{\mathrm{RAND2}} \text{ sets } \mathsf{bad}] \tag{10}$$

In RAND2 the adversary is supplied with random bits as response to queries to both the encrypt and the decrypt oracles. Hence,

$$\Pr[A^{\mathrm{RAND2}} \Rightarrow 1] = \Pr[A^{\$(.,.),\$(.,.)} \Rightarrow 1] \tag{11}$$

Now, from Equation (7), (8), (9), (10) and (11) we get

$$\begin{aligned}
\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(A) &= \Pr[A^{\mathbf{E}_\pi,\mathbf{D}_\pi} \Rightarrow 1] - \Pr[A^{\$(.,.),\$(.,.)} \Rightarrow 1] && (12)\\
&= \Pr[A^{\mathrm{PEP1}} \Rightarrow 1] - \Pr[A^{\mathrm{RAND2}} \Rightarrow 1]\\
&= \Pr[A^{\mathrm{PEP1}} \Rightarrow 1] - \Pr[A^{\mathrm{RAND1}} \Rightarrow 1]\\
&\leq \Pr[A^{\mathrm{RAND1}} \text{ sets } \mathsf{bad}]\\
&= \Pr[A^{\mathrm{RAND2}} \text{ sets } \mathsf{bad}] && (13)
\end{aligned}$$

Our task is thus to bound $\Pr[A^{\mathrm{RAND2}}$ sets bad$]$.

Game RAND3**:** Here we make two subtle changes to the game RAND2. Here instead of the *Domain* and *Range* sets we use multisets $\mathcal{D}$ and $\mathcal{R}$ respectively. In the game RAND3 on either an encryption or a decryption query by the adversary a random string is given as output. Next, the internal variables are adjusted in the first phase of the finalization step. The bad flag is set at the second phase of the finalization step by checking whether a value occurs in either $\mathcal{R}$ or $\mathcal{D}$ more than once. The game RAND3 is shown in Figure 6. RAND3 sets bad in exactly the same conditions in which RAND2 sets bad, hence

$$\Pr[A^{\mathrm{RAND2}} \text{ sets } \mathsf{bad}] = \Pr[A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}]. \tag{14}$$

Game NON**:** In this game we get rid of the adversary and its interactions. We want to bound $\Pr[A^{\mathrm{RAND3}}$ sets $\mathsf{bad}]$, we assume that the adversary is deterministic and so the probability is over the random choices of $P \xleftarrow{\$} \{0,1\}^{nm_s}$ and $C \xleftarrow{\$} \{0,1\}^{nm_s}$ and the other random choices made during the finalization steps but not on the private coins of the adversary.

In the previous games, for an encipher query, the adversary specified the tweak and the plaintext; and for a decipher query, he specified the tweak and the ciphertext. We now consider the stronger condition, whereby the adversary specifies the tweak, the plaintext and the ciphertext in both the encryption and the decryption queries. Even under this condition, we would like to show that the flag is rarely set to bad. We do this by modifying the game RAND3 into a new game NON (non-interactive). NON depends on a fixed transcript $\mathsf{tr} = (\mathbf{ty}, \mathbf{T}, \mathbf{P}, \mathbf{C})$ with $\mathbf{ty} = (ty^1, ty^2, \ldots, ty^q)$, $\mathbf{T} = (\mathsf{T}^1, \mathsf{T}^2, \ldots \mathsf{T}^q)$, and $\mathbf{P} = (\mathsf{P}^1, \mathsf{P}^2, \ldots \mathsf{P}^q)$, and $\mathbf{C} = (\mathsf{C}^1, \mathsf{C}^2, \ldots \mathsf{C}^q)$ where $ty^s = \{\mathsf{Enc}, \mathsf{Dec}\}$, $\mathsf{T}^s \in \{0,1\}^n$ and $\mathsf{P}^s = \mathsf{P}^s_1, \mathsf{P}^s_2, \ldots, \mathsf{P}^s_{m_s}$, and $\mathsf{C}^s = \mathsf{C}^s_1, \mathsf{C}^s_2, \ldots, \mathsf{C}^s_{m_s}$. If this fixed transcript does not contain any pointless query, then the transcript is called *allowed*.

Now fix a transcript tr which maximizes the probability of bad being set. This transcript tr is hardwired into the game NON. The syntax of NON is the same as the syntax of RAND3, except that the part before the finalization step is not present in NON. The main difference between NON and RAND3 is in the interpretation of the variables. The tweaks, plaintext and ciphertext blocks in RAND3 are given by the adversary while in NON they are part of the transcript tr which is hardwired into the game. We denote this difference by using the symbols $\mathsf{T}^s, \mathsf{P}^s_i$ and C^s_i to denote the tweaks, plaintext and ciphertext blocks respectively in game NON. We have

$$\Pr[A^{\mathrm{RAND3}} \text{ sets } \mathsf{bad}] \leq \Pr[A^{\mathrm{NON}} \text{ sets } \mathsf{bad}]. \tag{15}$$

A.2 Analysis of Game NON

In the analysis we consider the sets $\mathcal{D}$ and $\mathcal{R}$ to consist of the formal variables instead of their values. For example, whenever we set $\mathcal{D} \leftarrow \mathcal{D} \cup \{X\}$ for some variable X we think of it as setting $\mathcal{D} \leftarrow \mathcal{D} \cup \{``X"\}$ where "X" is the name of that formal variable. Thus, our goal would be to bound the probability that two formal variables in sets $\mathcal{D}$ and $\mathcal{R}$ take the same value. The formal variables in both $\mathcal{D}$ and $\mathcal{R}$ do not depend on the random choices made in the game NON.

