MODULE *Verify*

EXTENDS *Sequences*, *Naturals*, *EncapPacket*, *Modbus*

LOCAL $HMACSIZE \triangleq 64$
LOCAL $START \triangleq$ "!"
LOCAL $MINMESSAGESIZE \triangleq 65$
LOCAL $PASSWORD \triangleq$ "lolpassword"
LOCAL $Range(T) \triangleq \{T[x] : x \in \text{DOMAIN } T\}$
$HMAC \triangleq \langle$"0", "5", "b", "7", "1", "b", "9", "3",
"0", "3", "a", "d", "6", "c", "7", "4",
"E", "6", "2", "3", "7", "a", "2", "7",
"5", "6", "c", "9", "5", "f", "8", "3",
"e", "c", "8", "8", "7", "e", "d", "9",
"c", "c", "9", "a", "2", "f", "7", "1",
"d", "6", "d", "6", "6", "d", "6", "0",
"d", "3", "9", "2", "e", "c", "4", "2"$\rangle$ not concerned with the inner workings of *SHA*2

$signedMessages \triangleq$ these are in *ASCII* but they are converted to decimal before being used below. See *StrTupleToNumTup*
$\langle$ [*text* $\mapsto \langle$"!"$\rangle \circ HMAC \circ \langle$"1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", "3", "7", "E", "\r", "\n"$\rangle$
id $\mapsto \langle$"u", "t", "i", "1"$\rangle$],
[*text* $\mapsto \langle$":"$\rangle \circ HMAC \circ \langle$"1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", "3", "7", "E", "\r", "\n"$\rangle$
id $\mapsto \langle$"u", "t", "i", "1"$\rangle$],
[*text* $\mapsto \langle$":"$\rangle \circ HMAC \circ \langle$"1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", ":", "1", "1", "0", "3", "0
id $\mapsto \langle$"u", "t", "i", "3"$\rangle$],
[*text* $\mapsto \langle$":"$\rangle \circ HMAC \circ \langle$"1", "1", "0", "3", "0", "0", "6", "B", "0", "0", "0", ":", "1", "1", "0", "3", "0
id $\mapsto \langle$"u", "t", "i", "3"$\rangle$]
$\rangle$

$genHMAC(str, pass) \triangleq \langle$"l", "o", "l", "h", "m", "a", "c"$\rangle$ not concerned with the inner workings of *SHA*2

$SendMessage(str) \triangleq$ TRUE sending message to another cell. Assuming this works

--fair algorithm *Verify*

variables $msg = \langle\rangle$,
$retreivedHMAC = \langle\rangle$,
$incomingMessages = signedMessages$,
$generatedHMAC = \langle\rangle$,
$CompareHMAC \in$ BOOLEAN , since we dont model *SHA*2 this is random
$trustnet\_out = \langle\rangle$,
$hmacsMatch =$ FALSE
begin
*to*1: **while** $Len(incomingMessages) > 0$ **do**
if $Len(incomingMessages) = 1$ **then**
$msg := incomingMessages[1]$;
$incomingMessages := \langle\rangle$;

```
        else
            msg := Head(incomingMessages) ;
            incomingMessages := Tail(incomingMessages) ;
        end if ;

         receive("verify", msg);
        verify2: retreivedHMAC  := GetHMAC(msg.text) ;
        verify3: generatedHMAC := genHMAC(msg.text, PASSWORD) ;
        verify4: hmacsMatch := CompareHMAC ;
        if hmacsMatch then
             verify5: send("trustnet_out", [id ↦ msg.id, isValid ↦ TRUE, source ↦ "verify", text ↦ msg.text]);
            trustnet_out := Append(trustnet_out, [id ↦ msg.id, isValid ↦ TRUE, source ↦ "verify", text ↦ ms
        else
             verify6: send("trustnet_out", [id ↦ msg.id, isValid ↦ FALSE, source ↦ "verify", text ↦ msg.text]);
            trustnet_out := Append(trustnet_out, [id ↦ msg.id, isValid ↦ FALSE, source ↦ "verify", text ↦ m
        end if ;
    end while ;

end algorithm
```

```
 BEGIN TRANSLATION
VARIABLES msg, retreivedHMAC, incomingMessages, generatedHMAC, CompareHMAC,
          trustnet_out, hmacsMatch, pc

vars ≜ ⟨msg, retreivedHMAC, incomingMessages, generatedHMAC, CompareHMAC,
        trustnet_out, hmacsMatch, pc⟩

Init ≜  Global variables
        ∧ msg = ⟨⟩
        ∧ retreivedHMAC = ⟨⟩
        ∧ incomingMessages = signedMessages
        ∧ generatedHMAC = ⟨⟩
        ∧ CompareHMAC ∈ BOOLEAN
        ∧ trustnet_out = ⟨⟩
        ∧ hmacsMatch = FALSE
        ∧ pc = "to1"

to1 ≜  ∧ pc = "to1"
       ∧ IF Len(incomingMessages) > 0
            THEN  ∧ IF Len(incomingMessages) = 1
                       THEN  ∧ msg′ = incomingMessages[1]
                             ∧ incomingMessages′ = ⟨⟩
                       ELSE  ∧ msg′ = Head(incomingMessages)
                             ∧ incomingMessages′ = Tail(incomingMessages)
                  ∧ pc′ = "verify2"
            ELSE  ∧ pc′ = "Done"
                  ∧ UNCHANGED ⟨msg, incomingMessages⟩
```

