BPS: a Format-Preserving Encryption Proposal

Eric Brier, Thomas Peyrin and Jacques Stern

Ingenico, France
{forename.name}@ingenico.com

Abstract. In recent months, attacks on servers of payment processors have led to the disclosure of tens of millions of credit card numbers (also known as Personal Account Numbers, PANs). As an answer, end-to-end encryption has been advocated and an encryption standard that preserves the format of the data would be welcome. More generally, a format-preserving encryption scheme would be welcomed for many real-life applications. Unfortunately, this request falls in an area that is not yet adequately covered by cryptography theory: direct constructions [1,20] have not received enough attention to be considered for standardization, and constructions based on Feistel schemes (as proposed by [5,4]) suffer from the lack of tight exact security estimates. Very recently, the use of unbalanced Feistel schemes has been suggested and a precise security bound, based on Markov chains, has been derived [12]. However, the bound comes at the cost of a large number of calls to the underlying cipher.
In this paper, we present a generic format-preserving symmetric encryption algorithm `BPS`, which can cipher short or long string of characters from any given set. In particular, this construction offers a tweak capability, very useful in practice when the user would like to cipher very small strings of data. We also provide particular instances for the case of PANs ciphering. Very recently, a similar proposal has been independently submitted to the NIST standardization process [3].

Most block ciphers from the industry and the academic world handle binary domain $\{0,1\}^n$, with a block size often equal to $n = 64$ bits or 128 bits. While those ciphers are clearly dealing with the most useful cases in practice, what if one wants to design a cipher that maintains another message domain $\mathcal{M}$ whose cardinality $|\mathcal{M}|$ is arbitrary? For example, such a primitive could be really useful in applications where the data manipulated is composed of digits and not bits, as it is the case for credit-card numbers (PANs). Of course, it is always possible to use a standardized block cipher with a larger binary domain $\mathcal{M}' = \{0,1\}^n$ ($|\mathcal{M}| \leq |\mathcal{M}'|$) and to use extra data fields coming with the ciphertext to restore an equivalent format. However, we are looking here for elegant constructions that are not based on any engineering trick and that produce ciphertexts with strictly no expansion. In practice, the expansion is equivalent as breaking the format, which many actors of the communication channel may not support.

Several dedicated block ciphers have recently been proposed to answer this challenge for particular situations [20,6,1,9]. Yet, in practice it would be interesting to have a construction that uses already standardized block ciphers or hash functions such as `TDES` [14], `AES` [15] or `SHA-2` [13] as internal primitive. In particular, those primitives are the most likely to be available on hardware. Black and Rogaway [5] provided a theoretic study of this problem. In their article, three potential constructions of arbitrary finite domains cipher have been proposed. The first method, named *prefix cipher*, uses as internal primitive a cipher E'_K with a larger domain than $|\mathcal{M}|$ and defines the permutation $E_K(i)$ by first computing all the $|\mathcal{M}|$ ciphertexts $j = E'_K(i)$ of messages i with $0 \leq i \leq |\mathcal{M}| - 1$ and by sorting them according to their value. The ordinal position in the sorted table of values j

corresponding to the query i gives $E_K(i)$. The second method, named *cycle-walking cipher*, also uses a cipher E'_K with a larger domain than $|\mathcal{M}|$. For a plaintext i, one outputs the value $j = E'_K(i)$ if $j \in \mathcal{M}$. The out-of-range ciphertexts are simply treated by applying the permutation E'_K again until one reaches the domain $\mathcal{M}$. Finally, the last method, named *generalized-Feistel cipher*, uses a Feistel construction [7] with some random functions F_i and modular additions. This construction maintains two branches with domains L and R such that $|\mathcal{M}| \leq L \times R$. When $|\mathcal{M}| < L \times R$, out-of-range ciphertexts may be reached and the construction is then combined with the cycle-walking cipher (i.e. the permutation is applied again until a valid ciphertext is reached).

The first method is interesting for small values of $|\mathcal{M}|$ but is completely unpractical otherwise since $2^{|\mathcal{M}|}$ time and memory are required in order to start using the cipher. The second method is practical but presents a drawback: the duration of a ciphering process is not deterministic. This could be a problem in some applications, even if the potential threat of timing attacks should not be harmful (see [4]). Finally, the last method seems to be the most elegant and promising one, even if the best known security proof yet only achieves a *birthday paradox* bound (for the binary case, better proofs are known [17,18]). More precisely, the analysis is an adaptation of the well known Luby-Rackoff security proof [11] and it shows that when the attacker is limited to access less than $Q = 2^{\min\{L,R\}/2}$ plaintext/ciphertext pairs, she has not enough information to distinguish this construction from a random permutation with domain $\mathcal{M}$.