The formal variables which enter $\mathcal{D}$ and $\mathcal{R}$ depend on the length m_s of the sth query. We identify three cases.

$\underline{m_s = 1}$:

Domain elements : T^s, $R^s \oplus \mathsf{bin}(m_s)$, xEN^s and $\mathsf{P}^s_1 \oplus EN^s$;

Range elements : R^s, EN^s, EEN^s and $\mathsf{C}^s_1 \oplus xEEN^s$;

$\underline{m_s = 2}$:

Domain elements : T^s, $R^s \oplus \mathsf{bin}(m_s)$, xEN^s,
$\mathsf{P}_1^s \oplus R^s(x)\mathsf{P}_2^s(x) \oplus EN^s$, $\mathsf{C}_1^s \oplus L^s(x)\mathsf{C}_2^s(x) \oplus EEN^s$,
$\mathsf{P}_1^s \oplus M_1^s \oplus EN^s$ and $R^s(x)\mathsf{P}_2^s(x) \oplus M_1^s \oplus EEN^s$.

Range elements : R^s, EN^s, EEN^s, M_1^s, M_2^s,
$\mathsf{C}_1^s \oplus M_2^s \oplus EN^s$ and $L^s(x)\mathsf{C}_2^s(x) \oplus M_2^s \oplus EEN^s$.

$\underline{m_s \geq 3}$:

Domain elements : T^s, $R^s \oplus \mathsf{bin}(m_s)$, xEN^s,
$MPP^s = \bigoplus_{i=1}^{m_s}(R^s)^{i-1}\mathsf{P}_i^s \oplus EN^s$,
$MCC^s = \bigoplus_{i=1}^{m_s}(L^s)^{i-1}\mathsf{C}_i^s \oplus EEN^s$ and
$PP_i^s = (R^s)^{i-1}\mathsf{P}_i^s \oplus p_{m_s,i}(x)M_1^s$ for $1 \leq i \leq m_s$.

Range elements : R^s, EN^s, EEN^s, M_1^s, M_2^s, and
$CC_i^s = (L^s)^{i-1}\mathsf{C}_i^s \oplus p_{m_s,i}(x)M_2^s$ for $1 \leq i \leq m_s$.

Notes:

1. From the games, it is clear that the domain contains only distinct values of T^s. Also, if the tweak T^s has been used previously, then the previous value of R^s is used.

2. If $(T^s, m_s) = (T^r, m_r)$ for some $r < s$, then EN^r and EEN^r are used as values of EN^s and EEN^s respectively.

Let X_1^s, X_2^r be distinct elements of $\mathcal{D}$ and Y_1^s, Y_2^r be distinct elements of $\mathcal{R}$. We use the superscripts s and r to denote that the quantities are obtained from the sth and the rth adversarial queries. We would like to upper bound the probabilities of $X_1^s = X_2^r$ and $Y_1^s = Y_2^r$. We first consider X_1^s and X_2^r. There are two main cases to consider.

Intra query collision: This case arises, when $s = r$ and the quantities X_1^s and X_2^s are obtained from the same query. We consider $Z^s = X_1^s \oplus X_2^s$ where X_1^s and X_2^s vary over the quantities obtained from a single query. In each case, Z^s can be written as $Z^s = Z_1^s \oplus Z_2^s$, where Z_2^s is a random n-bit string and Z_2^s does not occur in the expression for Z_1^s. Consequently, we have

$$\Pr[X_1^s = X_2^s] = 1/2^n.$$

A detailed verification is tedious and has been done by us. We do not describe all the details but only some of the more non-trivial cases.

1. $m_s = 1$, $X_1^s = xEN_s$ and $X_2^s = \mathsf{P}_1 \oplus EN^s$: This gives $Z^s = \mathsf{P}_1 \oplus (x \oplus 1)EN^s$. Here $Z_1^s = \mathsf{P}_1$ and $Z_2^s = (x \oplus 1)EN^s$. Since EN^s is a random n-bit string (considered to be a polynomial of degree at most $n-1$) and $(x \oplus 1)$ is a non-zero polynomial, the product $(x \oplus 1)EN^s \bmod \tau(x)$ is also a random polynomial of degree at most $n-1$, i.e., a random n-bit string.

2. $m_s \geq 3$, $X_1^s = PP_i^s = (R^s)^{i-1}\mathsf{P}_i^s \oplus p_{m_s,i}(x)M_1^s$ and $X_1^s = PP_j^s = (R^s)^{j-1}\mathsf{P}_j^s \oplus p_{m_s,j}(x)M_1^s$. Now $Z^s = (R^s)^{i-1}\mathsf{P}_i^s \oplus (R^s)^{j-1}\mathsf{P}_j^s \oplus (p_{m_s,i}(x) \oplus p_{m_s,j}(x))M_1^s$ where $Z_2^s = (p_{m_s,i}(x) \oplus p_{m_s,j}(x))M_1^s$. Recall that by the second property of allowed sequences, we have $(p_{m_s,i}(x) \oplus p_{m_s,j}(x)) \not\equiv 0 \bmod \tau(x)$. Consequently, Z_2^s is a random n-bit string.

