Simple 2-out-of-2 secret-sharing scheme:

Σ:		
$\mathcal{M} = \{0,1\}^\ell$ $t = 2$ $n = 2$	$\underline{\text{Share}(m):}$ $s_1 \leftarrow \{0,1\}^\ell$ $s_2 := s_1 \oplus m$ return (s_1, s_2)	$\underline{\text{Reconstruct}(s_1, s_2):}$ return $s_1 \oplus s_2$

Claim:

Σ is a secure 2-out-of-2 secret-sharing scheme. That is,

$$\mathcal{L}^{\Sigma}_{\text{tsss-L}} \equiv \mathcal{L}^{\Sigma}_{\text{tsss-R}}.$$

We will **use** the fact that one-time pad has one-time security ($\mathcal{L}^{\text{OTP}}_{\text{ots-L}} \equiv \mathcal{L}^{\text{OTP}}_{\text{ots-R}}$).

Overview:

Want to show:

$\mathcal{L}^{\Sigma}_{\text{tsss-L}}$		$\mathcal{L}^{\Sigma}_{\text{tsss-R}}$
$\textsc{query}(m_L, m_R, U)$: if $\lvert U \rvert \geq 2$: return err $\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_L)$ return $(s_i)_{i \in U}$	$\equiv$	$\textsc{query}(m_L, m_R, U)$: if $\lvert U \rvert \geq 2$: return err $\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_R)$ return $(s_i)_{i \in U}$

Standard hybrid technique:

- Starting with $\mathcal{L}^{\Sigma}_{\text{tsss-L}}$, make a sequence of small modifications
- Each modification has no effect on calling program / adversary
- Sequence of modifications ends with $\mathcal{L}^{\Sigma}_{\text{tsss-R}}$

Security proof

$\mathcal{L}^{\Sigma}_{\text{tsss-L}}$
$\underline{\text{QUERY}(m_L, m_R, U)}$:
if $\lvert U \rvert \geq 2$: return err
$\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_L)$
return $(s_i)_{i \in U}$

Starting point is $\mathcal{L}^{\Sigma}_{\text{tsss-L}}$.

$\mathcal{L}_{\text{tsss-L}}^{\Sigma}$

QUERY(m_L, m_R, U):
  if $|U| \geq 2$: return err
  $\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_L)$
  return $(s_i)_{i \in U}$

Starting point is $\mathcal{L}_{\text{tsss-L}}^{\Sigma}$. Fill in details of Σ

$\mathcal{L}^{\Sigma}_{\text{tsss-L}}$

QUERY(m_L, m_R, U):
  if $|U| \geq 2$: return err
  $s_1 \leftarrow \{0,1\}^\ell$
  $s_2 := s_1 \oplus m_L$
  return $(s_i)_{i \in U}$

Details of Σ filled in.

$\mathcal{L}^{\Sigma}_{\text{tsss-L}}$
$\underline{\text{QUERY}(m_L, m_R, U)\text{:}}$
if $\lvert U \rvert \geq 2$: return err
$s_1 \leftarrow \{0,1\}^\ell$
$s_2 := s_1 \oplus m_L$
return $(s_i)_{i \in U}$

Details of Σ filled in.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_2
  else return null
```

Duplicate body for the 3 possible unauthorized sets: $\{1\}, \{2\}, \emptyset$.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_2
  else return null
```

Duplicate body for the 3 possible unauthorized sets: $\{1\}, \{2\}, \emptyset$.

Security proof

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_2
  else return null
```

s_2 not used in this branch.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_2
  else return null
```

s_2 not used in this branch, so we can change how it is assigned.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_2
  else return null
```

s_2 not used in this branch, so we can change how it is assigned.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return s_2
  else return null
```

Recognize s_2 as OTP encryption of m_L.

Security proof

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_2 ← QUERY'(m_L, m_R)
    return s_2
  else return null
```

$\diamond$

$\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$

```
QUERY'(m_L, m_R):
  k ← {0,1}^ℓ
  c := k ⊕ m_L
  return c
```

Write it in terms of the “left” OTP security library.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_2 ← QUERY'(m_L, m_R)
    return s_2
  else return null
```

$\diamond$

$\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$

```
QUERY'(m_L, m_R):
  k ← {0,1}^ℓ
  c := k ⊕ m_L
  return c
```

Write it in terms of the “left” OTP security library.

