MODULE *sinsemilla*

Sinsemilla hash function specification

Specifies what is needed to implement a *sinsemilla* hash function algorithm.

EXTENDS *TLC*, *Naturals*, *Integers*, *Sequences*, *Utils*, *Randomization*

--algorithm *sinsemilla*

variables

Holder for a point on the *Pallas* curve.

$point = [a \mapsto 0,\ b \mapsto 0]$;

Holder for a sequence of characters.

$characters = \langle\rangle$;

Holder for a sequence of bytes.

$bytes = \langle\rangle$;

Holder for a sequence of bytes when the bytes variable is already busy.

$auxiliar\_bytes = \langle\rangle$;

Holder for a sequence of bits.

$bits = \langle\rangle$;

Holder for a sequence of slices.

$slices = \langle\rangle$;

define

The number of bits in a chunk.

$k \triangleq 10$

The domain separator string for the Q point: "*z.cash.SinsemillaQ*".

$SinsemillaQ \triangleq$
$\langle$"z", ".", "c", "a", "s", "h", ".", "S", "i", "n", "s", "e", "m", "i", "l", "l", "a", "Q"$\rangle$

The domain separator string for the S point: "*z.cash.SinsemillaS*".

$SinsemillaS \triangleq$
$\langle$"z", ".", "c", "a", "s", "h", ".", "S", "i", "n", "s", "e", "m", "i", "l", "l", "a", "S"$\rangle$

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

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

$TypeInvariantPoint \triangleq point \in [a : Nat,\ b : Nat]$
$TypeInvariantCharacters \triangleq characters \in Seq(\text{STRING})$
$TypeInvariantBytes \triangleq bytes \in Seq(Nat)$
$TypeInvariantAuxiliarBytes \triangleq bytes \in Seq(Nat)$
$TypeInvariantBits \triangleq bits \in Seq(\{0, 1\})$
$TypeInvariantSlices \triangleq slices \in Seq(Seq(\{0, 1\}))$

$InvType \triangleq TypeInvariantPoint \land TypeInvariantCharacters \land TypeInvariantBytes$
$\land\ TypeInvariantBytes \land TypeInvariantBits \land TypeInvariantSlices$

Liveness property stating that the point holder will eventually end up with a point different than the starting one.

$Liveness \triangleq \Diamond(point \neq [a \mapsto 0,\ b \mapsto 0])$

Bytes should always be a sequence of integers representing bytes.

$SafetyBytesSequence \triangleq \quad \wedge bytes = \langle\rangle \vee (\forall\, i \in 1 \,..\, Len(bytes) : bytes[i] \in 0 \,..\, 255)$

Slices should always be a sequence of sequences of bits and each slice should have no length greater than k. We only can have a slice with length $<$ than k when we are building the slices in the "PadLastSlice" label of the pad procedure.

$SafetySlicesSequence \triangleq$
$\quad \wedge slices = \langle\rangle \vee (\forall\, i \in 1 \,..\, Len(slices) : slices[i] \in Seq(\{0, 1\}) \wedge Len(slices[i]) \leq k)$

$Safety \triangleq SafetyBytesSequence \wedge SafetySlicesSequence$

end define ;

Convert a sequence of characters to a sequence of bytes.

macro *characters_to_bytes*()
begin
$bytes := [c \in 1 \,..\, Len(characters) \mapsto Ord(characters[c])]$ **;**
end macro ;

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

macro *bytes_to_bits*()
begin
$bits := FlattenSeq([byte \in 1 \,..\, Len(bytes) \mapsto ByteToBitSequence(bytes[byte])])$ **;**
end macro ;

Convert a sequence of bytes to a a sequence of characters.

macro *bytes_to_characters*()
begin
$characters := [b \in 1 \,..\, Len(bytes) \mapsto Chr(bytes[b])]$ **;**
end macro ;

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

macro *point_to_bytes*()
begin
$bytes := \langle point.a,\, point.b\rangle$ **;**
end macro ;

The main procedure that hashes a message using the *Sinsemilla* hash function.

procedure *sinsemilla_hash*(*domain*, *message*)
begin
Encode the domain characters as bytes and store them in *auxiliar_bytes* for later use.
EncodeDomain:
$characters := domain$ **;**
characters_to_bytes() **;**
$auxiliar\_bytes := bytes$ **;**
Encode the message characters as bits and store them in bits for later use.
EncodeMessage:
$characters := message$ **;**

characters_to_bytes() ;
bytes_to_bits() ;

With the domain bytes in bytes and the message bits in bits, call the main procedure to hash the message.