3. Reasonings similar to the above two points occur for other cases.

Inter query collision: For this case we have $s \neq r$, i.e., the elements X_1^s and X_2^r correspond to quantities obtained from separate queries. We divide this into two subcases.

$(T^s, m_s) \neq (T^r, m_r)$: Again a detailed and tedious case analysis shows that $\Pr[X_1^s = X_2^r] = 1/2^n$.

$(T^s, m_s) = (T^r, m_r)$: This implies that two same length queries have been submitted with the same tweak. In this case, we have $R^s = R^r$, $EN^s = EN^r$ and $EEN^s = EEN^r$. We divide this into three cases.

1. $(X_1^s, X_2^r) = (MPP^s, MPP^r)$: In this case, we necessarily have $m_s \geq 3$. We write

$$Z = X_1^s \oplus X_2^r = \bigoplus_{i=1}^{m_s} \Big((R^s(x))^{i-1} \mathsf{P}_i(x) \Big)$$

   where $\mathsf{P}_i(x) = (\mathsf{P}_i^s(x) \oplus \mathsf{P}_i^r(x))$. Consider $Z(X) = \bigoplus_{i=1}^{m_s} \alpha_i X^{i-1}$, with $\alpha_i = \mathsf{P}_i(x)$ to be a polynomial over $GF(2^n)$. The coefficients of this polynomial are determined by the transcript which supplies the quantities $\mathsf{P}_i^s(x)$ and $\mathsf{P}_i^r(x)$. However, since $(T^s, m_s) = (T^r, m_r)$ and the condition that the transcript does not contain pointless queries, we cannot have $\mathsf{P}_i^s(x) = \mathsf{P}_i^r(x)$ for all $i = 1, \ldots, m_s$. Thus, $Z(X)$ is a non-zero polynomial of degree at most $m_s - 1$ and hence has at most $m_s - 1$ distinct roots. We have $X_1^s = X_2^r$ if and only if $Z = Z(R) = 0$. Since R is randomly chosen, the probability that it equals one of the roots of $Z(X)$ is at most $(m_s - 1)/2^n$ and hence $\Pr[X_1^s = X_2^r] \leq (m_s - 1)/2^n$.

2. $(X_1^s, X_2^r) = (MCC^s, MCC^r)$: This is similar to the previous case and hence $\Pr[X_1^s = X_2^r] \leq (m_s - 1)/2^n$.

3. $(X_1^s, X_2^r) \neq (MPP^s, MPP^r)$ and $(X_1^s, X_2^r) \neq (MCC^s, MCC^r)$: In this case, again a detailed and tedious case analysis shows that $\Pr[X_1^s = X_2^r] = 1/2^n$.

We are now in a position to upper bound the probability $\Pr[X_1^s = X_2^r]$. Suppose the adversary makes a total of t_j queries of length l_j for $j = 1, \ldots, p$ for some p. Then $q = t_1 + \cdots + t_p$ and $\sigma_n = \sum_{s=1}^{q} m_s = \sum_{i=1}^{p} t_i l_i$. Note that $\sum_{s=1}^{q} m_s$ is the total number of n-bit ciphertext blocks provided by the adversary. Also, we have

$$3q + \sum_{s=1}^{q} m_s \leq |\mathcal{D}| \leq 5q + \sum_{s=1}^{q} m_s.$$

The probability of $X_1^s = X_2^r$ under the condition $(T^s, m_s) = (T^r, m_r)$ and $(X_1^s, X_2^r) = (MPP^s, MPP^r)$ is at most

$$A = \binom{t_1}{2} \frac{l_1 - 1}{2^n} + \cdots + \binom{t_p}{2} \frac{l_p - 1}{2^n}.$$

Then the probability of $X_1^s = X_2^r$ under the condition $(T^s, m_s) = (T^r, m_r)$ and $(X_1^s, X_2^r) = (MCC^s, MCC^r)$ is also at most A. There are $\binom{|\mathcal{D}|}{2}$ many two element subsets of $\mathcal{D}$. Out of these there are at most

$$B = 2 \times \left(\binom{t_1}{2} + \ldots + \binom{t_p}{2} \right)$$

pairs of the type $(X_1^s, X_2^r) = (MPP^s, MPP^r)$ or $(X_1^s, X_2^r) = (MCC^s, MCC^r)$. For each of the $\binom{|\mathcal{D}|}{2} - B$ two element subsets of $\mathcal{D}$, we have $\Pr[X_1^s = X_2^r] = 1/2^n$. Thus, the probability that two elements of $\mathcal{D}$ take the same value is at most

$$C = \left(\binom{|\mathcal{D}|}{2} - B\right) \times \frac{1}{2^n} + 2 \times \left(\binom{t_1}{2}\frac{l_1 - 1}{2^n} + \cdots + \binom{t_p}{2}\frac{l_p - 1}{2^n}\right).$$

