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.

$slices = \langle\rangle$;

Holder for a number, in particular the number of slices.

$n = 0$;

Holder for a number used as the current slice index in the main loop.

$i = 1$;

Holder for a point used as an accumulator.

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

define

The number of bits in a chunk.

$k \triangleq 10$

The maximum number of chunks allowed.

$c \triangleq 253$

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.

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

Type invariants.

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

$TypeInvariantBits \triangleq bits \in Seq(\{0, 1\})$
$TypeInvariantSlices \triangleq slices \in Seq(Seq(\{0, 1\}))$
Check all type invariants.
$InvType \triangleq TypeInvariantPoint \land TypeInvariantCharacters \land TypeInvariantBytes$
$\land TypeInvariantBytes \land TypeInvariantBits \land TypeInvariantSlices$

Point holder will eventually end up with a point different than the starting one.
$LivenessPoint \triangleq \Diamond(point \neq [a \mapsto 0,\ b \mapsto 0])$
Accumulator accumulates.
$LivenessAccumulator \triangleq \Diamond(accumulator \neq [a \mapsto 0,\ b \mapsto 0])$
Index should always be incremented.
$LivenessIndex \triangleq \Diamond(i > 1)$
Slices should always be produced.
$LivenessSlices \triangleq \Diamond(Len(slices) > 0)$
Check all liveness properties.
$Liveness \triangleq LivenessPoint \land LivenessAccumulator \land LivenessIndex \land LivenessSlices$

Bytes should always be a sequence of integers representing bytes.
$SafetyBytesSequence \triangleq \land bytes = \langle\rangle \lor (\forall index \in 1 \,..\, Len(bytes) : bytes[index] \in 0 \,..\, 255)$
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.
$SafetySlicesSequence \triangleq$
$\land slices = \langle\rangle \lor (\forall index \in 1 \,..\, Len(slices) : slices[index] \in Seq(\{0, 1\}) \land Len(slices[index]) \leq k)$
The number of slices should be less than or equal to the maximum number of chunks allowed.
$SafetyMaxChunks \triangleq n \leq c$
Check all safety properties.
$Safety \triangleq SafetyBytesSequence \land SafetySlicesSequence \land SafetyMaxChunks$
end define ;

Convert a sequence of characters to a sequence of bytes.
macro *characters_to_bytes*()
begin
$bytes := [char \in 1 \,..\, Len(characters) \mapsto Ord(characters[char])]$ **;**
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 := ⟨*point.a*, *point.b*⟩ **;**
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*()
begin
CalculateN:
Calculate the number of slices needed to hash the message.
$n := Len(bits) \div k$ **;**
CallPad:
Use the global bits as input and get slices in slices.
call *pad*() **;**
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 ≤ 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()
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.
    $point := [$
      $a \mapsto \text{CHOOSE } r \in RandomSubset(1,\ 1\ ..\ 3) : \text{TRUE},$
      $b \mapsto \text{CHOOSE } r \in RandomSubset(1,\ 1\ ..\ 3) : \text{TRUE}$
    ] **;**
  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:
    $bytes := \langle$
      $BitSequenceToByte(SubSeq(bits,\ 1,\ 8)),$
      $BitSequenceToByte(\langle SubSeq(bits,\ 9,\ 10)[1],\ SubSeq(bits,\ 9,\ 10)[2],\ 0,\ 0,\ 0,\ 0,\ 0,\ 0\rangle),$
      $0,$
      0
    $\rangle$ **;**
  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 ($chksum(pcal) =$ "$ed72fbd9$" $\land\ chksum(tla) =$ "$882f4631$")

CONSTANT $defaultInitValue$

VARIABLES $point$, $characters$, $bytes$, $auxiliar\_bytes$, $bits$, $slices$, n, i, $accumulator$, pc, $stack$

define statement

$k \triangleq 10$

$c \triangleq 253$

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

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

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

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

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

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

$LivenessAccumulator \triangleq \Diamond(accumulator \neq [a \mapsto 0,\ b \mapsto 0])$

$LivenessIndex \triangleq \Diamond(i > 1)$

$LivenessSlices \triangleq \Diamond(Len(slices) > 0)$

$Liveness \triangleq LivenessPoint \land LivenessAccumulator \land LivenessIndex \land LivenessSlices$

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

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

$SafetyMaxChunks \triangleq n \leq c$

$Safety \triangleq SafetyBytesSequence \land SafetySlicesSequence \land SafetyMaxChunks$