SinsemillaHashToPoint:
bytes := *auxiliar_bytes* ;
call *sinsemilla_hash_to_point*() ;

Decode the point coordinates to characters.

DecodeCipherText:
point_to_bytes() ;
bytes_to_characters() ;

Return:
print *characters* ;
return ;
end procedure ;

Convert the message bits into a *Pallas* point, using the domain bytes stored in bytes as the domain separator and the message bits stored in bits as the message.

procedure *sinsemilla_hash_to_point*()
variables

The number of chunks in the message.

$n = Len(bits) \div k$,

The accumulator point.

accumulator,

The index of the current slice to be used in the main loop.

$i = 1$;
begin

CallPad:

Use the global bits as input and get slices in slices.

call *pad*(n) ;

CallQ:

Produce a *Pallas* point with the bytes stored, these bytes are set in the caller as domain bytes.

call q() ;

InitializeAcc:

With the point we got from calling q, initialize the accumulator.

accumulator := *point* ;

MainLoop:

Loop through the slices.

while $i \leq n$ **do**

CallS:

Produce a *Pallas* point calling s given the padded bits (10 bits).

bits := *slices*[i] ;
call s() ;

Accumulate:

Incomplete addition of the accumulator and the point.

accumulator :=

```
                IncompleteAddition(IncompleteAddition(accumulator, point), accumulator);
            IncrementIndex:
                i := i + 1;
        end while ;
    AssignAccumulatorToPoint:
        point := accumulator;
    return;
end procedure ;
```

Pad the message bits with zeros until the length is a multiple of k. Create chunks of k bits.

```
procedure pad(n)
begin
    GetSlices:
        slices := [index ∈ 1 .. n ↦ IF (index * k + k) ≥ Len(bits) THEN
            SubSeq(bits, index * k, Len(bits))
         ELSE SubSeq(bits, index * k, index * k + k − 1)];
    PadLastSlice:
        slices[Len(slices)] := [index ∈ 1 .. k ↦ IF index ≤ Len(slices[Len(slices)]) THEN
            slices[Len(slices)][index]
         ELSE 0];
    return;
end procedure ;
```

Produce a *Pallas* point with the bytes stored in bytes, these bytes are set in the caller as domain bytes.

```
procedure q()
begin
    Q:
        call hash_to_pallas(SinsemillaQ, bytes);
    return;
end procedure ;
```

Produce a *Pallas* point given the padded bits (10 bits). First we call *IntToLEOSP* on the bits and then we call *hash_to_pallas* with the result.

```
procedure s()
begin
    CallI2LEOSP:
        call IntToLEOSP32();
    S:
        call hash_to_pallas(SinsemillaS, bytes);
    return;
end procedure ;
```

Produce a *Pallas* point with the separator and message bytes stored in separator and *message_bytes*.

```
procedure hash_to_pallas(separator, message_bytes)
begin
    HashToPallas:
```

```
        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.
      point := [
          a ↦ CHOOSE r ∈ RandomSubset(1, 1 .. 3) : TRUE,
          b ↦ CHOOSE r ∈ RandomSubset(1, 1 .. 3) : TRUE
      ];
  return;
end procedure ;
```

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 zeros at the end.