Then we have,

$$2^n C \leq |\mathcal{D}|^2 + \sum_{i=1}^{p} t_i^2 l_i.$$

Using the loose approximation

$$\sum_{i=1}^{p} t_i^2 l_i \leq \sum_{i=1}^{p} t_i^2 l_i^2 \leq \left(\sum_{i=1}^{p} t_i l_i\right)^2 = \left(\sum_{s=1}^{q} m_s\right)^2 \leq |\mathcal{D}|^2$$

we have $2^n C \leq 2|\mathcal{D}|^2$. Thus, the probability of a domain collision is at most

$$2 \times \frac{(5q + \sigma_n)^2}{2^n}. \tag{16}$$

The probability that two elements Y_1^s and Y_1^r in $\mathcal{R}$ are equal can be analysed in a similar but simpler manner. The analysis is simpler, since in this case we do not have to consider the collision of the type $MPP^s = MPP^r$ and $MCC^s = MCC^r$. Hence, the probability of collision in $\mathcal{R}$ is

$$\frac{\binom{|\mathcal{R}|}{2}}{2^n} \leq \frac{(5q + \sigma_n)^2}{2^n}. \tag{17}$$

Thus, combining (12), (14), (15), (16) and (17) we have,

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{PEP}[\mathrm{Perm}(n)]}(A) \leq 3 \times \frac{(5q + \sigma_n)^2}{2^n}. \tag{18}$$

Figure 4: Game PEP1

Subroutine Ch-$\pi(X)$

$Y \xleftarrow{\$} \{0,1\}^n$; **if** $Y \in Range$ **then** bad = true; $\boxed{Y \xleftarrow{\$} \overline{Range}}$; **endif**;

if $X \in Domain$ **then** bad = true; $\boxed{Y = \pi(X)}$; **endif**

$\pi(X) = Y$; $Domain = Domain \cup \{X\}$; $Range = Range \cup \{Y\}$; **return**(Y);

Subroutine Ch-$\pi^{-1}(Y)$

$X \xleftarrow{\$} \{0,1\}^n$; **if** $X \in Domain$, bad = true; $\boxed{X \xleftarrow{\$} \overline{Domain}}$; **endif**;

if $Y \in Range$ **then** bad = true; $\boxed{X = \pi^{-1}(Y)}$; **endif**;

$\pi(X) = Y$; $Domain = Domain \cup \{X\}$; $Range = Range \cup \{Y\}$; **return**(X);

Initialization:

for all $X \in \{0,1\}^n$ $\pi(X) =$ undef **endfor**

bad = false

Respond to the s^{th} adversarial query as follows:

Encipher query: Enc$(T^s; P_1^s, \ldots, P_{m_s}^s)$

if $T^s == T^r$ for some $r < s$ **then**
$R^s = R^r$;
if $m_s == m_r$ **then**
$EN^s = EN^r$; $EEN^s = EEN^r$;
else
$EN^s =$ Ch-$\pi(R^s \oplus \mathsf{bin}(m_s))$; $EEN^s =$ Ch-$\pi(xEN^s)$;
endif
else
$R^s =$ Ch-$\pi(T^s)$;
$EN^s =$ Ch-$\pi(R^s \oplus \mathsf{bin}(m_s))$; $EEN^s =$ Ch-$\pi(xEN^s)$;
endif
if $m_s == 1$, **then**
$PPP_1^s = P_1^s \oplus EN^s$; $CCC_1^s =$ Ch-$\pi(PPP_1)$;
$C_1^s = CCC_1^s \oplus xEEN^s$; **return** C_1^s;
endif
if $m_s == 2$, **then**
$PP_1^s = P_1^s$; $PP_2^s = R^s(x)P_2^s(x)$;
$MPP^s = PP_1^s \oplus PP_2^s \oplus EN^s$; $M_1^s =$ Ch-$\pi(MPP^s)$;
$PPP_1^s = PP_1^s \oplus M_1^s \oplus EN^s$;
$PPP_2^s = PP_2^s \oplus M_1^s \oplus EEN^s$;
$CCC_1^s =$ Ch-$\pi(PPP_1^s)$; $CCC_2^s =$ Ch-(PPP_2^s);
$MCC^s = CCC_1^s \oplus CCC_2^s \oplus EN^s$; $M_2^s =$ Ch-$\pi(MCC^s)$;
$CC_1^s = CCC_1^s \oplus M_2^s \oplus EN^s$;
$CC_2^s = CCC_2^s \oplus M_2^s \oplus EEN^s$;
$C_1^s = CC_1^s$; $C_2^s = R^s(x)CC_2^s(x)$; **return** C_1^s, C_2^s;
endif
if $m_s \geq 3$, **then**
$R_1^s = 1$; $PP_1^s = P_1$; $MPP^s = PP_1^s$;
for $i = 2$ to m_s do
$R_i^s = R^s(x)R_{i-1}^s(x)$; $PP_i^s(x) = R_i^s(x)P_i^s(x)$;
$MPP^s = MPP^s \oplus PP_i^s$;
end for
$MPP^s = MPP^s \oplus EN^s$;
$M_1^s =$ Ch-$\pi(MPP^s)$; $MCC^s = 0^n$;
for $i = 1$ to m_s do
$PPP_i^s = PP_i^s \oplus p_{m_s,i}(x)M_1^s(x)$;
$CCC_i^s =$ Ch-$\pi(PPP_i^s)$; $MCC^s = MCC^s \oplus CCC_i^s$;
end for
$MCC^s = MCC^s \oplus EEN^s$; $M_2^s =$ Ch-$\pi(MCC^s)$;
$CC_1^s = CCC_1^s \oplus p_{m_s,1}(x)M_2^s(x)$; $R_1^s = 1$; $C_1^s = CC_1^s$;
for $i = 2$ to m_s do
$CC_i^s = CCC_i^s \oplus p_{m_s,i}(x)M_2^s(x)$;
$R_i^s(x) = R^s(x)R_{i-1}^s(x)$; $C_i^s = R_i^s(x)CC_i^s(x)$;
end for
return $C_1^s, C_2^s, \ldots, C_{m_s}^s$;
endif

