1 Syntax

[BLMR]'s proposed syntax involves a tuple

$$\mathsf{UpEnc} = (\mathsf{UpEnc.KeyGen}, \mathsf{UpEnc.Enc}, \mathsf{UpEnc.Dec}, \mathsf{UpEnc.ReKeyGen}, \mathsf{UpEnc.ReEnc})$$

, where $\mathsf{UpEnc.ReKeyGen}$ takes two keys (generated by $\mathsf{UpEnc.KeyGen}$) and generates a constant size "rekeying token", which will be passed into $\mathsf{UpEnc.ReEnc}$ together with the ciphertext encrypted under the first key, which will in turn output the ciphertext encrypted under the other key.

I don't think that this is necessary. I propose a syntax of the form

$$\mathsf{UpEnc} = (\mathsf{UpEnc.KeyGen}, \mathsf{UpEnc.Enc}, \mathsf{UpEnc.Dec}, \mathsf{UpEnc.ReEnc})$$

, where $\mathsf{UpEnc.ReEnc}$ instead takes the two keys and the ciphertext. I argue the reason behind this in a separate report.

We can define the specific syntax for the tuple of algorithms we propose

1. $KeyGen : \{\epsilon\} \to \{0,1\}^k$, a key generation algorithm returning a k bit long key, this should be randomized.

2. $Enc : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$, an encryption algorithm that takes a key and a string and returns a string, this should be randomized.

3. $Dec : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$, a decryption algorithm that takes a key and a string and returns a string, this should not be randomized.

4. $ReEnc : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$, an encryption algorithm that takes an old key, a new key, an encryption of a string under the old key and returns an encryption of the string under the new key. This should be randomized.

2 Correctness

This is the correctness condition

<u>$\mathbf{G}^{\mathrm{corr}}_{\mathsf{UpEnc},A}$</u>

- $i \leftarrow 0$
- $j \leftarrow 0$
- $j^* \leftarrow\$ A^{\textsc{Set},\textsc{KeyGen},\textsc{Enc}}()$
- $(M, C, k) \leftarrow M_{j^*}$
- **if** $k = \epsilon$ **then**
  - **return** `false`
- **end**
- $M' \leftarrow \mathsf{UpEnc.Dec}(C, K_k)$
- **return** $(M' = M)$

<u>$\textsc{KeyGen}()$</u>

- $K_i \leftarrow\$ \mathsf{UpEnc.KeyGen}$
- $i \leftarrow i + 1$
- **return** K_i

<u>$\textsc{Set}(M)$</u>

- $M_i \leftarrow (M, \epsilon, \epsilon)$
- $j \leftarrow j + 1$

<u>$\textsc{Enc}(i', j')$</u>

- **if** $(j' > j) \vee (i' > i)$ **then**
  - **return** $\perp$
- **end**
- $(M, C, k) \leftarrow M_{j'}$
- **if** $k = \epsilon$ **then**
  - $C \leftarrow\$ \mathsf{UpEnc.Enc}(\mathsf{K}_{\mathsf{i}'}, \mathsf{M})$
- **else**
  - $C \leftarrow\$ \mathsf{UpEnc.ReEnc}(K_k, K_{i'}, C)$
- **end**
- $M_{j'} \leftarrow (M, C, i')$
- **return** C

Then, we can say that UpEnc satisfies the correctness condition if, for all adversaries A we have that:

$$\Pr[\mathbf{G}^{\mathrm{corr}}_{\mathsf{UpEnc},A}] = 0$$

3 Rivest (1997)

$\underline{\mathbf{G}^{\mathrm{ind}}_{\mathsf{AONT}}(A)}$

$\quad b \leftarrow_{\$} \{0,1\}$

$\quad b' \leftarrow_{\$} A^{\mathrm{LR}}$

$\quad$ **return** $(b = b')$

$\underline{\mathrm{LR}(M, N, i)}$

$\quad$ **if** $|M| \neq |N|$ **then**

$\quad\quad$ **return** $\perp$

$\quad$ **end**

$\quad (m_1, m_2 \ldots m_s) \leftarrow M$

$\quad (n_1, n_2 \ldots n_s) \leftarrow N$

$\quad n_i \leftarrow \epsilon$

$\quad m_i \leftarrow \epsilon$

$\quad$ **if** $b = 0$ **then**

$\quad\quad y \leftarrow_{\$} \mathsf{AONT.Transform}(m_1, m_2 \ldots m_s)$

$\quad$ **else**

$\quad\quad y \leftarrow_{\$} \mathsf{AONT.Transform}(n_1, n_2 \ldots n_s)$

$\quad$ **end**

$\quad$ **return** y

Then we say that the indistinguishability adversary A has l-AONT-IND advantage:

$$\mathbf{Adv}^{\mathrm{aont\text{-}ind}}_{\mathsf{AONT}}(A) = 2 \cdot \Pr\left[\,\mathbf{G}^{\mathrm{ind}}_{\mathsf{AONT}}(A)\,\right] - 1$$