This algorithm is the one implemented in Zebra.

```
procedure IntToLEOSP32()
begin
  IntToLEOSP:
      bytes := ⟨
          BitSequenceToByte(SubSeq(bits, 1, 8)),
          BitSequenceToByte(⟨SubSeq(bits, 9, 10)[1], SubSeq(bits, 9, 10)[2], 0, 0, 0, 0, 0, 0⟩),
          0,
          0
      ⟩   ;
  return;
end procedure ;
```

Call the main procedure with the domain and message. Strings are represented as sequences of characters.

```
fair process main = "MAIN"
begin
  SinSemillaHashCall:
      call sinsemilla_hash(
          ⟨"t", "e", "s", "t", " ", "S", "i", "n", "s", "e", "m", "i", "l", "l", "a"⟩,
          ⟨"m", "e", "s", "s", "a", "g", "e"⟩
      );
end process ;
end algorithm ;
```

BEGIN TRANSLATION ($chksum(pcal) =$ "85f15526" $\wedge$ $chksum(tla) =$ "944187a6")

Procedure variable n of procedure *sinsemilla_hash_to_point* at line 121 col 5 changed to $n\_$

CONSTANT *defaultInitValue*

VARIABLES *point*, *characters*, *bytes*, *auxiliar_bytes*, *bits*, *slices*, *pc*, *stack*

define statement

$k \triangleq 10$

$SinsemillaQ \triangleq$
⟨"z", ".", "c", "a", "s", "h", ".", "S", "i", "n", "s", "e", "m", "i", "l", "l", "a", "Q"⟩

$SinsemillaS \triangleq$
$\langle$ "z", ".", "c", "a", "s", "h", ".", "S", "i", "n", "s", "e", "m", "i", "l", "l", "a", "S" $\rangle$

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

$TypeInvariantPoint \triangleq point \in [a : Nat,\ b : Nat]$
$TypeInvariantCharacters \triangleq characters \in Seq(\text{STRING})$
$TypeInvariantBytes \triangleq bytes \in Seq(Nat)$
$TypeInvariantAuxiliarBytes \triangleq bytes \in Seq(Nat)$
$TypeInvariantBits \triangleq bits \in Seq(\{0, 1\})$
$TypeInvariantSlices \triangleq slices \in Seq(Seq(\{0, 1\}))$

$InvType \triangleq TypeInvariantPoint \land TypeInvariantCharacters \land TypeInvariantBytes$
$\land\ TypeInvariantBytes \land TypeInvariantBits \land TypeInvariantSlices$

$Liveness \triangleq \Diamond(point \neq [a \mapsto 0,\ b \mapsto 0])$

$SafetyBytesSequence \triangleq \quad \land\ bytes = \langle\rangle \lor (\forall\, i \in 1\,..\,Len(bytes) : bytes[i] \in 0\,..\,255)$

$SafetySlicesSequence \triangleq$
$\land\ slices = \langle\rangle \lor (\forall\, i \in 1\,..\,Len(slices) : slices[i] \in Seq(\{0, 1\}) \land Len(slices[i]) \leq k)$

$Safety \triangleq SafetyBytesSequence \land SafetySlicesSequence$

VARIABLES $domain,\ message,\ n\_,\ accumulator,\ i,\ n,\ separator,\ message\_bytes$

$vars \triangleq \langle point,\ characters,\ bytes,\ auxiliar\_bytes,\ bits,\ slices,\ pc,\ stack,$
$domain,\ message,\ n\_,\ accumulator,\ i,\ n,\ separator,\ message\_bytes$
$\rangle$

$ProcSet \triangleq$ { "MAIN" }

$Init \triangleq$ Global variables
$\land\ point = [a \mapsto 0,\ b \mapsto 0]$
$\land\ characters = \langle\rangle$
$\land\ bytes = \langle\rangle$
$\land\ auxiliar\_bytes = \langle\rangle$
$\land\ bits = \langle\rangle$
$\land\ slices = \langle\rangle$
Procedure *sinsemilla_hash*
$\land\ domain = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ message = [self \in ProcSet \mapsto defaultInitValue]$
Procedure *sinsemilla_hash_to_point*
$\land\ n\_ = [self \in ProcSet \mapsto Len(bits) \div k]$