This proof holds whatever the computing power of the attacker is. However, for intermediate values of $|\mathcal{M}|$, one can assume that the attacker can indeed access to Q queries in practice. For example, let's consider the case of the encryption of credit-card numbers between two parties. Only about a dozen digits are unpredictable in a credit-card number, thus we consider $\mathcal{M} = \{0, \ldots, 9\}^{12}$ and $|\mathcal{M}| = 10^{12}$. In this case, the generalized-Feistel cipher birthday proof [5] ensures security up to 1000 plaintext/ciphertext pairs. Note that a proposal by Spies [21] combining balanced Feistel networks and the cycle-walking technique has been submitted to the NIST in 2008.

A first improvement would be to design a tweakable block cipher [10,8] instead, as recently published by Bellare *et al.* [4]. In this case, the designer is ensured that much more plaintext/ciphertext pairs are necessary in order to attack the scheme (since these pairs are likely to use different tweak values). In our previous example, the attacker would have to get 1000 plaintext/ciphertext pairs with the same tweak value instead of 1000 random plaintext/ciphertext pairs.

In parallel, another route has recently been taken by Morris, Rogaway and Stegers [12], who used highly unbalanced Feistel schemes. Using the theory of Markov chains, the authors were able to derive exact security bounds. Despite their attractive features, these bounds come at the cost of a large number of calls to the underlying cipher due to many Feistel rounds, which might make them unsuitable in practice.

Very recently, a proposal combining the tweak feature [4] and the new security proofs techniques [12] has been submitted as a NIST proposal [3].

A second improvement would be to increase the expected security of the more conservative approach based on balanced Feistel schemes by improving the proven security bounds. In our case, that would mean going beyond the birthday bound. To achieve this, one would naturally draw his inspiration from Patarin's recent work on Feistel networks security [16,17,18,19], also crossing the birthday bound barrier. However, since the domain size can be small, we are aiming here at concrete security instead of asymptotic security, and this task seems quite difficult for the time being. Note however, that, contrary to unbalanced Feistel networks, the best bound one can achieve is $\mathcal{O}(2^n)$, since it is always

possible for a computationally unbounded adversary to distinguish a r-round Feistel cipher manipulating n-bit blocks from a random permutation with $r \times 2^n$ queries (by simply trying to guess all the r unknown internal functions used). This does not disqualify balanced Feistel networks since they seem to require much less calls to the underlying cipher.

Our contribution. We propose a simple yet very flexible format-preserving encryption algorithm. Our proposal can cipher short or long strings composed of characters from any set. `BPS` can use any standardized primitives such as `TDES` [14], `AES` [15] or `SHA-2` [13] as internal brick.

1 The Generic `BPS` Cipher

For the description, we will use the following notations and the little-endian order. Assume that one wants to cipher strings of characters from a set S, with s representing the cardinality of that set: $s = |S|$. For example, we have $S = \{0, 1\}$ and $s = 2$ in the case of bits, or $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $s = 10$ in the case of digits. The only parameter that matters here is the cardinality of the set of characters S, since one can always define a bijective mapping from S to $\{0, 1, \ldots, s-1\}$. Thus, in the following, we will only deal with integers in $\{0, 1, \ldots, s-1\}$ each representing a character in S, which we call a s-integer.

We denote by len the length of the string ST to cipher, i.e. $len = |ST|$, and we also denotes by $ST[i]$ the i-th s-integer of the string ST starting the counting from the left, with $0 \leq i < len$. For example, with $s = 16$, if $ST = 0\ 14\ 2\ 11$ we have $len = 4$ and $ST[0] = 0$, $ST[1] = 14$, $ST[2] = 2$, $ST[3] = 11$. Then, $ST||ST'$ denotes the concatenation of the strings ST and ST', i.e. if $ST' = 8\ 15\ 6\ 4$ then $ST||ST' = 0\ 14\ 2\ 11\ 8\ 15\ 6\ 4$.