Security proof

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_2 ← QUERY'(m_L, m_R)
    return s_2
  else return null
```

$\diamond$

$\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$

```
QUERY'(m_L, m_R):
  k ← {0,1}^ℓ
  c := k ⊕ m_R
  return c
```

OTP security says we can replace $\mathcal{L}_{\text{ots-L}}$ with $\mathcal{L}_{\text{ots-R}}$.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_2 ← QUERY′(m_L, m_R)
    return s_2
  else return null
```

$\diamond$

$\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$

```
QUERY′(m_L, m_R):
  k ← {0,1}^ℓ
  c := k ⊕ m_R
  return c
```

OTP security says we can replace $\mathcal{L}_{\text{ots-L}}$ with $\mathcal{L}_{\text{ots-R}}$.

$\textsc{query}(m_L, m_R, U)$:
    if $|U| \geq 2$: return err
    if $U = \{1\}$:
        $s_1 \leftarrow \{0,1\}^\ell$
        $s_2 := s_1 \oplus m_R$
        return s_1
    elsif $U = \{2\}$:
        $s_2 \leftarrow \textsc{query}'(m_L, m_R)$
        return s_2
    else return null

$\diamond$

$\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$

$\textsc{query}'(m_L, m_R)$:
    $k \leftarrow \{0,1\}^\ell$
    $c := k \oplus m_R$
    return c

Inline the subroutine call.

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_2
  else return null
```

Inline the subroutine call.

QUERY(m_L, m_R, U):

```
if |U| ≥ 2: return err
if U = {1}:
    s1 ← {0,1}^ℓ
    s2 := s1 ⊕ mR
    return s1
elsif U = {2}:
    s1 ← {0,1}^ℓ
    s2 := s1 ⊕ mR
    return s2
else return null
```

Inline the subroutine call.

QUERY(m_L, m_R, U):

```
if |U| ≥ 2: return err
if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_1
elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return s_2
else return null
```

Three branches of if-statement can be unified.

$\textsc{query}(m_L, m_R, U)$:

```
if |U| ≥ 2: return err
s_1 ← {0,1}^ℓ
s_2 := s_1 ⊕ m_R
return (s_i)_{i∈U}
```

$$\begin{array}{l}\text{if } |U| \geq 2\text{: return } \texttt{err}\\ s_1 \leftarrow \{0,1\}^\ell\\ s_2 := s_1 \oplus m_R\\ \text{return } (s_i)_{i \in U}\end{array}$$

Three branches of if-statement can be unified..

$\textsc{query}(m_L, m_R, U)$:

 if $|U| \geq 2$: return err

 $s_1 \leftarrow \{0,1\}^\ell$

 $s_2 := s_1 \oplus m_R$

 return $(s_i)_{i \in U}$

Three branches of if-statement can be unified..

```
QUERY(m_L, m_R, U):
  if |U| ≥ 2: return err
  s_1 ← {0,1}^ℓ
  s_2 := s_1 ⊕ m_R
  return (s_i)_{i∈U}
```

This happens to be $\mathcal{L}^{\Sigma}_{\text{tsss-R}}$.

$\mathcal{L}^{\Sigma}_{\text{tsss-R}}$
$\underline{\text{QUERY}(m_L, m_R, U)}$:
if $\lvert U\rvert \geq 2$: return `err`
$\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_R)$
return $(s_i)_{i \in U}$

This happens to be $\mathcal{L}^{\Sigma}_{\text{tsss-R}}$.

$\mathcal{L}^{\Sigma}_{\text{tsss-R}}$
QUERY(m_L, m_R, U):
if $\lvert U \rvert \geq 2$: return err
$\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_R)$
return $(s_i)_{i \in U}$

This happens to be $\mathcal{L}^{\Sigma}_{\text{tsss-R}}$.

Summary

We showed:

$\mathcal{L}^{\Sigma}_{\text{tsss-L}}$		$\mathcal{L}^{\Sigma}_{\text{tsss-R}}$
$\underline{\text{QUERY}(m_L, m_R, U)}$: if $\lvert U \rvert \geq 2$: return err $\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_L)$ return $(s_i)_{i \in U}$	$\equiv \cdots \equiv$	$\underline{\text{QUERY}(m_L, m_R, U)}$: if $\lvert U \rvert \geq 2$: return err $\boldsymbol{s} \leftarrow \Sigma.\text{Share}(m_R)$ return $(s_i)_{i \in U}$

So Σ is a secure 2-out-of-2 secret-sharing scheme.

Generalization:

If $\mathcal{E}$ is **any** encryption scheme with one-time secrecy, then the following is a secure 2-out-of-2 threshold secret sharing scheme:

$$\mathcal{M} = \mathcal{E}.\mathcal{M}$$
$$t = 2$$
$$n = 2$$

Share(m):
- $s_1 \leftarrow \mathcal{E}.\text{KeyGen}$
- $s_2 := \mathcal{E}.\text{Enc}(s_1, m)$
- return (s_1, s_2)

Reconstruct(s_1, s_2):
- return $\mathcal{E}.\text{Dec}(s_1, s_2)$