MODULE *Utils*

Utility functiuons used by the sinsemilla hash algorithm, they represent either functions that can probably be found in programming languages or functions that are specific to the sinsemilla hash algorithm that need to be coded.

EXTENDS *CharUtils*, *Community*

LOCAL INSTANCE *Sequences*

LOCAL INSTANCE *Naturals*

LOCAL INSTANCE *FiniteSets*

LOCAL INSTANCE *Randomization*

Convert a byte to a sequence of bits.

$$ByteToBitSequence(b) \triangleq$$
LET
$$Bit(i) \triangleq \text{IF } b \div (2^i)\%2 = 1 \text{ THEN } 1 \text{ ELSE } 0$$
IN
$$\langle Bit(7),\ Bit(6),\ Bit(5),\ Bit(4),\ Bit(3),\ Bit(2),\ Bit(1),\ Bit(0)\rangle$$

Convert a sequence of bits to a byte.

$$BitSequenceToByte(bits) \triangleq$$
LET
$$BitValue(pos) \triangleq \text{IF } bits[pos+1] = 1 \text{ THEN } 2^{(7-pos)} \text{ ELSE } 0$$
IN
$$BitValue(0) + BitValue(1) + BitValue(2) + BitValue(3) + BitValue(4) + BitValue(5) + BitValue(6) + BitValue(7)$$

Maximum of two numbers.

$$Max(a,\ b) \triangleq \text{IF } a \geq b \text{ THEN } a \text{ ELSE } b$$

Convert a sequence of characters to a sequence of bytes.

$$CharactersToBytes(characters) \triangleq [char \in 1\ ..\ Len(characters) \mapsto Ord(characters[char])]$$

Convert a sequence of bytes to a flat sequence of bits.

$$BytesToBits(bytes) \triangleq FlattenSeq([byte \in 1\ ..\ Len(bytes) \mapsto ByteToBitSequence(bytes[byte])])$$

Convert a sequence of bytes to a sequence of characters.

$$BytesToCharacters(bytes) \triangleq [b \in 1\ ..\ Len(bytes) \mapsto Chr(bytes[b])]$$

Convert a *Pallas* point to a sequence of fixed bytes. Here we just use the point coordinates as bytes.

$$PointToBytes(point) \triangleq \langle point.a,\ point.b\rangle$$

Integer to Little-Endian Octet String Pairing.

This procedure assumes $k = 10$, so we have 8 bits to build the first byte and 2 bits for the second. The second byte is formed by the first two bits of the second byte of the input and 6 zeros. We reach the 32 bytes by adding two zero bytes at the end.

This algorithm is the one implemented in Zebra.

$$IntToLEOSP32(bits) \triangleq \langle$$
$$BitSequenceToByte(SubSeq(bits,\ 1,\ 8)),$$

$BitSequenceToByte(\langle SubSeq(bits,\ 9,\ 10)[1],\ SubSeq(bits,\ 9,\ 10)[2],\ 0,\ 0,\ 0,\ 0,\ 0,\ 0\rangle),$
$0,$
$0\rangle$

The incomplete addition operator. Sums the x and y coordinates of two points on the *Pallas* curve.

$IncompleteAddition(x,\ y) \triangleq [a \mapsto x.a + y.a,\ b \mapsto x.b + y.b]$

Pad bits with zeros to make length a multiple of k

$PadBits(bits,\ k) \triangleq bits \circ [index \in 1\ ..\ (k - (Len(bits)\%k)) \mapsto 0]$

Divide padded bits into chunks of k bits

$DivideInChunks(paddedBits,\ k) \triangleq [index \in 1\ ..\ (Len(paddedBits) \div k) \mapsto SubSeq(paddedBits,\ (index - 1) * k$

Produce a Pallas point with the separator and message bytes stored in separator and message_bytes.
Here we decouple the input message and separator from the outputs by choosing random coordinates.
From now on, in this model, we can t releate the original message with the ciphertext anymore.

$HashToPallas(separator,\ message\_bytes) \triangleq [$
$\quad a \mapsto \text{CHOOSE } r \in RandomSubset(1,\ 1\ ..\ 3) : \text{TRUE},$
$\quad b \mapsto \text{CHOOSE } r \in RandomSubset(1,\ 1\ ..\ 3) : \text{TRUE}$
$]$

Ceiling division.

$CeilDiv(a,\ b) \triangleq \text{IF } a\%b = 0 \text{ THEN } a \div b \text{ ELSE } (a \div b) + 1$