`BPS` makes an extensive use of the modular addition (resp. subtraction) that we denote by $\boxplus$ (resp. $\boxminus$). When we will write $C = A \boxplus B \pmod{x}$, we consider that $A, B, C \in \mathbb{N}$. Of course, we have $C \in \{0, \ldots, x-1\}$. Moreover, we will use the bitwise exclusive or (XOR) operation that we denote $c = a \oplus b$ and where a, b, c are bit-words of the same length.

`BPS` is built upon two components: an internal length-limited block cipher (which itself uses an internal function such as `TDES` [14], `AES` [15] or `SHA-2` [13]) and a mode of operation in order to handle long strings. The two next sections respectively describe the two components.

1.1 The Internal Block Cipher `BC`

We denote by `BC` the internal cipher of `BPS`, distinguishing the encryption and the decryption processes by `BC` and $\texttt{BC}^{-1}$ respectively. We instantiate the cipher according to the cardinality s of the characters set and according to the block length b of the cipher we are building. Thus, $Y = \texttt{BC}_{F,s,b,w}(X, K, T)$ denote the w-round encryption (an even number) of a s-integer string X of length b, with key K and the 64-bit tweak value T. Of course, since we are building format-preserving encryption, the output string Y will also be composed of b s-integers.

We denote by f the number of output bits of the internal function F. We have the natural restriction that at least two characters must be ciphered, i.e. $b \geq 2$. Also, the bit length k of the key K is limited according to the internal function F used. If F is a f-bit block cipher that manipulates k'-bit keys, then we require that $k = k'$. In the case of F being a `HMAC` construction [2] with a f-bit hash function, one can use a key of arbitrary length. We denote by $F_K(x)$ the application of the block cipher E with the key K on the plaintext x ($F_K(x) = E_K(x)$), or the application of the `HMAC` construction with the hash function H and the key K on the message x ($F_K(x) = \texttt{HMAC}[H]_K(x)$).

We first divide the 64-bit tweak T into two 32-bit sub-tweaks T_L and T_R, i.e. if T is considered as a 64-bit integer, then $T_R = T \bmod 2^{32}$ and $T_L = (T - T_R)/2^{32}$. Then, we divide the s-integer input string X of length b into two sub-strings X_L and X_R of similar length l and r respectively, i.e. $X = X_L||X_R$. More precisely, if b is even, $X_L = X[0] \dots X[l-1]$ and $X_R = X[l] \dots X[l+r-1]$, where $l = r = b/2$. If b is odd, $X_L = X[0] \dots X[l-1]$ and $X_R = X[l] \dots X[l+r-1]$, where $l = (b+1)/2$ and $r = (b-1)/2$.

The internal state of the cipher is composed of two branches L and R, each of $f - 32$ bits.[1] We impose the last restriction (the explanation is given in later sections): $b \leq max_b$, with

$$max_b = 2 \times \lfloor \log_s(2^{f-32}) \rfloor.$$

For example, when using `AES` as internal function, each branch will be represented by a 96-bit integer and when ciphering digits ($s = 10$) we would have the restriction $b \leq 56$. We give in Table 1 the maximal value max_b for b according to s and the internal function F used.

Table 1. Maximal value max_b for the number b of input s-integers of `BC` and `BC`$^{-1}$, according to the characters set cardinality s and the internal function used.

	$s = 2$ bits	$s = 10$ digits	$s = 61$
`TDES`	64	18	10
`AES`	192	56	32
`SHA-2`	448	134	74

The encryption `BC` is composed of w simple Feistel-like rounds, and each of them will update the right or left branch in turn. We denote by L_i (resp. R_i) the left (resp. right) branch value after application of round i. The left and right branches are initialized with X_L and X_R respectively:

$$\begin{cases} L_0 = X_L[0].s^0 + X_L[1].s^1 + \dots + X_L[l-1].s^{l-1} \\ R_0 = X_R[0].s^0 + X_R[1].s^1 + \dots + X_R[r-1].s^{r-1} \end{cases}$$

When the encryption process `BC` is instantiated with a block cipher E, for each $0 \leq i < w$ we apply the round function (see Figure 1):

$$\begin{cases} L_{i+1} = L_i \boxplus E_K((T_R \oplus i).2^{f-32} + R_i) \pmod{s^l} & \text{if i is even} \\ L_{i+1} = L_i & \text{if i is odd} \end{cases}$$

$$\begin{cases} R_{i+1} = R_i & \text{if i is even} \\ R_{i+1} = R_i \boxplus E_K((T_L \oplus i).2^{f-32} + L_i) \pmod{s^r} & \text{if i is odd} \end{cases}$$

Finally, the output string Y is the concatenation of Y_L and Y_R, i.e. $Y = Y_L||Y_R$ with Y_L and Y_R built by decomposing L_w and R_w into the s basis:

[1] Conceptually, the two branches always maintain the formating and thus the left branch manipulates data in $\{0, \dots, s^l - 1\}$ and the right branch manipulates data in $\{0, \dots, s^r - 1\}$. However, those branches are always coded on $(f-32)$-bit integers for consistency with the concatenation function.