$\land\ accumulator = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ i = [self \in ProcSet \mapsto 1]$
Procedure pad
$\land\ n = [self \in ProcSet \mapsto defaultInitValue]$
Procedure *hash_to_pallas*
$\land\ separator = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ message\_bytes = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ stack = [self \in ProcSet \mapsto \langle\rangle]$
$\land\ pc = [self \in ProcSet \mapsto$ "SinSemillaHashCall"$]$

$EncodeDomain(self) \triangleq$
$\land\ pc[self] =$ "EncodeDomain"
$\land\ characters' = domain[self]$
$\land\ bytes' = [c \in 1 \,..\, Len(characters') \mapsto Ord(characters'[c])]$
$\land\ auxiliar\_bytes' = bytes'$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "EncodeMessage"$]$
$\land$ UNCHANGED $\langle point, bits, slices, stack, domain,$
$message, n\_, accumulator, i, n,$
$separator, message\_bytes\rangle$

$EncodeMessage(self) \triangleq$
$\land\ pc[self] =$ "EncodeMessage"
$\land\ characters' = message[self]$
$\land\ bytes' = [c \in 1 \,..\, Len(characters') \mapsto Ord(characters'[c])]$
$\land\ bits' = FlattenSeq([byte \in 1 \,..\, Len(bytes') \mapsto ByteToBitSequence(bytes'[byte]$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "SinsemillaHashToPoint"$]$
$\land$ UNCHANGED $\langle point, auxiliar\_bytes, slices, stack,$
$domain, message, n\_, accumulator, i, n,$
$separator, message\_bytes\rangle$

$SinsemillaHashToPoint(self) \triangleq$
$\land\ pc[self] =$ "SinsemillaHashToPoint"
$\land\ bytes' = auxiliar\_bytes$
$\land\ stack' = [stack$ EXCEPT $![self] = \langle[procedure \mapsto$ "sinsemilla_hash_to_p
$pc \mapsto$ "DecodeCipherText",
$n\_ \mapsto n\_[self],$
$accumulator \mapsto accumulator[self],$
$i \mapsto i[self]]\rangle$
$\circ\ stack[self]]$
$\land\ n\_' = [n\_$ EXCEPT $![self] = Len(bits) \div k]$
$\land\ accumulator' = [accumulator$ EXCEPT $![self] = defaultInitValue]$
$\land\ i' = [i$ EXCEPT $![self] = 1]$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "CallPad"$]$
$\land$ UNCHANGED $\langle point, characters,$
$auxiliar\_bytes, bits, slices,$
$domain, message, n, separator,$
$message\_bytes\rangle$

$DecodeCipherText(self) \triangleq \land\ pc[self] =$ "DecodeCipherText"

$$
\begin{array}{l}
\land\ bytes' = \langle point.a,\ point.b\rangle \\
\land\ characters' = [b \in 1\ ..\ Len(bytes') \mapsto Chr(bytes'[b])] \\
\land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“Return”}] \\
\land\ \text{UNCHANGED}\ \langle point,\ auxiliar\_bytes,\ bits,\ slices, \\
\qquad\qquad\qquad stack,\ domain,\ message,\ n\_, \\
\qquad\qquad\qquad accumulator,\ i,\ n,\ separator, \\
\qquad\qquad\qquad message\_bytes\rangle
\end{array}
$$

$$
\begin{array}{lll}
Return(self) & \triangleq & \land\ pc[self] = \text{“Return”} \\
& & \land\ PrintT(characters) \\
& & \land\ pc' = [pc\ \text{EXCEPT}\ ![self] = Head(stack[self]).pc] \\
& & \land\ domain' = [domain\ \text{EXCEPT}\ ![self] = Head(stack[self]).domain] \\
& & \land\ message' = [message\ \text{EXCEPT}\ ![self] = Head(stack[self]).message] \\
& & \land\ stack' = [stack\ \text{EXCEPT}\ ![self] = Tail(stack[self])] \\
& & \land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,\ auxiliar\_bytes,\ bits, \\
& & \qquad\qquad slices,\ n\_,\ accumulator,\ i,\ n,\ separator, \\
& & \qquad\qquad message\_bytes\rangle
\end{array}
$$

