MODULE *Operators*

This module defines a collection of helper functions and operators used by the protocol module.
It includes :
− Random hash generation and basic arithmetic operators
- String and sequence manipulation operators
- Cryptographic abstractions for key derivation, memo encryption, and memo decryption

```
LOCAL INSTANCE Randomization
LOCAL INSTANCE FiniteSets
LOCAL INSTANCE Sequences
LOCAL INSTANCE Naturals
```

Basic Arithmetic & Utility

Modulo operator.

```
$Mod(a, b) \triangleq a - (b * (a \div b))$
```

Minimum of two numbers.

```
$Min(a, b) \triangleq$ IF $a \leq b$ THEN $a$ ELSE $b$
```

Pad a sequence with "0" characters to length n.

```
$Pad(n) \triangleq [i \in 1 \,..\, n \mapsto \text{"0"}]$
```

Sequence Manipulation Helpers

Returns the last element of a sequence, or the empty sequence if none.

```
$Last(seq) \triangleq$ IF $Len(seq) = 0$ THEN $\langle\rangle$ ELSE $seq[Len(seq)]$
```

Split a sequence (*e.g.*, a memo) into chunks of size *chunk_size*.
Pads the final chunk with zeros if necessary.

```
$SplitAndPadMemo(memo, chunk\_size) \triangleq$
    LET $numChunks \triangleq$ IF $(Mod(Len(memo), chunk\_size) = 0)$
                              THEN $Len(memo) \div chunk\_size$
                              ELSE $(Len(memo) \div chunk\_size) + 1$
    IN  $[i \in 1 \,..\, numChunks \mapsto$
            LET $start \triangleq (i - 1) * chunk\_size + 1$
                $stop  \triangleq Min(i * chunk\_size, Len(memo))$
                $chunk \triangleq SubSeq(memo, start, stop)$
            IN  IF $Len(chunk) < chunk\_size$
                 THEN $chunk \circ Pad(chunk\_size - Len(chunk))$
                 ELSE $chunk]$
```

Recursively removes trailing "0" characters from a sequence.

```
RECURSIVE $RemoveTrailingZeros(\_)$
$RemoveTrailingZeros(seq) \triangleq$
    IF $seq = \langle\rangle$ THEN $\langle\rangle$
     ELSE IF $Last(seq) = \text{"0"}$ THEN $RemoveTrailingZeros(SubSeq(seq, 1, Len(seq) - 1))$
            ELSE $seq$
```

Flattens a sequence of sequences into a single sequence.

```
RECURSIVE Flatten(_)
Flatten(seqOfSeqs) ≜
    IF seqOfSeqs = ⟨⟩ THEN ⟨⟩
     ELSE Head(seqOfSeqs) ∘ Flatten(Tail(seqOfSeqs))
```

Cryptographic Abstractions

Generate a random hash (abstractly modeled as a random sequence of bytes) of length n.

```
RandomHash(n) ≜ [i ∈ 1 . . n ↦ CHOOSE x ∈ 0 . . 255 : TRUE]
```

A simplified model of the key derivation function.

In a real system, this would be a secure *PRF* applied to a constant concatenated with the salt.

```
EncryptionKey(memo_key, salt) ≜ [memo_key ↦ memo_key, salt ↦ salt, randomness ↦ RandomHash(2)]
```

Encrypt a memo chunk using an encryption key and a nonce.

Here the nonce is abstracted as the chunk index.

```
EncryptMemoChunk(encryption_key, i, chunk) ≜
    [encryption_key ↦ encryption_key, nonce ↦ i, chunk ↦ chunk]
```

Encrypt a memo (a set of chunks) using the derived encryption key.

```
EncryptMemo(encryption_key, chunks) ≜
    [i ∈ DOMAIN chunks ↦ EncryptMemoChunk(encryption_key, i, chunks[i])]
```

Decrypt a memo chunk using the memo key and salt.

```
DecryptMemoChunk(memo_key, salt, encrypted_chunk) ≜
    IF EncryptionKey(memo_key, salt) = encrypted_chunk.encryption_key
         THEN encrypted_chunk.chunk
         ELSE "decryption failed"
```

Decrypt all memo chunks using the memo key and salt.

```
DecryptMemo(memo_key, salt, encrypted_chunks) ≜
    [i ∈ DOMAIN encrypted_chunks ↦ DecryptMemoChunk(memo_key, salt, encrypted_chunks[i])]
```

High-Level Memo Processing

Given a sequence of decrypted memo chunks, removes trailing zeros from the final chunk and concatenates all chunks into a single sequence.

```
DecryptedMemoFinal(decryptedChunks) ≜
    LET lastChunk ≜ RemoveTrailingZeros(decryptedChunks[Len(decryptedChunks)])
        allButLast ≜ IF Len(decryptedChunks) > 1
                       THEN SubSeq(decryptedChunks, 1, Len(decryptedChunks) − 1)
                       ELSE ⟨⟩
    IN  Flatten(allButLast) ∘ lastChunk
```

Verify that a transaction is valid.

```
VerifyTx(tx) ≜ TRUE
```

$ToSet(s) \triangleq$

The image of the given sequence s. $Cardinality(ToSet(s)) \leq Len(s)$ see https:$//en.wikipedia.org/$wiki$/Image_(mathematics)$

$\{s[i] : i \in \text{DOMAIN } s\}$