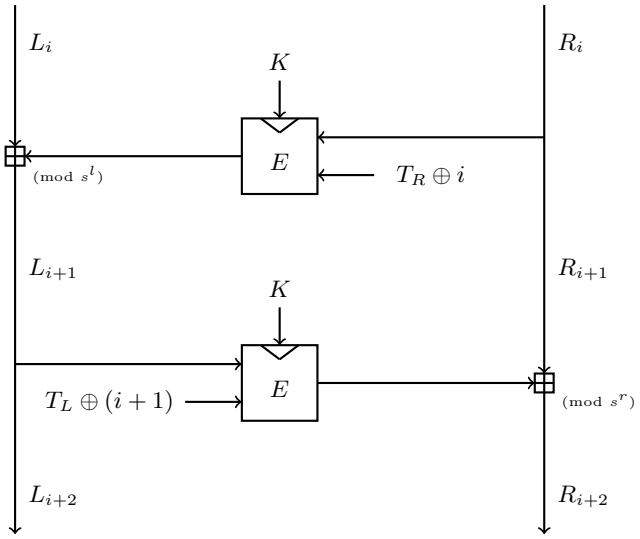

Fig. 1. 2 rounds of the encryption BC of the internal block cipher.

$$\begin{cases} Y_L[0].s^0 + Y_L[1].s^1 + \ldots + Y_L[l-1].s^{l-1} = L_w \\ Y_R[0].s^0 + Y_R[1].s^1 + \ldots + Y_R[r-1].s^{r-1} = R_w \end{cases}$$

Note that because of our restriction on the number of input s-integers, we always ensure that each branch can be coded on a $(f-32)$-bit word. The overall encryption process BC is given in Algorithm 1.

Algorithm 1 : encryption $\mathtt{BC}_{F,s,b,w}(X,K,T)$

$T_R = T \bmod 2^{32}$;
$T_L = (T - T_R)/2^{32}$;
$l = \lceil b/2 \rceil$, $r = \lfloor b/2 \rfloor$;
$L_0 = X[0].s^0 + X[1].s^1 + \ldots + X[l-1].s^{l-1}$;
$R_0 = X[l].s^0 + X[l+1].s^1 + \ldots + X[l+r-1].s^{r-1}$;
for $i = 0$ **to** $w-1$ **do**
  if (i is even) **then**
    $L_{i+1} = L_i \boxplus F_K((T_R \oplus i).2^{f-32} + R_i) \bmod s^l$;
    $R_{i+1} = R_i$;
  else
    $R_{i+1} = R_i \boxplus F_K((T_L \oplus i).2^{f-32} + L_i) \bmod s^r$;
    $L_{i+1} = L_i$;
end for
for $i = 0$ **to** $l-1$ **do**
  $Y[i] = L_w \bmod s$, $L_w = (L_w - Y[i])/s$;
end for
for $i = 0$ **to** $r-1$ **do**
  $Y[i+l] = R_w \bmod s$, $R_w = (R_w - Y[i+l])/s$;
end for
return Y;

The decryption $\mathtt{BC}^{-1}$ is composed of w simple Feistel-like rounds, and each of them will update the right or left branch in turn. We denote by L_i (resp. R_i) the left (resp. right) branch value after application of round i. The left and right branches are initialized with X_L and X_R respectively:

$$\begin{cases} L_w = X_L[0].s^0 + X_L[1].s^1 + \ldots + X_L[l-1].s^{l-1} \\ R_w = X_R[0].s^0 + X_R[1].s^1 + \ldots + X_R[r-1].s^{r-1} \end{cases}$$