$$
\begin{array}{lll}
sinsemilla\_hash(self) & \triangleq & EncodeDomain(self) \lor EncodeMessage(self) \\
& & \qquad \lor SinsemillaHashToPoint(self) \\
& & \qquad \lor DecodeCipherText(self) \lor Return(self)
\end{array}
$$

$$
\begin{array}{lll}
CallPad(self) & \triangleq & \land\ pc[self] = \text{“CallPad”} \\
& & \land\ \land\ n' = [n\ \text{EXCEPT}\ ![self] = n\_[self]] \\
& & \quad\ \land\ stack' = [stack\ \text{EXCEPT}\ ![self] = \langle [procedure \mapsto \text{“pad”}, \\
& & \qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{“CallQ”}, \\
& & \qquad\qquad\qquad\qquad\qquad\qquad n \mapsto n[self]] \rangle \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[self]] \\
& & \land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“GetSlices”}] \\
& & \land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,\ auxiliar\_bytes, \\
& & \qquad\qquad bits,\ slices,\ domain,\ message,\ n\_, \\
& & \qquad\qquad accumulator,\ i,\ separator,\ message\_bytes\rangle
\end{array}
$$

$$
\begin{array}{lll}
CallQ(self) & \triangleq & \land\ pc[self] = \text{“CallQ”} \\
& & \land\ stack' = [stack\ \text{EXCEPT}\ ![self] = \langle [procedure \mapsto \text{“q”}, \\
& & \qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{“InitializeAcc”}] \rangle \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[self]] \\
& & \land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“Q”}] \\
& & \land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,\ auxiliar\_bytes,\ bits, \\
& & \qquad\qquad slices,\ domain,\ message,\ n\_,\ accumulator,\ i,\ n, \\
& & \qquad\qquad separator,\ message\_bytes\rangle
\end{array}
$$

$$
\begin{array}{lll}
InitializeAcc(self) & \triangleq & \land\ pc[self] = \text{“InitializeAcc”} \\
& & \land\ accumulator' = [accumulator\ \text{EXCEPT}\ ![self] = point] \\
& & \land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“MainLoop”}] \\
& & \land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,
\end{array}
$$

$auxiliar\_bytes$, $bits$, $slices$, $stack$,
$domain$, $message$, $n\_$, i, n, $separator$,
$message\_bytes\rangle$

$MainLoop(self) \triangleq \land pc[self] = \text{“MainLoop”}$
$\land$ IF $i[self] \leq n\_[self]$
THEN $\land pc' = [pc$ EXCEPT $![self] = \text{“CallS”}]$
ELSE $\land pc' = [pc$ EXCEPT $![self] = \text{“AssignAccumulatorToPoint”}]$
$\land$ UNCHANGED $\langle point, characters, bytes, auxiliar\_bytes,$
$bits, slices, stack, domain, message, n\_,$
$accumulator, i, n, separator, message\_bytes\rangle$

$CallS(self) \triangleq \land pc[self] = \text{“CallS”}$
$\land bits' = slices[i[self]]$
$\land stack' = [stack$ EXCEPT $![self] = \langle[procedure \mapsto \text{“s”},$
$pc \mapsto \text{“Accumulate”}]\rangle$
$\circ stack[self]]$
$\land pc' = [pc$ EXCEPT $![self] = \text{“CallI2LEOSP”}]$
$\land$ UNCHANGED $\langle point, characters, bytes, auxiliar\_bytes,$
$slices, domain, message, n\_, accumulator, i, n,$
$separator, message\_bytes\rangle$