Decipher query: Dec$(T^s; C_1^s, \ldots, C_{m_s}^s)$

if $T^s == T^r$ for some $r < s$ **then**
$R^s = R^r$; $L^s(x) = (R^s)^{-1}(x)$;
if $m_s == m_r$ **then**
$EN^s = EN^r$; $EEN^s = EEN^r$
else
$EN^s =$ Ch-$\pi(R^s \oplus \mathsf{bin}(m_s))$; $EEN^s =$ Ch-$\pi(xEN^s)$;
endif
else
$R^s =$ Ch-$\pi(T^s)$; $L^s(x) = (R^s)^{-1}(x)$;
$EN^s =$ Ch-$\pi(R^s \oplus \mathsf{bin}(m_s))$; $EEN^s =$ Ch-$\pi(xEN^s)$;
endif
if $m_s == 1$, **then**
$CCC_1^s = C_1^s \oplus xEEN^s$; $PPP_1^s =$ Ch-$\pi^{-1}(CCC_1^s)$;
$P_1^s = PPP_1^s \oplus EN^s$; **return** P_1^s;
endif
if $m_s == 2$, **then**
$CC_1^s = C_1^s$; $CC_2^s = L^s(x)C_2^s(x)$;
$MCC^s = CC_1^s \oplus CC_2^s \oplus EEN^s$; $M_2^s =$ Ch-$\pi(MCC^s)$;
$CCC_1^s = CC_1^s \oplus M_2^s \oplus EN^s$;
$CCC_2^s = CC_2^s \oplus M_2^s \oplus EEN^s$;
$PPP_1^s =$ Ch-$\pi^{-1}(CCC_1^s)$; $PPP_2^s =$ Ch-$\pi^{-1}(CCC_2^s)$;
$MPP^s = PPP_1^s \oplus PPP_2^s \oplus EEN^s$; $M_1^s =$ Ch-$\pi(MPP^s)$;
$PP_1^s = PPP_1^s \oplus M_1^s \oplus EN^s$;
$PP_2^s = PPP_2^s \oplus M_1^s \oplus EEN^s$;
$P_1^s = PP_1^s$; $P_2^s = L^s(x)PP_2^s(x)$; **return** P_1^s, P_2^s;
endif
if $m_s \geq 3$, **then**
$L_1^s = 1$; $CC_1^s = C_1^s$; $MCC^s = CC_1^s$;
for $i = 2$ to m_s do
$L_i^s = L^s(x)L_{i-1}^s(x)$; $CC_i^s = L_i^s(x)C_i^s(x)$;
$MCC^s = MCC^s \oplus CC_i^s$;
end for
$MCC^s = MCC^s \oplus EEN^s$;
$M_2^s =$ Ch-$\pi(MCC^s)$; $MPP^s = 0^n$;
for $i = 1$ to m_s do
$CCC_i^s = CC_i^s \oplus p_{m_s,i}(x)M_2^s(x)$;
$PPP_i^s =$ Ch-$\pi^{-1}(CCC_i^s)$; $MPP^s = MPP^s \oplus PPP_i^s$;
end for
$MPP^s = MPP^s \oplus EN^s$; $M_1^s =$ Ch-$\pi(MPP^s)$;
$PP_1^s = PPP_1^s \oplus p_{m_s,1}(x)M_1^s(x)$; $L_1^s = 1$; $P_1^s = PP_1^s$;
for $i = 2$ to m_s do
$PP_i^s = PPP_i^s \oplus p_{m_s,i}(x)M_1^s$;
$L_i^s(x) = L^s(x)L_{i-1}^s(x)$; $P_i^s = L_i^s(x)PP_i^s(x)$;
end for
return $P_1^s, P_2^s, \ldots, P_{m_s}^s$;
endif

Figure 5: Game RAND2

Subroutine Check-Domain-Range(X, Y)
if $X \in Domain$ **then** bad = true; **endif**
if $Y \in Range$ **then** bad = true; **endif**
$Domain = Domain \cup \{X\}$; $Range = Range \cup \{Y\}$;
INITIALIZATION
$Domain = Range = \emptyset$; bad = false;