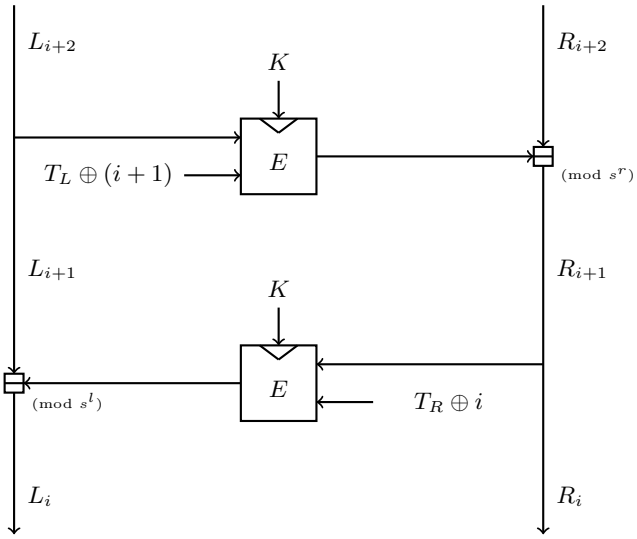

Fig. 2. 2 rounds of the decryption $\mathtt{BC}^{-1}$ of the internal block cipher.

When the decryption process $\mathtt{BC}^{-1}$ is instantiated with a block cipher E, for each $w > i \geq 0$ we apply the round function (see Figure 2):

$$\begin{cases} L_i = L_{i+1} \boxminus E_K((T_R \oplus i).2^{f-32} + R_{i+1}) \pmod{s^l} & \text{if i is even} \\ L_i = L_{i+1} & \text{if i is odd} \end{cases}$$

$$\begin{cases} R_i = R_{i+1} & \text{if i is even} \\ R_i = R_{i+1} \boxminus E_K((T_L \oplus i).2^{f-32} + L_{i+1}) \pmod{s^r} & \text{if i is odd} \end{cases}$$

Finally, the output string Y is the concatenation of Y_L and Y_R, i.e. $Y = Y_L || Y_R$ with Y_L and Y_R built by decomposing L_0 and R_0 into the s basis:

$$\begin{cases} Y_L[0].s^0 + Y_L[1].s^1 + \ldots + Y_L[l-1].s^{l-1} = L_0 \\ Y_R[0].s^0 + Y_R[1].s^1 + \ldots + Y_R[r-1].s^{r-1} = R_0 \end{cases}$$

Note that because of our restriction on the number of input s-integers, we always ensure that each branch can be coded on a $(f-32)$-bit word. The overall decryption process $\mathtt{BC}^{-1}$ is given in Algorithm 2.

Algorithm 2 : decryption $\mathtt{BC}^{-1}_{F,s,b,w}(X, K, T)$

$T_R = T \bmod 2^{32}$;
$T_L = (T - T_R)/2^{32}$;
$l = \lceil b/2 \rceil$, $r = \lfloor b/2 \rfloor$;
$L_w = X[0].s^0 + X[1].s^1 + \ldots + X[l-1].s^{l-1}$;
$R_w = X[l].s^0 + X[l+1].s^1 + \ldots + X[l+r-1].s^{r-1}$;
for $i = w - 1$ **to** 0 **do**
  if (i is even) **then**
    $L_i = L_{i+1} \boxminus F_K((T_R \oplus i).2^{f-32} + R_{i+1}) \bmod s^l$;
    $R_i = R_{i+1}$;
  else
    $R_i = R_{i+1} \boxminus F_K((T_L \oplus i).2^{f-32} + L_{i+1}) \bmod s^r$;
    $L_i = L_{i+1}$;
end for
for $i = 0$ **to** $l - 1$ **do**
  $Y[i] = L_w \bmod s$, $L_w = (L_w - Y[i])/s$;
end for
for $i = 0$ **to** $r - 1$ **do**
  $Y[i + l] = R_w \bmod s$, $R_w = (R_w - Y[i + l])/s$;
end for
return Y;

1.2 The Operating Mode

We described in previous section the internal encryption BC and decryption $\mathtt{BC}^{-1}$ routines that can handle plaintext of a limited length. Namely, one can cipher from 2 to $max_b = 2 \times \lfloor \log_s(2^{f-32}) \rfloor$ s-integers with one call. If one needs to cipher larger input strings (up to $max_b.2^{16}$ characters), we define an operating mode using BC and $\mathtt{BC}^{-1}$. This very simple process is similar to the classical Cipher-Block Chaining mode (CBC mode) for block ciphers, but instead of the XOR operation each character is added separately modulo s (for example, $275849 + 150965 = 325704$ with digits). When the total length is not a multiple of max_b, a shift is applied for the last call of the internal cipher in order to accommodate the current position of the cursor, i.e. the first input s-integers of the last cipher call are the last output s-integers of the previous cipher call. The whole process is given in Algorithm 3 for the encryption and in Algorithm 4 for the decryption. Of course, the operating mode can naturally support the use of initialization vectors (IV): the plaintext string of the first block processing is added character per character modulo s to an IV string of the same length.

