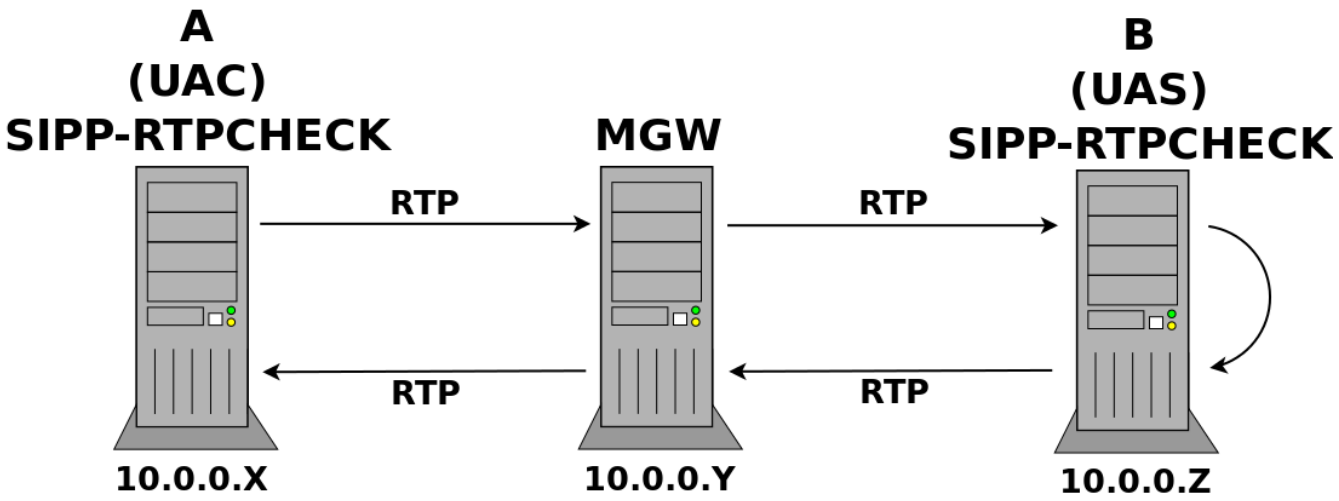
SIPP RTPCHECK functionality XML scenario syntax reference (OpenSource version)

DOCUMENT VERSION: 1.0

This page details the XML syntax used by the SIPP tool for bidirectional RTP or bidirectional SRTP checking...

BIDIRECTIONAL RTP CHECKING XML SYNTAX

Versions >= **3.7.0** of **SIPP** support bidirectional RTP checking with a setup similar to the following:



Typically...

- Endpoint A is either a TRUNK (FIXED or SRV) or a SET – simulated by SIPP running on a VM.
- Endpoint B is a PBX – simulated by SIPP running on a VM.

...but the roles of Endpoint A and B can also be reversed.

However – the endpoint acting as UAC is ALWAYS the one sending/receiving the packets used for the test while the endpoint acting as UAS is ALWAYS the one echoing back the packets.

Endpoint A sends RTP packets which go through MGW who then forwards them to Endpoint B.

Endpoint B then echos the RTP packets back to MGW who then forwards them back to Endpoint A.

The payload of the RTP packets is a simple bit pattern that can have one of six possible hexadecimal values (used by the tool to check that what is received is actually what was sent) – there are six possible values since various permutations of SIP signalling callflows may involve up to six different RTP streams:

1. 0xAA
2. 0xBB
3. 0xCC
4. 0xDD
5. 0xEE
6. 0xFF

In normal circumstances when two-way RTP media is forwarded back and forth by MGW as expected the exact same packets sent by Endpoint A would be received back by Endpoint A – test success.

Any discrepencies between packets are logged as errors by Endpoint A – too many lost/incorrect packets result in failure.

Each of the two SIPP instances indicated in the diagram above requires an XML scenario file. The syntax used for the general SIP/SDP signalling is the same as the regular SIPP syntax. However for anything specific to RTP checking the following additional constructs are supported:

XML SYNTAX (UAC SCENARIO):

A. Here is the XML syntax to generate outgoing RTP packets with bit patterns – the tool will automatically expect incoming RTP packets with identical bit patterns to come back to proceed with comparison:

```
<nop>
  <action>
    <exec rtp_stream="MEDIA_PATTERN_TYPE,MEDIA_PATTERN_ID,MEDIA_PATTERN_PAYLOAD_ID,
MEDIA_PATTERN_PAYLOAD_NAME_RATE"
  </action>
</nop>
```

Where:

MEDIA_PATTERN_TYPE indicates what media type the pattern is for:

- apattern : AUDIO
- vpattern : VIDEO

MEDIA_PATTERN_ID indicates which bit pattern to use for the media type:

- 1 : 0xAA
- 2 : 0xBB
- 3 : 0xCC
- 4 : 0xDD
- 5 : 0xEE
- 6 : 0xFF

MEDIA_PATTERN_PAYLOAD_ID indicates the static (0-95) / dynamic (96-127) payload id to use for the media type:

- 0 : G.711 u-law (PCMU/8000)
- 8 : G.711 a-law (PCMA/8000)
- 9 : G.722/8000
- 18 : G.729/8000
- 96-127 : H264/90000

NOTE: H264/90000 is the ONLY supported VIDEO media type – which also happens to be the ONLY one that uses a dynamic payload id

MEDIA_PATTERN_PAYLOAD_NAME_RATE indicates the encoding name as well as clock rate of the payload:

- PCMU/8000
- PCMA/8000
- G722/8000
- G729/8000
- H264/90000

Here's an example of an AUDIO stream which uses bit pattern #1 with static payload id 0 (described as "PCMU/8000" in the SDP):

```
<nop>
  <action>
    <exec rtp_stream="apattern,1,0,PCMU/8000" />
  </action>
</nop>
```

Here's an example of a VIDEO stream which uses bit pattern #1 with dynamic payload id 99 (described as "H264/90000" in the SDP):

```
<nop>
  <action>
    <exec rtp_stream="vpattern,1,99,H264/90000" />
  </action>
</nop>
```

B. Here is the XML syntax to pause an RTP stream (useful to simulate a "PUT-ON-HOLD" operation):

```
<nop>
  <action>
    <exec rtp_stream="MEDIA_PAUSE_TYPE" />
  </action>
</nop>
```

Where:

MEDIA_PAUSE_TYPE indicates what media type the pause is for:

- pauseapattern : AUDIO
- pausevpattern : VIDEO

Here's an example showing how to pause an AUDIO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="pauseapattern" />
  </action>
</nop>
```

Here's an example showing how to pause a VIDEO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="pausevpattern" />
  </action>
</nop>
```

C. Here is the XML syntax to resume an RTP stream (useful to simulate a "RETRIEVE-FROM-HOLD" operation):

```
<nop>
  <action>
    <exec rtp_stream="MEDIA_RESUME_TYPE" />
  </action>
</nop>
```

Where:

MEDIA_RESUME_TYPE indicates what media type the resume is for:

- resumeapattern : AUDIO
- resumevpattern : VIDEO

Here's an example showing how to resume an AUDIO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="resumeapattern" />
  </action>
</nop>
```

Here's an example showing how to resume a VIDEO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="resumevpattern" />
  </action>
</nop>
```

SAMPLE UAC SCENARIO:

Here's a working sample UAC scenario demonstrating RTP checking for BOTH AUDIO / VIDEO bit streams:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- UC360 INVITE/200/ACK/BYE/200 sample UAC scenario -->

<scenario name="Basic UC360 UAC">

  <send retrans="500">
    <![CDATA[

      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: 16001 <sip:16001@[remote_ip]:[remote_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 10 INVITE
      Contact: <sip:16001@[local_ip]:[local_port]>
      Content-Type: application/sdp
      Max-Forwards: 70
      User-Agent: VIRTUAL Mitel-UC-Endpoint (Mitel UC360 Collaboration Point/2.1.0.99;  08:00:0F:74:80:E1)
      Subject: Conference
      Session-Expires: 3600;refresher=uas
      Min-SE: 90
      Supported: 100rel
      Require: 100rel
      Content-Length: [len]

      v=0
      o=16001 0 0 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[media_ip_type] [media_ip]
      t=0 0
      m=audio [media_port] RTP/AVP 0 18 9 103 8 101
      a=rtcp:[media_port+1]
      a=sendrecv
      a=rtpmap:0 PCMU/8000
      a=rtpmap:18 G729/8000
      a=rtpmap:9 G722/16000
      a=fmtp:9 bitrate=64000
      a=rtpmap:103 G7221/16000
      a=fmtp:103 bitrate=32000
      a=rtpmap:8 PCMA/8000
      a=rtpmap:101 telephone-event/8000
```

```
      a=fmtp:101 0-11,16
      m=video [media_port+2] RTP/AVP 99 98 96 97
      b=TIAS:1536000
      b=AS:1597
      a=rtcp:[media_port+3]
      a=maxprate:192.0
      a=sendrecv
      a=rtpmap:99 H264/90000
      a=fmtp:99 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1
      a=rtpmap:98 H264/90000
      a=fmtp:98 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
      a=rtpmap:96 H264/90000
      a=fmtp:96 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
      a=rtpmap:97 H264/90000
      a=fmtp:97 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1

    ]]>
  </send>

  <recv response="100"
        optional="true">
  </recv>

  <recv response="180">
    <action>
      <!-- NOTE:  Save the 180 Ringing's CSeq header for later reuse -->
      <ereg regexp=".*" search_in="hdr" header="CSeq:" check_it="true" assign_to="1" />
      <!-- NOTE:  Save the 180 Ringing's RSeq header for later reuse -->
      <ereg regexp=".*" search_in="hdr" header="RSeq:" check_it="true" assign_to="2" />
    </action>
  </recv>

  <!-- Send PRACK for 180 Ringing -->
  <send retrans="500">
    <![CDATA[

      PRACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: 16001 <sip:16001@[local_ip]:[local_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 11 PRACK
      RAck: [$2][$1]
      Contact: <sip:16001@[local_ip]:[local_port]>
      Max-Forwards: 70
      Subject: Conference
      Content-Length: 0