$\land$ UNCHANGED $\langle retreivedHMAC,\ generatedHMAC,\ CompareHMAC,\ trustnet\_out,$
$hmacsMatch\rangle$

$verify2 \triangleq \land\ pc = \text{“verify2”}$
$\land\ retreivedHMAC' = GetHMAC(msg.text)$
$\land\ pc' = \text{“verify3”}$
$\land$ UNCHANGED $\langle msg,\ incomingMessages,\ generatedHMAC,\ CompareHMAC,$
$trustnet\_out,\ hmacsMatch\rangle$

$verify3 \triangleq \land\ pc = \text{“verify3”}$
$\land\ generatedHMAC' = genHMAC(msg.text,\ PASSWORD)$
$\land\ pc' = \text{“verify4”}$
$\land$ UNCHANGED $\langle msg,\ retreivedHMAC,\ incomingMessages,\ CompareHMAC,$
$trustnet\_out,\ hmacsMatch\rangle$

$verify4 \triangleq \land\ pc = \text{“verify4”}$
$\land\ hmacsMatch' = CompareHMAC$
$\land$ IF $hmacsMatch'$
THEN $\land\ trustnet\_out' = Append(trustnet\_out,\ [id \mapsto msg.id,\ isValid \mapsto \text{TRUE},\ source \mapsto \text{“ve}$
ELSE $\land\ trustnet\_out' = Append(trustnet\_out,\ [id \mapsto msg.id,\ isValid \mapsto \text{FALSE},\ source \mapsto \text{“ve}$
$\land\ pc' = \text{“to1”}$
$\land$ UNCHANGED $\langle msg,\ retreivedHMAC,\ incomingMessages,\ generatedHMAC,$
$CompareHMAC\rangle$

$Next \triangleq\ to1 \lor verify2 \lor verify3 \lor verify4$
$\lor$ Disjunct to prevent deadlock on termination
$(pc = \text{“Done”} \land \text{UNCHANGED } vars)$

$Spec \triangleq\ \land\ Init \land \Box[Next]_{vars}$
$\land\ \text{WF}_{vars}(Next)$

$Termination \triangleq\ \Diamond(pc = \text{“Done”})$

END TRANSLATION

$SAFETYCHECK \triangleq$
The password is never changed
$\land\ PASSWORD = \text{“lolpassword”}$
$SAFE1 \triangleq\ PASSWORD = \text{“lolpassword”}$
if we send something then it was valid SSW
$\land$ flag $= \text{TRUE} \Rightarrow IsSSW(signedMessage)$
$SAFE2 \triangleq\ \forall\, x \in Range(trustnet\_out) : \exists\, y \in Range(signedMessages) : y.text = x.text$
buffers don't overflow
$SAFE3 \triangleq\ \land$ IF $msg \neq \langle\rangle$
THEN $Len(msg.text) \leq MAXMODBUSSIZE$
ELSE TRUE
$\land\, \forall\, x \in Range(trustnet\_out) : Len(x.text) \leq MAXMODBUSSIZE$

$\land \lor Len(retreivedHMAC) = 64$
$\lor Len(retreivedHMAC) = 0$

$LIVELINESS \triangleq$

if we get a message then something is eventually sent sent

$LIVE1 \triangleq \Diamond\Box(Len(incomingMessages) = 0)$

$\land Len(bareMessage) \geq MINMESSAGESIZE \leadsto$ result $=$ TRUE

$LIVE2 \triangleq \Diamond\Box(Len(trustnet\_out) = Len(signedMessages))$

if we get a message it is eventually processed

$LIVE3 \triangleq \forall\, x \in Range(signedMessages) : \Diamond(\exists\, y \in Range(trustnet\_out) : y.text = x.text \land y.id = x.id)$

$\land Len(bareMessage) \geq MINMESSAGESIZE \leadsto IsSSW(macMessage)$

* Modification History
* Last modified *Mon Jul* 08 18:01:10 *EDT* 2019 by *mehdi*
* Last modified *Tue* May 08 03:43:33 *EDT* 2018 by *SabraouM*
* Created Sun May 06 09:03:31 *EDT* 2018 by *SabraouM*