ENCIPHER QUERY: Enc$(T^s; P_1^s, \ldots, P_{m_s}^s)$
if $T^s == T^r$ for some $r < s$ **then**
$R^s = R^r$; $L^s(x) = (R^s)^{-1}(x)$;
if $m_s == m_r$ **then**
$EN^s = EN^r$; $EEN^s = EEN^r$;
else
$EN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(R^s \oplus \mathsf{bin}(m_s), EN^s)$;
$EEN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(xEN^s, EEN^s)$;
endif
else
$R^s \xleftarrow{\$} \{0,1\}^n$; $L^s(x) = (R^s)^{-1}(x)$; Check-Domain-Range(T^s, R^s);
$EN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(R^s \oplus \mathsf{bin}(m_s), EN^s)$;
$EEN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(xEN^s, EEN^s)$;
endif
if $m_s == 1$, **then**
$PPP_1^s = P_1^s \oplus EN^s$; $C_1^s \xleftarrow{\$} \{0,1\}^n$; $CCC_1^s = C_1^s \oplus xEEN^s$;
Check-Domain-Range(PPP_1^s, CCC_1^s); **return** C_1^s;
endif
if $m_s == 2$, **then**
$PP_1^s = P_1^s$; $PP_2^s = R^s(x)P_2^s(x)$;
$MPP^s = PP_1^s \oplus PP_2^s \oplus EN^s$;
$M_1^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MPP^s, M_1^s)$;
$PPP_1^s = PP_1^s \oplus M_1^s \oplus EN^s$;
$PPP_2^s = PP_2^s \oplus M_1^s \oplus EEN^s$;
$C_1^s \xleftarrow{\$} \{0,1\}^n$; $C_2^s \xleftarrow{\$} \{0,1\}^n$;
$CC_1^s = C_1^s$; $CC_2^s = L^s(x)C_2^s(x)$;
$MCC^s = CC_1^s \oplus CC_2^s \oplus EEN^s$;
$M_2^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MCC^s, M_2^s)$;
$CCC_1^s = CC_1^s \oplus M_2^s \oplus EN^s$;
$CCC_2^s = CC_2^s \oplus M_2^s \oplus EEN^s$;
Check-Domain-Range(PPP_1^s, CCC_1^s);
Check-Domain-Range(PPP_2^s, CCC_2^s);
return C_1^s, C_2^s;
endif
if $m_s \geq 3$, **then**
$R_1^s = 1$; $PP_1^s = P_1$; $MPP^s = PP_1^s$;
for $i = 2$ to m_s do
$R_i^s = R^s(x)R_{i-1}^s(x)$; $PP_i^s(x) = R_i^s(x)P_i^s(x)$;
$MPP^s = MPP^s \oplus PP_i^s$;
end for
$MPP^s = MPP^s \oplus EN^s$;
$M_1^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MPP^s, M_1^s)$;
$C_1^s \xleftarrow{\$} \{0,1\}^n$; $CC_1^s = C_1^s$; $MCC^s = CC_1^s$; $L_1^s = 1$;
for $i = 2$ to m_s do
$C_i^s \xleftarrow{\$} \{0,1\}^n$; $L_i^s(x) = L^s(x)L_{i-1}^s(x)$;
$CC_i^s = L_i^s(x)C_i^s(x)$; $MCC^s = MCC^s \oplus CC_i^s$;
end for
$MCC^s = MCC^s \oplus EEN^s$;
$M_2^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MCC^s, M_2^s)$;
for $i = 1$ to m_s do
$PPP_i^s = PP_i^s \oplus p_{m_s,i}(x)M_1^s(x)$;
$CCC_i^s = CC_i^s \oplus p_{m_s,i}(x)M_2^s(x)$;
Check-Domain-Range(PPP_i, CCC_i);
end for
return $C_1^s, C_2^s, \ldots, C_{m_s}^s$;
endif

DECIPHER QUERY Dec$(T^s; C_1^s, \ldots, C_{m_s}^s)$
if $T^s == T^r$ for some $r < s$ **then**
$R^s = R^r$; $L^s(x) = (R^s)^{-1}(x)$;
if $m_s == m_r$ **then**
$EN^s = EN^r$; $EEN^s = EEN^r$;
else
$EN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(R^s \oplus \mathsf{bin}(m_s), EN^s)$;
$EEN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(xEN^s, EEN^s)$;
endif
else
$R^s \xleftarrow{\$} \{0,1\}^n$; $L^s(x) = (R^s)^{-1}(x)$; Check-Domain-Range(T^s, R^s);
$EN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(R^s \oplus \mathsf{bin}(m_s), EN^s)$;
$EEN^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(xEN^s, EEN^s)$;
endif
if $m_s == 1$, **then**
$CCC_1^s = C_1^s \oplus xEEN^s$; $P_1^s \xleftarrow{\$} \{0,1\}^n$; $PPP_1^s = P_1^s \oplus EN^s$;
Check-Domain-Range(PPP_1^s, CCC_1^s); **return** P_1^s;
endif
if $m_s == 2$, **then**
$CC_1^s = C_1^s$; $CC_2^s = L^s(x)C_2^s(x)$;
$MCC^s = CC_1^s \oplus CC_2^s \oplus EEN^s$;
$M_2^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MCC^s, M_2^s)$;
$CCC_1^s = CC_1^s \oplus M_2^s \oplus EN^s$;
$CCC_2^s = CC_2^s \oplus M_2^s \oplus EEN^s$;
$P_1^s \xleftarrow{\$} \{0,1\}^n$; $P_2^s \xleftarrow{\$} \{0,1\}^n$;
$PP_1^s = P_1^s$; $PP_2^s = R^s(x)P_2^s(x)$;
$MPP^s = PP_1^s \oplus PP_2^s \oplus EN^s$;
$M_1^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MPP^s, M_1^s)$;
$PPP_1^s = PP_1^s \oplus M_1^s \oplus EN^s$;
$PPP_2^s = PP_2^s \oplus M_1^s \oplus EEN^s$;
Check-Domain-Range(PPP_1^s, CCC_1^s);
Check-Domain-Range(PPP_2^s, CCC_2^s);
return P_1^s, P_2^s;
endif
if $m_s \geq 3$, **then**
$L_1^s = 1$; $CC_1^s = C_1^s$; $MCC^s = CC_1^s$;
for $i = 2$ to m_s do
$L_i^s = L^s(x)L_{i-1}^s(x)$; $CC_i^s = L_i^s(x)C_i^s(x)$;
$MCC^s = MCC^s \oplus CC_i^s$;
end for
$MCC^s = MCC^s \oplus EEN^s$;
$M_2^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MCC^s, M_2^s)$;
$P_1^s \xleftarrow{\$} \{0,1\}^n$; $PP_1^s = P_1^s$; $MPP^s = PP_1^s$; $R_1^s = 1$;
for $i = 2$ to m_s do
$P_i^s \xleftarrow{\$} \{0,1\}^n$; $L_i^s(x) = L^s(x)L_{i-1}^s(x)$;
$P_i^s = L_i^s(x)PP_i^s(x)$; $MPP^s = MPP^s \oplus PP_i^s$;
end for
$MPP^s = MPP^s \oplus EN^s$;
$M_1^s \xleftarrow{\$} \{0,1\}^n$; Check-Domain-Range$(MPP^s, M_1^s)$;
for $i = 1$ to m_s do
$CCC_i^s = CC_i^s \oplus p_{m_s,i}(x)M_2^s(x)$;
$PPP_i^s = PP_i^s \oplus p_{m_s,i}(x)M_1^s$;
Check-Domain-Range(PPP_i, CCC_i);
end for
return $P_1^s, P_2^s, \ldots, P_{m_s}^s$;
endif