$Accumulate(self) \triangleq \land pc[self] = \text{“Accumulate”}$
$\land accumulator' = [accumulator$ EXCEPT $![self] = IncompleteAddition(IncompleteAd$
$\land pc' = [pc$ EXCEPT $![self] = \text{“IncrementIndex”}]$
$\land$ UNCHANGED $\langle point, characters, bytes, auxiliar\_bytes,$
$bits, slices, stack, domain, message, n\_,$
$i, n, separator, message\_bytes\rangle$

$IncrementIndex(self) \triangleq \land pc[self] = \text{“IncrementIndex”}$
$\land i' = [i$ EXCEPT $![self] = i[self] + 1]$
$\land pc' = [pc$ EXCEPT $![self] = \text{“MainLoop”}]$
$\land$ UNCHANGED $\langle point, characters, bytes,$
$auxiliar\_bytes, bits, slices, stack,$
$domain, message, n\_, accumulator, n,$
$separator, message\_bytes\rangle$

$AssignAccumulatorToPoint(self) \triangleq \land pc[self] = \text{“AssignAccumulatorToPoint”}$
$\land point' = accumulator[self]$
$\land pc' = [pc$ EXCEPT $![self] = Head(stack[self]).pc]$
$\land n\_' = [n\_$ EXCEPT $![self] = Head(stack[self]).n\_]$
$\land accumulator' = [accumulator$ EXCEPT $![self] = Head(stack[self]).$
$\land i' = [i$ EXCEPT $![self] = Head(stack[self]).i]$
$\land stack' = [stack$ EXCEPT $![self] = Tail(stack[self])]$
$\land$ UNCHANGED $\langle characters, bytes,$
$auxiliar\_bytes, bits, slices,$

$domain,\ message,\ n,$
$separator,\ message\_bytes\rangle$

$sinsemilla\_hash\_to\_point(self) \triangleq\ CallPad(self) \lor CallQ(self)$
$\lor\ InitializeAcc(self)$
$\lor\ MainLoop(self) \lor CallS(self)$
$\lor\ Accumulate(self)$
$\lor\ IncrementIndex(self)$
$\lor\ AssignAccumulatorToPoint(self)$

$GetSlices(self) \triangleq\ \land\ pc[self] = \text{“GetSlices”}$
$\land\ slices' = [index \in 1\ ..\ n[self] \mapsto \text{IF}\ (index * k + k) \geq Len(bits)\ \text{THEN}$
$SubSeq(bits,\ index * k,\ Len(bits))$
$\text{ELSE}\ \ SubSeq(bits,\ index * k,\ index * k + k - 1)]$
$\land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“PadLastSlice”}]$
$\land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,\ auxiliar\_bytes,$
$bits,\ stack,\ domain,\ message,\ n\_,$
$accumulator,\ i,\ n,\ separator,\ message\_bytes\rangle$

$PadLastSlice(self) \triangleq\ \land\ pc[self] = \text{“PadLastSlice”}$
$\land\ slices' = [slices\ \text{EXCEPT}\ ![Len(slices)] = [index \in 1\ ..\ k \mapsto$
$slices[Len(slices)][index]$
$\text{ELSE}\ \ 0]]$
$\land\ pc' = [pc\ \text{EXCEPT}\ ![self] = Head(stack[self]).pc]$
$\land\ n' = [n\ \text{EXCEPT}\ ![self] = Head(stack[self]).n]$
$\land\ stack' = [stack\ \text{EXCEPT}\ ![self] = Tail(stack[self])]$
$\land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,\ auxiliar\_bytes,$
$bits,\ domain,\ message,\ n\_,\ accumulator,$
$i,\ separator,\ message\_bytes\rangle$