2 Overview of BPS

The operating mode of BPS is simple and efficient. It is very similar to the well known Cipher-Block Chaining mode (CBC mode) for block cipher encryption with an IV set to 0. Moreover, we also incorporate a counter on the tweak input. More precisely, a 16-bit counter will be XORed on the 16 most significant bits of both the right and left 32-bit tweak words T_L and T_R (since those two words act separately in the BPS internal block cipher). Only the 16 most significant bits are impacted in order to avoid any conjunction with the local round counter applied on the least significant bits of the tweak in BC or $\mathtt{BC}^{-1}$. Therefore, we limit the size of the input string to 2^{16} blocks (which should be sufficient for all applications of format-preserving encryption). Finally, the local round counter ensures that we never use twice the same internal function F during a whole encryption process.

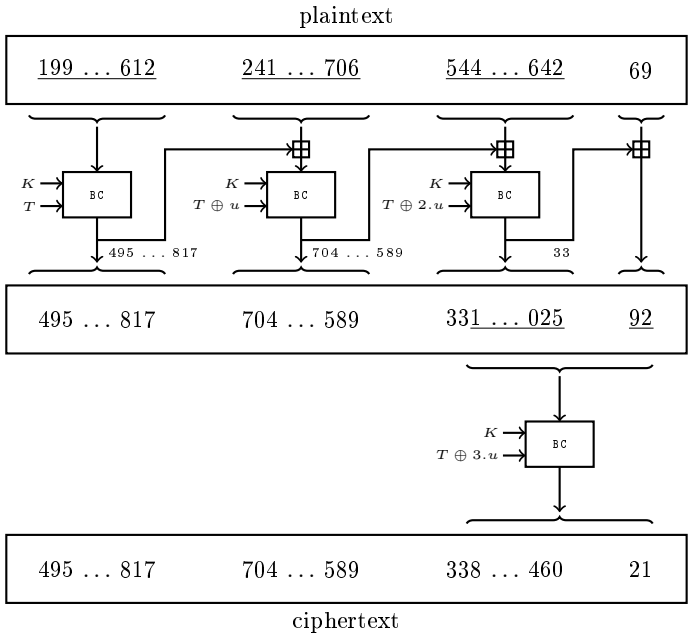

Fig. 3. Example of operating mode encryption process with $s = 10$ (digits) and $len = 3.max_b + 2$. We denote $u = 2^{16} + 2^{48}$.

Algorithm 3 : encryption $\texttt{BPS}_{F,s,len,w}(X, K, T)$

$max_b = 2 \times \lfloor \log_s(2^{f-32}) \rfloor$;
if $(len \leq max_b)$ **then**
  $Y[0, \ldots, len-1] = \texttt{BC}_{F,s,len,w}(X[0, \ldots, len-1], K, T)$;
  return Y;
$rest = len \bmod (max_b)$, $c = 0$, $i = 0$;
$Y[0, \ldots, len-1] = X[0, \ldots, len-1]$;
while $(len - c > max_b)$ **do**
  if $(i \neq 0)$ **then**
    $Y[c, \ldots, c + max_b - 1] =$
    $(Y[c - max_b] + Y[c] \bmod s), \ldots, (Y[c-1] + Y[c + max_b - 1] \bmod s)$;
  $Y[c, \ldots, c + max_b - 1] =$
  $\texttt{BC}_{F,s,max_b,w}(Y[c, \ldots, c + max_b - 1], K, T \oplus (i.2^{16}) \oplus (i.2^{48}))$;
  $c = c + max_b$, $i = i + 1$;
end while
if $(len \neq c)$ **then**
  $Y[len - rest, \ldots, len - 1] =$
  $(Y[len - rest - max_b] + Y[len - rest] \bmod s), \ldots, (Y[len - max_b - 1] + Y[len - 1] \bmod s)$;
  $Y[len - max_b, \ldots, len - 1] =$
  $\texttt{BC}_{F,s,len,w}(Y[len - max_b, \ldots, len - 1], K, T \oplus (i.2^{16}) \oplus (i.2^{48}))$;
return Y;