4 Boyko (1999)/ Canetti et. al (2000)

$\underline{\mathbf{G}^{\mathrm{leak}}_{\mathsf{AONT},l}(A)}$

$b \leftarrow_{\$} \{0,1\}$
$b' \leftarrow_{\$} A^{\mathrm{LR}}$
return $(b = b')$

$\underline{\mathrm{LR}(M, N, S)}$

if $|M| \neq |N|$ **then**
  return $\perp$
end
$(m_1, m_2 \dots m_s) \leftarrow M$
$(n_1, n_2 \dots n_s) \leftarrow N$
if $b = 0$ **then**
  $y \leftarrow_{\$} \mathsf{AONT.Transform}(m_1, m_2 \dots m_s)$
else
  $y \leftarrow_{\$} \mathsf{AONT.Transform}(n_1, n_2 \dots n_s)$
end
if $(|S| \neq |y|) \vee (\mathsf{Hamm}(S) > (|y| - l))$ **then**
  return $\perp$
else
  $y \leftarrow y \,\&\, S$
end
return y

Note that $|M|$ is the length of the string M in bits, & is a bitwise AND and $\mathsf{Hamm}(M)$ *takes the hamming weight of M*

Then we say that the leakage adversary A has l-AONT-LEAK advantage:

$$\mathbf{Adv}^{\text{aont-leak}}_{\mathsf{AONT},l}(A) = 2 \cdot \Pr\left[\, \mathbf{G}^{\mathrm{leak}}_{\mathsf{AONT},l}(A) \,\right] - 1$$

5 Leakage Resilience Model

$\underline{\mathbf{G}^{\mathrm{lr}}_{\mathsf{AONT},m}(A)}$

$b \leftarrow\!\!\$ \{0,1\}$
$b' \leftarrow\!\!\$ A^{\mathrm{LR}}$
return $(b = b')$

$\underline{\mathrm{LR}(M, N, C)}$

if $|M| \neq |N|$ **then**
| **return** $\perp$
end
$(m_1, m_2 \ldots m_s) \leftarrow M$
$(n_1, n_2 \ldots n_s) \leftarrow N$
if $b = 0$ **then**
| $y \leftarrow\!\!\$ \mathsf{AONT.Transform}(m_1, m_2 \ldots m_s)$
else
| $y \leftarrow\!\!\$ \mathsf{AONT.Transform}(n_1, n_2 \ldots n_s)$
end
if $(C \notin \mathcal{C}_{|y|,(|y|-m)})$ **then**
| **return** $\perp$
else
| **return** $C(y)$
end

Note that $\mathcal{C}_{n,m}$ is the set of boolean circuits taking n inputs and m outputs, expressed in a string in some reasonable encoding. Then, for $C \in \mathcal{C}_{n,m}$, when we run $C(S)$ for some binary string S of length n, C will take as input the bits of S and return a m bit long string.

Then we say that the leakage resilience adversary A has m-AONT-LR advantage:

$$\mathbf{Adv}^{\text{aont-lr}}_{\mathsf{AONT},m}(A) = 2 \cdot \Pr\left[\mathbf{G}^{\mathrm{lr}}_{\mathsf{AONT},m}(A)\right] - 1$$

6 Relationship between Notions

6.1 $\mathsf{AONT.bl}$-AONT-L $\Longrightarrow$ AONT-IND

Theorem 6.1 *For any AONT-IND adversary A, we can construct* $\mathsf{AONT.bl}$-*AONT-L adversary B, running in the same time and making the same number of queries, such that*

$$\boldsymbol{Adv}^{\textit{aont-ind}}_{\mathsf{AONT}}(A) \leq \boldsymbol{Adv}^{\textit{aont-leak}}_{\mathsf{AONT},\mathsf{AONT.bl}}(B)$$

Here is the adversary (the full proof is omitted for now):

$\underline{B^{\mathrm{LR}}}$

$b \leftarrow\!\!\$\; A^{\textsc{SimLR}}$

return b

$\underline{\textsc{SimLR}(M, N, i)}$

$mask \leftarrow \epsilon$

$s \leftarrow \lceil \frac{|M|}{\mathsf{AONT.bl}} \rceil$

for $j = 1, 2, \ldots s$ **do**

  if $j \neq i$ **then**

    $mask \leftarrow mask || 1^{\mathsf{AONT.bl}}$

  else

    $mask \leftarrow mask || 0^{\mathsf{AONT.bl}}$

  end

end

return $\mathrm{LR}(M, N, mask)$

6.2 l-AONT-L $\Longrightarrow$ l-AONT-LR

Theorem 6.2 *For any l-AONT-L adversary A, we can construct l-AONT-LR adversary B, running in the same time and making the same number of queries, such that*

$$\boldsymbol{Adv}^{\textit{aont-leak}}_{\mathsf{AONT},l}(A) \leq \boldsymbol{Adv}^{\textit{aont-lr}}_{\mathsf{AONT},l}(B)$$

Here is the adversary (the full proof is omitted for now):