VARIABLES $domain$, $message$, $separator$, $message\_bytes$

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

$ProcSet \triangleq \{$"MAIN"$\}$

$Init \triangleq$ Global variables
$\land point = [a \mapsto 0,\ b \mapsto 0]$
$\land characters = \langle\rangle$
$\land bytes = \langle\rangle$
$\land auxiliar\_bytes = \langle\rangle$
$\land bits = \langle\rangle$
$\land slices = \langle\rangle$
$\land n = 0$
$\land i = 1$
$\land accumulator = [a \mapsto 0,\ b \mapsto 0]$
Procedure $sinsemilla\_hash$
$\land domain = [self \in ProcSet \mapsto defaultInitValue]$
$\land message = [self \in ProcSet \mapsto defaultInitValue]$
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"
$\land characters' = domain[self]$
$\land bytes' = [char \in 1 \,..\, Len(characters') \mapsto Ord(characters'[char])]$
$\land auxiliar\_bytes' = bytes'$
$\land pc' = [pc$ EXCEPT $![self] =$ "EncodeMessage"$]$
$\land$ UNCHANGED $\langle point, bits, slices, n, i, accumulator,$
$\quad stack, domain, message, separator,$
$\quad message\_bytes\rangle$

$EncodeMessage(self) \triangleq$
$\land pc[self] =$ "EncodeMessage"
$\land characters' = message[self]$
$\land bytes' = [char \in 1 \,..\, Len(characters') \mapsto Ord(characters'[char])]$
$\land bits' = FlattenSeq([byte \in 1 \,..\, Len(bytes') \mapsto ByteToBitSequence(bytes'[byte]$

$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“SinsemillaHashToPoint”}]$
$\land\ \text{UNCHANGED } \langle point,\ auxiliar\_bytes,\ slices,\ n,\ i,$
$accumulator,\ stack,\ domain,\ message,$
$separator,\ message\_bytes\rangle$

$SinsemillaHashToPoint(self) \triangleq \land\ pc[self] = \text{“SinsemillaHashToPoint”}$
$\land\ bytes' = auxiliar\_bytes$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = \langle[procedure \mapsto \text{“sinsemilla\_hash\_to\_p}$
$pc \mapsto \text{“DecodeCipherText”}]\rangle$
$\circ\ stack[self]]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“CalculateN”}]$
$\land\ \text{UNCHANGED } \langle point,\ characters,$
$auxiliar\_bytes,\ bits,\ slices,\ n,$
$i,\ accumulator,\ domain,\ message,$
$separator,\ message\_bytes\rangle$

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

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

$sinsemilla\_hash(self) \triangleq EncodeDomain(self) \lor EncodeMessage(self)$
$\lor\ SinsemillaHashToPoint(self)$
$\lor\ DecodeCipherText(self) \lor Return(self)$

$CalculateN(self) \triangleq \land\ pc[self] = \text{“CalculateN”}$
$\land\ n' = (Len(bits) \div k)$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“CallPad”}]$
$\land\ \text{UNCHANGED } \langle point,\ characters,\ bytes,\ auxiliar\_bytes,$
$bits,\ slices,\ i,\ accumulator,\ stack,$
$domain,\ message,\ separator,\ message\_bytes\rangle$

$CallPad(self) \triangleq \land\ pc[self] = \text{“CallPad”}$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = \langle[procedure \mapsto \text{“pad”},$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad pc \quad \mapsto \text{"CallQ"}]\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \circ\, stack[self]] \\
\land\, pc' = [pc \text{ EXCEPT } ![self] = \text{"GetSlices"}] \\
\land\, \text{UNCHANGED } \langle point, characters, bytes, auxiliar\_bytes, \\
\qquad bits, slices, n, i, accumulator, domain, \\
\qquad message, separator, message\_bytes\rangle
\end{array}
$$

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

$$
\begin{array}{lcl}
InitializeAcc(self) & \triangleq & \land\, pc[self] = \text{"InitializeAcc"} \\
& & \land\, accumulator' = point \\
& & \land\, pc' = [pc \text{ EXCEPT } ![self] = \text{"MainLoop"}] \\
& & \land\, \text{UNCHANGED } \langle point, characters, bytes, \\
& & \qquad auxiliar\_bytes, bits, slices, n, i, \\
& & \qquad stack, domain, message, separator, \\
& & \qquad message\_bytes\rangle
\end{array}
$$

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

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

$$
\begin{array}{lcl}
Accumulate(self) & \triangleq & \land\, pc[self] = \text{"Accumulate"} \\
& & \land\, accumulator' = IncompleteAddition(IncompleteAddition(accumulator, point), accu \\
& & \land\, pc' = [pc \text{ EXCEPT } ![self] = \text{"IncrementIndex"}] \\
& & \land\, \text{UNCHANGED } \langle point, characters, bytes, auxiliar\_bytes,
\end{array}
$$