The core of BPS is built upon a Feistel network. This choice seems natural in regards to the history of block ciphers and format-preserving algorithms. Indeed, Feistel networks have been studied for a long time by the cryptography community and is considered as

Algorithm 4 : decryption $\texttt{BPS}^{-1}_{F,s,len,w}(X,K,T)$

$max_b = 2 \times \lfloor \log_s(2^{f-32}) \rfloor$;
if $(len \leq max_b)$ **then**
  $Y[0,\ldots,len-1] = \texttt{BC}^{-1}_{F,s,len,w}(X[0,\ldots,len-1],K,T)$;
  return Y;
$rest = len \bmod (max_b)$, $c = len - rest$, $i = \lfloor c/max_b \rfloor$;
$Y[0,\ldots,len-1] = X[0,\ldots,len-1]$;
if $(len \neq c)$ **then**
  $Y[len-max_b,\ldots,len-1] =$
  $\texttt{BC}^{-1}_{F,s,len,w}(Y[len-max_b,\ldots,len-1],K,T \oplus (i.2^{16}) \oplus (i.2^{48}))$;
  $Y[len-rest,\ldots,len-1] =$
  $(Y[len-rest] - Y[len-max_b-rest] \bmod s),\ldots,(Y[len-1] - Y[len-max_b-1] \bmod s)$;
while $(c \neq 0)$ **do**
  $c = c - max_b$, $i = i - 1$;
  $Y[c,\ldots,c+max_b-1] =$
  $\texttt{BC}^{-1}_{F,s,max_b,w}(X[c,\ldots,c+max_b-1],K,T \oplus (i.2^{16}) \oplus (i.2^{48}))$;
  if $(i \neq 0)$ **then**
    $Y[c,\ldots,c+max_b-1] =$
    $(Y[c] - Y[c-max_b] \bmod s),\ldots,(Y[c+max_b-1] - Y[c-1] \bmod s)$;
end while
return Y;

a robust method for building block ciphers. In particular, one can leverage the advances concerning security proofs [17,18,5] and generic attacks [16]. This lowers the probability of unexpected successful cryptanalysis compared to ad-hoc proposals [1,20]. This kind of construction seems also to be the best and simplest for solving the problem of format-preserving encryption, as early noticed by Black and Rogaway [5]. We believe that their original proposal is very elegant and is a major step for format-preserving constructions, but we made some adaptation in order to smoothly support any string length and add tweak capability.

We stated in previous sections that the number of s-integers that the internal block cipher BC and $\texttt{BC}^{-1}$ can handle is upper bounded by

$$max_b = 2 \times \lfloor \log_s(2^{f-32}) \rfloor.$$

This bound is due to the effects of the modular addition. Indeed, as already analyzed in [4], the statistical distance between the uniform distribution on $\mathbb{Z}_M$ and the distribution obtained by picking a random point x in $\mathbb{Z}_N$ and returning $x \bmod M$ is lower bounded by M/N. Thus, for each use of the modular addition in BC or $\texttt{BC}^{-1}$, our bound forces this statistical distance to be at most 2^{-32} and we believe this is sufficient to simulate a uniform output.

One of the main goals of BPS is its adaptability. One can use BPS to cipher strings from 2 to $max_b.2^{16}$ s-integers, picked from any set. Moreover, all the block cipher and hash function standardized primitives (TDES [14], AES [15] or SHA-2 [13]) can be used as basic internal bricks. This is another layer of confidence concerning the assumptions made on the security of the internal function F since one can rely on the advances of the academic community regarding cryptanalysis. In practice, the 64-bit block cipher TDES remains implemented in many applications, it is also very valuable to have a format-preserving encryption algorithm that can handle 64-bit internal primitives.

Another essential quality of BPS is its efficiency. First, one can note that the input key for all the block cipher internal calls is constant. This requires only one internal cipher keying per BPS encryption. Considering the non-negligible time a key schedule can take, this often saves a lot of operations. Moreover, we recommend to use $w = 8$ rounds for the Feistel network, which makes the whole encryption process very efficient.