$pad(self) \triangleq\ GetSlices(self) \lor PadLastSlice(self)$

$Q(self) \triangleq\ \land\ pc[self] = \text{“Q”}$
$\land\ \land\ message\_bytes' = [message\_bytes\ \text{EXCEPT}\ ![self] = bytes]$
$\land\ separator' = [separator\ \text{EXCEPT}\ ![self] = SinsemillaQ]$
$\land\ stack' = [stack\ \text{EXCEPT}\ ![self] = \langle[procedure \mapsto\ \text{“hash\_to\_pallas”},$
$pc \mapsto\ Head(stack[self]).pc,$
$separator \mapsto\ separator[self],$
$message\_bytes \mapsto\ message\_bytes[self]]\rangle$
$\circ\ Tail(stack[self])]$
$\land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“HashToPallas”}]$
$\land\ \text{UNCHANGED}\ \langle point,\ characters,\ bytes,\ auxiliar\_bytes,\ bits,$
$slices,\ domain,\ message,\ n\_,\ accumulator,\ i,\ n\rangle$

$q(self) \triangleq\ Q(self)$

$CallI2LEOSP(self) \triangleq\ \land\ pc[self] = \text{“CallI2LEOSP”}$

$$
\begin{array}{l}
\quad \land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto \text{“IntToLEOSP32”},\ pc \mapsto \text{“S”}] \rangle \circ stack[self]] \\
\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“IntToLEOSP”}] \\
\quad \land \text{UNCHANGED } \langle point,\ characters,\ bytes,\ auxiliar\_bytes,\ bits,\ slices,\ domain,\ message,\ n\_,\ accumulator,\ i,\ n,\ separator,\ message\_bytes \rangle
\end{array}
$$

$$
\begin{array}{l}
S(self) \triangleq \land\ pc[self] = \text{“S”} \\
\quad \land\ \land\ message\_bytes' = [message\_bytes \text{ EXCEPT } ![self] = bytes] \\
\quad\quad \land\ separator' = [separator \text{ EXCEPT } ![self] = SinsemillaS] \\
\quad\quad \land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto \text{“hash\_to\_pallas”},\ pc \mapsto Head(stack[self]).pc,\ separator \mapsto separator[self],\ message\_bytes \mapsto message\_bytes[self]] \rangle \circ Tail(stack[self])] \\
\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“HashToPallas”}] \\
\quad \land \text{UNCHANGED } \langle point,\ characters,\ bytes,\ auxiliar\_bytes,\ bits,\ slices,\ domain,\ message,\ n\_,\ accumulator,\ i,\ n \rangle
\end{array}
$$

$$
s(self) \triangleq CallI2LEOSP(self) \lor S(self)
$$

$$
\begin{array}{l}
HashToPallas(self) \triangleq \land\ pc[self] = \text{“HashToPallas”} \\
\quad \land\ point' = [\ a \mapsto \text{CHOOSE } r \in RandomSubset(1, 1\,..\,3) : \text{TRUE},\ b \mapsto \text{CHOOSE } r \in RandomSubset(1, 1\,..\,3) : \text{TRUE}\ ] \\
\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc] \\
\quad \land\ separator' = [separator \text{ EXCEPT } ![self] = Head(stack[self]).separator] \\
\quad \land\ message\_bytes' = [message\_bytes \text{ EXCEPT } ![self] = Head(stack[self]).message\_b \\
\quad \land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])] \\
\quad \land \text{UNCHANGED } \langle characters,\ bytes,\ auxiliar\_bytes,\ bits,\ slices,\ domain,\ message,\ n\_,\ accumulator,\ i,\ n \rangle
\end{array}
$$

$$
hash\_to\_pallas(self) \triangleq HashToPallas(self)
$$

$$
\begin{array}{l}
IntToLEOSP(self) \triangleq \land\ pc[self] = \text{“IntToLEOSP”} \\
\quad \land\ bytes' = \langle\ BitSequenceToByte(SubSeq(bits,\ 1,\ 8)),\ BitSequenceToByte(\langle SubSeq(bits,\ 9,\ 10)[1],\ SubSeq(bits,\ 9,\ 10)[2],\ 0,\ 0,\ 0,\ 0,\ 0,\ 0 \rangle),\ 0,\ 0\ \rangle \\
\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]
\end{array}
$$

$$\begin{array}{l}
\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])] \\
\land \text{ UNCHANGED } \langle point,\ characters,\ auxiliar_bytes,\ bits, \\
\qquad\qquad slices,\ domain,\ message,\ n_,\ accumulator, \\
\qquad\qquad i,\ n,\ separator,\ message_bytes\rangle
\end{array}$$