Figure 6: Game RAND3

Respond to the s^{th} adversary query as follows:
Encipher query $\mathsf{Enc}(N^s; P_1^s, \ldots, P_{m_s}^s)$
  $ty^s = \mathsf{Enc}$; $C^s = C_1^s, C_2^s, \ldots, C_{m_s}^s \xleftarrow{\$} \{0,1\}^{nm_s}$; **return** C^s;
Decipher query $\mathsf{Dec}(N^s; C_1^s, \ldots, C_{m_s}^s)$
  $ty^s = \mathsf{Dec}$; $P^s = P_1^s, P_2^s, \ldots, P_{m_s}^s \xleftarrow{\$} \{0,1\}^{nm_s}$; **return** P^s;

Finalization:

First phase
$\mathcal{D} = \emptyset$; $\mathcal{R} = \emptyset$;
for $s = 1$ to q,
  if $T^s == T^r$ for some $r < s$ **then**
    $R^s = R^r$; $L^s(x) = (R^s)^{-1}(x)$;
    if $m_s == m_r$ **then**
      $EN^s = EN^r$; $EEN^s = EEN^r$;
    else
      $EN^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{R^s \oplus \mathsf{bin}(m_s)\}$; $\mathcal{R} = \mathcal{R} \cup \{EN^s\}$;
      $EEN^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{xEN^s\}$; $\mathcal{R} = \mathcal{R} \cup \{EEN^s\}$;
    endif
  else
    $R^s \xleftarrow{\$} \{0,1\}^n$; $L^s(x) = (R^s)^{-1}(x)$; $\mathcal{D} = \mathcal{D} \cup \{T^s\}$; $\mathcal{R} = \mathcal{R} \cup \{R^s\}$;
    $EN^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{R^s \oplus \mathsf{bin}(m_s)\}$; $\mathcal{R} = \mathcal{R} \cup \{EN^s\}$;
    $EEN^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{xEN^s\}$; $\mathcal{R} = \mathcal{R} \cup \{EEN^s\}$;
  endif

Case $ty^s = \mathsf{Enc}$:
if $m_s == 1$, **then**
  $PPP_1^s = P_1^s \oplus EN^s$; $C_1^s \xleftarrow{\$} \{0,1\}^n$; $CCC_1^s = C_1^s \oplus xEEN^s$;
  $\mathcal{D} = \mathcal{D} \cup \{PPP_1^s\}$; $\mathcal{R} = \mathcal{R} \cup \{CCC_1^s\}$;
endif
if $m_s == 2$, **then**
  $PP_1^s = P_1^s$; $PP_2^s = R^s(x)P_2^s(x)$;
  $MPP^s = PP_1^s \oplus PP_2^s \oplus EN^s$;
  $M_1^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MPP^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_1^s\}$;
  $PPP_1^s = PP_1^s \oplus M_1^s \oplus EN^s$;
  $PPP_2^s = PP_2^s \oplus M_1^s \oplus EEN^s$;
  $C_1^s \xleftarrow{\$} \{0,1\}^n$; $C_2^s \xleftarrow{\$} \{0,1\}^n$;
  $CC_1^s = C_1^s$; $CC_2^s = L^s(x)C_2^s(x)$;
  $MCC^s = CC_1^s \oplus CC_2^s \oplus EEN^s$;
  $M_2^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MCC^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_2^s\}$;
  $CCC_1^s = CC_1^s \oplus M_2^s \oplus EN^s$;
  $CCC_2^s = CC_2^s \oplus M_2^s \oplus EEN^s$;
  $\mathcal{D} = \mathcal{D} \cup \{PPP_1^s, PPP_2^s\}$; $\mathcal{R} = \mathcal{R} \cup \{CCC_1^s, CCC_2^s\}$;
endif
if $m_s \geq 3$, **then**
  $R_1^s = 1$; $PP_1^s = P_1$; $MPP^s = PP_1^s$;
  for $i = 2$ to m_s do
    $R_i^s = R^s(x)R_{i-1}^s(x)$; $PP_i^s(x) = R_i^s(x)P_i^s(x)$;
    $MPP^s = MPP^s \oplus PP_i^s$;
  end for
  $MPP^s = MPP^s \oplus EN^s$;
  $M_1^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MPP^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_1^s\}$;
  $C_1^s \xleftarrow{\$} \{0,1\}^n$; $CC_1^s = C_1^s$; $MCC^s = CC_1^s$; $L_1^s = 1$;
  for $i = 2$ to m_s do
    $C_i^s \xleftarrow{\$} \{0,1\}^n$; $L_i^s(x) = L^s(x)L_{i-1}^s(x)$;
    $CC_i^s = L_i^s(x)C_i^s(x)$; $MCC^s = MCC^s \oplus CC_i^s$;
  end for
  $MCC^s = MCC^s \oplus EEN^s$;
  $M_2^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MCC^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_2^s\}$;
  for $i = 1$ to m_s do
    $PPP_i^s = PP_i^s \oplus p_{m_s,i}(x)M_1^s(x)$;
    $CCC_i^s = CC_i^s \oplus p_{m_s,i}(x)M_2^s(x)$;
    $\mathcal{D} = \mathcal{D} \cup \{PPP_i^s\}$; $\mathcal{R} = \mathcal{R} \cup \{CCC_i^s\}$;
  end for
endif