Regarding the security, one can leverage on the existing proofs on Feistel networks [5]. However, while asymptotic proofs of security exist and could be adapted [17,18], finding tight concrete bounds is still an open problem. Solving it would be very welcome since we are potentially manipulating very small plaintext.

We chose to use $w = 8$ Feistel rounds in BC and $\mathtt{BC}^{-1}$ after analyzing the proven security bounds and the best known generic attacks. Patarin [16] already published a generic attack that can distinguish a $w \geq 6$ round Feistel based family of permutations from a random family of n-bit permutations with about $2^{\frac{n}{2}(w-4)}$ operations and $2^{\frac{n}{2}(w-6)}$ permutations. However, about 2^n plaintext/ciphertext pairs per tweak are required to mount the attack and this already exceeds our aimed goal of a s^b security. We also believe that such a threat is not relevant in practice since the technique only allows to distinguish several instances from our block cipher from a random keyed permutation family. Being resistant to this technique would certainly lead to an overkill in terms of performance (for example, the recent NIST submission FFX [3] do resist to this attack, but requires much more internal function calls than BPS).

Using the tweak is very valuable to avoid some kind of dictionary attacks. Indeed, if no tweak is used, an attacker could build a dictionary of plaintext / ciphertext pairs and find with good probability the eavesdropped encrypted PANs. This attack works when the amount of data manipulated is small, which is particularly the case for PANs encryption. Using random tweak will render this dictionary technique useless as one dictionary per tweak value would be required.

When using the tweak capability of BPS, we suggest to apply a truncated hash function on the input tweak data. For example, in the scenario of PANs ciphering, one should use the non-ciphered digits of the PAN (usually the 6 first for routing purposes and the last 4 for receipt printing) as a tweak data. Moreover, several other informations, such as transaction date, transaction amount, etc. can be incorporated as tweak data as well. We therefore recommend to concatenate all the available tweak data and apply a secure hash function on this bit string and then truncate the obtained hash output to 64-bit, as required by BPS's input tweak T size.

References

1. Thomas Baignères, Jacques Stern, and Serge Vaudenay. Linear cryptanalysis of non binary ciphers. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 184–211. Springer, 2007.
2. M. Bellare, R. Canetti, and H. Krawczyk. Hmac: Keyed-hashing for message authentication. RFC 2104, February 1997.
3. M. Bellare, P. Rogaway, and T. Spies. The ffx mode of operation for format-preserving encryption. NIST submission, November 2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf.
4. Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In M.J. Jacobson Jr, V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*. Springer, 2009.

5. John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In Bart Preneel, editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2002.
6. Paul Crowley. Mercy: A fast large block cipher for disk sector encryption. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2000.
7. Horst Feistel. *Cryptography and Computer Privacy*, pages 15–23. Scientific American, 1973.
8. David Goldenberg, Susan Hohenberger, Moses Liskov, Elizabeth Crump Schwartz, and Hakan Seyalioglu. On tweaking luby-rackoff blockciphers. In *ASIACRYPT*, pages 342–356, 2007.
9. Louis Granboulan, Éric Levieil, and Gilles Piret. Pseudorandom permutation families over abelian groups. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 57–77. Springer, 2006.
10. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
11. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
12. Ben Morris, Phillip Rogaway, and Till Stegers. How to encipher messages on a small domain - deterministic encryption and the thorp shuffle. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 286–302. Springer, 2009.
13. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002. http://csrc.nist.gov.
14. National Institute of Standards and Technology. SP800-67: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, May 2004. http://csrc.nist.gov.
15. National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard, November 2001. http://csrc.nist.gov.
16. Jacques Patarin. Generic attacks on feistel schemes. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.
17. Jacques Patarin. Luby-rackoff: 7 rounds are enough for $2^{n(1\text{-epsilon})}$ security. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 513–529. Springer, 2003.
18. Jacques Patarin. Security of random feistel schemes with 5 or more rounds. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2004.
19. Jacques Patarin. A proof of security in o(2n) for the xor of two random permutations. In Reihaneh Safavi-Naini, editor, *ICITS*, volume 5155 of *Lecture Notes in Computer Science*, pages 232–248. Springer, 2008.
20. Rich Schroeppel. Hasty pudding cipher specification, June 1998. http://www.princeton.edu/∼rblee/HPC/index.htm.
21. T. Spies. Feistel finite set encryption. NIST submission, February 2008. http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html.