$IntToLEOSP32(self) \triangleq IntToLEOSP(self)$

$$\begin{array}{lll}
SinSemillaHashCall & \triangleq & \land\ pc[\text{“MAIN”}] = \text{“SinSemillaHashCall”} \\
& & \land\ \land\ domain' = [domain \text{ EXCEPT } ![\text{“MAIN”}] = \langle \text{“t”},\ \text{“e”},\ \text{“s”},\ \text{“t”},\ \text{“ ”},\ \text{“S”},\ \text{“i”}, \\
& & \quad \land\ message' = [message \text{ EXCEPT } ![\text{“MAIN”}] = \langle \text{“m”},\ \text{“e”},\ \text{“s”},\ \text{“s”},\ \text{“a”},\ \text{“g”},\ \text{“e} \\
& & \quad \land\ stack' = [stack \text{ EXCEPT } ![\text{“MAIN”}] = \langle [procedure \mapsto \text{“sinsemilla_hash”}, \\
& & \qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{“Done”}, \\
& & \qquad\qquad\qquad\qquad\qquad\qquad domain \mapsto domain[\text{“MAIN”}], \\
& & \qquad\qquad\qquad\qquad\qquad\qquad message \mapsto message[\text{“MAIN”}]]\rangle \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[\text{“MAIN”}]] \\
& & \land\ pc' = [pc \text{ EXCEPT } ![\text{“MAIN”}] = \text{“EncodeDomain”}] \\
& & \land \text{ UNCHANGED } \langle point,\ characters,\ bytes,\ auxiliar_bytes, \\
& & \qquad\qquad bits,\ slices,\ n_,\ accumulator,\ i,\ n, \\
& & \qquad\qquad separator,\ message_bytes\rangle
\end{array}$$

$main \triangleq SinSemillaHashCall$

Allow infinite stuttering to prevent deadlock on termination.

$$\begin{array}{lll}
Terminating & \triangleq & \land \forall\, self \in ProcSet : pc[self] = \text{“Done”} \\
& & \land \text{ UNCHANGED } vars
\end{array}$$

$$\begin{array}{lll}
Next & \triangleq & main \\
& & \quad \lor (\exists\, self \in ProcSet : \ \lor\ sinsemilla_hash(self) \\
& & \qquad\qquad\qquad\qquad\quad \lor\ sinsemilla_hash_to_point(self) \\
& & \qquad\qquad\qquad\qquad\quad \lor\ pad(self) \lor q(self) \quad \lor\ s(self) \\
& & \qquad\qquad\qquad\qquad\quad \lor\ hash_to_pallas(self) \lor IntToLEOSP32(self)) \\
& & \quad \lor\ Terminating
\end{array}$$

$$\begin{array}{lll}
Spec & \triangleq & \land\ Init \land \Box[Next]_{vars} \\
& & \land\ \land \mathrm{WF}_{vars}(main) \\
& & \quad \land \mathrm{WF}_{vars}(sinsemilla_hash(\text{“MAIN”})) \\
& & \quad \land \mathrm{WF}_{vars}(sinsemilla_hash_to_point(\text{“MAIN”})) \\
& & \quad \land \mathrm{WF}_{vars}(pad(\text{“MAIN”})) \\
& & \quad \land \mathrm{WF}_{vars}(q(\text{“MAIN”})) \\
& & \quad \land \mathrm{WF}_{vars}(s(\text{“MAIN”})) \\
& & \quad \land \mathrm{WF}_{vars}(hash_to_pallas(\text{“MAIN”})) \\
& & \quad \land \mathrm{WF}_{vars}(IntToLEOSP32(\text{“MAIN”}))
\end{array}$$

$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$

END TRANSLATION