Case $ty^s = \mathsf{Dec}$:
if $m_s == 1$, **then**
  $CCC_1^s = C_1^s \oplus xEEN^s$; $P_1^s \xleftarrow{\$} \{0,1\}^n$; $PPP_1^s = P_1^s \oplus EN^s$;
  $\mathcal{D} = \mathcal{D} \cup \{PPP_1^s\}$; $\mathcal{R} = \mathcal{R} \cup \{CCC_1^s\}$;
endif
if $m_s == 2$, **then**
  $CC_1^s = C_1^s$; $CC_2^s = L^s(x)C_2^s(x)$;
  $MCC^s = CC_1^s \oplus CC_2^s \oplus EEN^s$;
  $M_2^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MCC^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_2^s\}$;
  $CCC_1^s = CC_1^s \oplus M_2^s \oplus EN^s$;
  $CCC_2^s = CC_2^s \oplus M_2^s \oplus EEN^s$;
  $P_1^s \xleftarrow{\$} \{0,1\}^n$; $P_2^s \xleftarrow{\$} \{0,1\}^n$;
  $PP_1^s = P_1^s$; $PP_2^s = R^s(x)P_2^s(x)$;
  $MPP^s = PP_1^s \oplus PP_2^s \oplus EN^s$;
  $M_1^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MPP^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_1^s\}$;
  $PPP_1^s = PP_1^s \oplus M_1^s \oplus EN^s$;
  $PPP_2^s = PP_2^s \oplus M_1^s \oplus EEN^s$;
  $\mathcal{D} = \mathcal{D} \cup \{PPP_1^s, PPP_2^s\}$; $\mathcal{R} = \mathcal{R} \cup \{CCC_1^s, CCC_2^s\}$;
endif
if $m_s \geq 3$, **then**
  $L_1^s = 1$; $CC_1^s = C_1^s$; $MCC^s = CC_1^s$;
  for $i = 2$ to m_s do
    $L_i^s = L^s(x)L_{i-1}^s(x)$; $CC_i^s = L_i^s(x)C_i^s(x)$;
    $MCC^s = MCC^s \oplus CC_i^s$;
  end for
  $MCC^s = MCC^s \oplus EEN^s$;
  $M_2^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MCC^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_2^s\}$;
  $P_1^s \xleftarrow{\$} \{0,1\}^n$; $PP_1^s = P_1^s$; $MPP^s = PP_1^s$; $R_1^s = 1$;
  for $i = 2$ to m_s do
    $P_i^s \xleftarrow{\$} \{0,1\}^n$; $L_i^s(x) = L^s(x)L_{i-1}^s(x)$;
    $P_i^s = L_i^s(x)PP_i^s(x)$; $MPP^s = MPP^s \oplus PP_i^s$;
  end for
  $MPP^s = MPP^s \oplus EN^s$;
  $M_1^s \xleftarrow{\$} \{0,1\}^n$; $\mathcal{D} = \mathcal{D} \cup \{MPP^s\}$; $\mathcal{R} = \mathcal{R} \cup \{M_1^s\}$;
  for $i = 1$ to m_s do
    $CCC_i^s = CC_i^s \oplus p_{m_s,i}(x)M_2^s(x)$;
    $PPP_i^s = PP_i^s \oplus p_{m_s,i}(x)M_1^s$;
    $\mathcal{D} = \mathcal{D} \cup \{PPP_i^s\}$; $\mathcal{R} = \mathcal{R} \cup \{CCC_i^s\}$;
  end for
endif

Second phase
  if (some value occurs more than once in $\mathcal{D}$) **then** $\mathsf{bad} = \mathsf{true}$ **endif**;
  if (some value occurs more than once in $\mathcal{R}$) **then** $\mathsf{bad} = \mathsf{true}$ **endif**.