$\underline{B^{\mathrm{LR}}}$

$b \leftarrow\!\!\$\; A^{\textsc{SimLR}}$

return b

$\underline{\textsc{SimLR}(M, N, mask)}$

return $\mathrm{LR}(M, N, C_{mask})$

$\underline{C_{mask}(X)}$

return $mask \,\&\, X$

6.3 AONT-IND $\not\Rightarrow$ $\mathsf{AONT.bl}$-AONT-L

For simplicity, we will use a trivial AONT to illustrate this, though this can also be done similarly with the "package transform" and other more practical AONTs. Consider the following AONT, $\mathsf{XOR} - \mathsf{AONT}$ with some sufficiently large block length to prevent exhaustive searches (e.g. 256 bits)

Then, we modify this to the following, with the same block length:
PACKAGE TRANSFORM IS NOT LEAKAGE RESISTANT

$\underline{\mathsf{XOR-AONT.Transform}(m_1, m_2, \ldots m_s)}$

for $i = 1, 2 \ldots s$ **do**
  for $j = 1, 2 \ldots \mathsf{XOR-AONT.bl} - 1$ **do**
    $n_{i,j} \leftarrow_{\$} \{0,1\}^{\mathsf{XOR-AONT.bl}}$
  end
  $n_{i,\mathsf{XOR-AONT.bl}} \leftarrow_{\$} m_i \oplus n_{i,1} \oplus n_{i,2} \cdots \oplus n_{i,\mathsf{XOR-AONT.bl}}$
end
return $(n_{1,1}, n_{1,2} \ldots n_{1,\mathsf{XOR-AONT.bl}}, n_{2,1}, n_{2,2} \ldots n_{s,\mathsf{XOR-AONT.bl}})$

$\underline{\mathsf{XOR-AONT.Inverse}(m'_1, m'_2 \ldots m'_{s'})}$

$(n_{1,1}, n_{1,2} \ldots n_{1,\mathsf{XOR-AONT.bl}}, n_{2,1}, n_{2,2} \ldots n_{s,\mathsf{XOR-AONT.bl}}) \leftarrow (m'_1, m'_2 \ldots m'_{s'})$
for $i = 1, 2, \ldots s$ **do**
  $m_i \leftarrow n_{i,1} \oplus n_{i,2} \oplus \cdots \oplus n_{i,\mathsf{XOR-AONT.bl}}$
end
return $(m_1, m_2 \ldots m_s)$

$\underline{\mathsf{MOD-XOR-AONT.Transform}(m_1, m_2, \ldots m_s)}$

for $i = 1, 2 \ldots s$ **do**
  for $j = 1, 2 \ldots \mathsf{MOD-XOR-AONT.bl} - 1$ **do**
    $n_{i,j} \leftarrow_{\$} \{0,1\}^{\mathsf{MOD-XOR-AONT.bl}}$
  end
  $n_{i,\mathsf{MOD-XOR-AONT.bl}} \leftarrow_{\$} m_i \oplus n_{i,1} \oplus n_{i,2} \cdots \oplus n_{i,\mathsf{MOD-XOR-AONT.bl}}$
end
$(m'_1, m'_2, \ldots m'_{s'}) \leftarrow$
$(n_{1,1}, n_{1,2} \ldots n_{1,\mathsf{MOD-XOR-AONT.bl}}, n_{2,1}, n_{2,2} \ldots n_{s,\mathsf{MOD-XOR-AONT.bl}})$
for $i = 1, 2 \ldots \mathsf{MOD-XOR-AONT.bl}$ **do**
  $b_i \leftarrow m'_i \,\&\, (0^{\mathsf{MOD-XOR-AONT.bl}-1}1)$
  $m'_i \leftarrow m'_i \,\&\, (1^{\mathsf{MOD-XOR-AONT.bl}-1}0)$
end
$m'_0 \leftarrow b_1 || b_2 \ldots b_{\mathsf{MOD-XOR-AONT.bl}}$
return $(m'_0, m'_1, m'_2, \ldots m'_{s'})$

$\underline{\mathsf{MOD-XOR-AONT.Inverse}(m'_0, m'_1 \ldots m'_{s'})}$

$b_1 || b_2 \ldots b_{\mathsf{MOD-XOR-AONT.bl}} \leftarrow m'_0$
for $i = 1, 2 \ldots \mathsf{MOD-XOR-AONT.bl}$ **do**
  $b_i \leftarrow b_i \,\&\, (0^{\mathsf{MOD-XOR-AONT.bl}-1}1)$
  $m'_i \leftarrow m'_i \oplus b_i$
end
$(n_{1,1}, n_{1,2} \ldots n_{1,\mathsf{MOD-XOR-AONT.bl}}, n_{2,1}, n_{2,2} \ldots n_{s,\mathsf{MOD-XOR-AONT.bl}}) \leftarrow$
$(m'_1, m'_2 \ldots m'_{s'})$
for $i = 1, 2, \ldots s$ **do**
  $m_i \leftarrow n_{i,1} \oplus n_{i,2} \oplus \cdots \oplus n_{i,\mathsf{XOR-AONT.bl}}$
end
return $(m_1, m_2 \ldots m_s)$