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

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

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

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

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

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

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

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

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

$$
\begin{array}{ll}
CallI2LEOSP(self) \triangleq & \land\ pc[self] = \text{“CallI2LEOSP”} \\
& \land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto \text{“IntToLEOSP32”}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{“S”}] \rangle \\
& \qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[self]] \\
& \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“IntToLEOSP”}] \\
& \land\ \text{UNCHANGED } \langle point, characters, bytes, auxiliar\_bytes, \\
& \qquad\qquad bits, slices, n, i, accumulator, domain, \\
& \qquad\qquad message, separator, message\_bytes \rangle
\end{array}
$$

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

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

$$
\begin{array}{ll}
HashToPallas(self) \triangleq & \land\ pc[self] = \text{“HashToPallas”} \\
& \land\ point' = [\\
& \qquad a \mapsto \text{CHOOSE } r \in RandomSubset(1, 1 \,..\, 3) : \text{TRUE}, \\
& \qquad b \mapsto \text{CHOOSE } r \in RandomSubset(1, 1 \,..\, 3) : \text{TRUE} \\
& \quad] \\
& \land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc] \\
& \land\ separator' = [separator \text{ EXCEPT } ![self] = Head(stack[self]).separator] \\
& \land\ message\_bytes' = [message\_bytes \text{ EXCEPT } ![self] = Head(stack[self]).message\_b
\end{array}
$$

$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land$ UNCHANGED $\langle characters, bytes, auxiliar\_bytes, bits,$
$slices, n, i, accumulator, domain,$
$message\rangle$

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

$IntToLEOSP(self) \triangleq \land\ pc[self] = \text{"IntToLEOSP"}$
$\land\ bytes' = \langle$
$BitSequenceToByte(SubSeq(bits, 1, 8)),$
$BitSequenceToByte(\langle SubSeq(bits, 9, 10)[1], SubSeq(bits, 9, 10)[2], 0, 0$
$0,$
0
$\rangle$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land$ UNCHANGED $\langle point, characters, auxiliar\_bytes, bits,$
$slices, n, i, accumulator, domain, message,$
$separator, message\_bytes\rangle$

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

$SinSemillaHashCall \triangleq \land\ pc[\text{"MAIN"}] = \text{"SinSemillaHashCall"}$
$\land\ \land\ domain' = [domain \text{ EXCEPT } ![\text{"MAIN"}] = \langle \text{"t"}, \text{"e"}, \text{"s"}, \text{"t"}, \text{" "}, \text{"S"}, \text{"i"},$
$\land\ message' = [message \text{ EXCEPT } ![\text{"MAIN"}] = \langle \text{"m"}, \text{"e"}, \text{"s"}, \text{"s"}, \text{"a"}, \text{"g"}, \text{"e}$
$\land\ stack' = [stack \text{ EXCEPT } ![\text{"MAIN"}] = \langle [procedure \mapsto \text{"sinsemilla\_hash"},$
$pc \mapsto \text{"Done"},$
$domain \mapsto domain[\text{"MAIN"}],$
$message \mapsto message[\text{"MAIN"}]]\rangle$
$\circ\ stack[\text{"MAIN"}]]$
$\land\ pc' = [pc \text{ EXCEPT } ![\text{"MAIN"}] = \text{"EncodeDomain"}]$
$\land$ UNCHANGED $\langle point, characters, bytes, auxiliar\_bytes,$
$bits, slices, n, i, accumulator,$
$separator, message\_bytes\rangle$

$main \triangleq SinSemillaHashCall$

Allow infinite stuttering to prevent deadlock on termination.
$Terminating \triangleq \land \forall\, self \in ProcSet : pc[self] = \text{"Done"}$
$\land$ UNCHANGED $vars$

$Next \triangleq main$
$\lor (\exists\, self \in ProcSet : \lor sinsemilla\_hash(self)$
$\lor sinsemilla\_hash\_to\_point(self)$
$\lor pad(self) \lor q(self) \lor s(self)$
$\lor hash\_to\_pallas(self) \lor IntToLEOSP32(self))$
$\lor Terminating$

$$
\begin{aligned}
Spec \triangleq\ & \wedge Init \wedge \Box[Next]_{vars} \\
& \wedge\ \wedge \mathrm{WF}_{vars}(main) \\
& \quad \wedge \mathrm{WF}_{vars}(sinsemilla\_hash(\text{“MAIN”})) \\
& \quad \wedge \mathrm{WF}_{vars}(sinsemilla\_hash\_to\_point(\text{“MAIN”})) \\
& \quad \wedge \mathrm{WF}_{vars}(pad(\text{“MAIN”})) \\
& \quad \wedge \mathrm{WF}_{vars}(q(\text{“MAIN”})) \\
& \quad \wedge \mathrm{WF}_{vars}(s(\text{“MAIN”})) \\
& \quad \wedge \mathrm{WF}_{vars}(hash\_to\_pallas(\text{“MAIN”})) \\
& \quad \wedge \mathrm{WF}_{vars}(IntToLEOSP32(\text{“MAIN”}))
\end{aligned}
$$

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

END TRANSLATION