    ]]>
  </send>

  <!-- receive 200 OK / PRACK (180 Ringing) -->
  <recv response="200">
  </recv>

  <!-- receive 200 OK / INVITE -->
  <recv response="200">
  </recv>

  <!-- NOTE:  [branch-5] is used to specify reuse of same [branch] value that was used five messages earlier (e.
g. INVITE) -->
  <send>
    <![CDATA[

      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch-5]
      From: "16001" <sip:16001@[remote_ip]:[remote_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 10 ACK
      Content-Length: 0
```

```
  ]]>
</send>

<nop>
    <action>
        <exec rtp_stream="apattern,1,0,PCMU/8000" />
        <exec rtp_stream="vpattern,1,99,H264/90000" />
    </action>
</nop>

<pause milliseconds="2000" />

<nop>
    <action>
        <exec rtp_stream="pauseapattern"/>
        <exec rtp_stream="pausevpattern"/>
    </action>
</nop>

<pause milliseconds="2000"/>

<nop>
    <action>
        <exec rtp_stream="resumeapattern"/>
        <exec rtp_stream="resumevpattern"/>
    </action>
</nop>

<nop>
    <action>
        <exec rtp_stream="apattern,2,0,PCMU/8000" />
        <exec rtp_stream="vpattern,2,99,H264/90000" />
    </action>
</nop>

<pause milliseconds="2000"/>

<nop>
    <action>
        <exec rtp_stream="apattern,3,0,PCMU/8000" />
        <exec rtp_stream="vpattern,3,99,H264/90000" />
    </action>
</nop>

<pause milliseconds="2000"/>

<nop>
    <action>
        <exec rtp_stream="apattern,4,0,PCMU/8000" />
        <exec rtp_stream="vpattern,4,99,H264/90000" />
    </action>
</nop>

<pause milliseconds="2000"/>

<nop>
    <action>
        <exec rtp_stream="apattern,5,0,PCMU/8000" />
        <exec rtp_stream="vpattern,5,99,H264/90000" />
    </action>
</nop>

<pause milliseconds="2000"/>

<nop>
    <action>
        <exec rtp_stream="apattern,6,0,PCMU/8000" />
        <exec rtp_stream="vpattern,6,99,H264/90000" />
    </action>
</nop>
```

```
  <pause milliseconds="2000"/>

  <send retrans="500">
    <![CDATA[

      BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch-1]
      From: "16001" <sip:16001@[remote_ip]:[remote_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 12 BYE
      Contact: <sip:16001@[local_ip]:[local_port]>
      Max-Forwards: 70
      Subject: Conference
      User-Agent: VIRTUAL Mitel-UC-Endpoint (Mitel UC360 Collaboration Point/2.1.0.99;  08:00:0F:74:80:E1)
      Content-Length: 0

    ]]>
  </send>

  <recv response="200">
  </recv>

  <!-- definition of the response time repartition table (unit is ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms)     -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>
```

COMMANDLINE (UAC SCENARIO):

The following command line can be used to launch the UAC scenario (assuming your scenario is contained in uac_scenario.xml):

```
./sipp 10.0.0.Y:5060 -sf uac_scenario.xml -i 10.0.0.X -t u1 -p 5060 -mp 4000 -m 1 -s 16002 -rtpcheck_debug
```

Where:

The "-rtpcheck_debug" parameter is used to write statistics files about the RTP streams in the current directory:

- "debugafile" logs statistics about the last AUDIO RTP stream check
- "debugvfile" logs statistics about the last VIDEO RTP stream check

The pass/fail criteria for RTP streaming checking is pretty broad – as long as at least ONE RTP packet sent is received back by the UAC testing is considered a SUCCESS – otherwise if NO sent RTP packet is received back by the UAC testing is considered a FAILURE.

Upon SUCCESS SIPP returns the "0" status code.

Upon FAILURE of RTP checking SIPP returns the "253" status code.

XML SYNTAX (UAS SCENARIO):

When doing RTP echo there is NO special syntax required – the stock SIPP functionality already implements RTP echo which can be triggered right from the command line.

SAMPLE UAS SCENARIO:

Here's a working sample UAS scenario demonstrating RTP checking for BOTH AUDIO / VIDEO bit streams:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<scenario name="Basic MCD UAS">
  <recv request="INVITE" crlf="true">
    <action>
      <!-- Save the INVITE's CSeq header for later reuse -->
      <ereg regexp=".*" search_in="hdr" header="CSeq:" check_it="true" assign_to="1" />
      <!-- Save the INVITE's Via header for later reuse -->
      <ereg regexp=".*" search_in="hdr" header="Via:" check_it="true" assign_to="2" />
    </action>
  </recv>

  <!-- NOTE:  The INVITE's Via/CSeq headers are used explicitly here -->
  <send>
    <![CDATA[

      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:];tag=[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Require: 100rel
      RSeq: 1
      Server: VIRTUAL Mitel-3300-ICP 12.0.1.99
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>

  <recv request="PRACK">
  </recv>

  <send>
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Server: VIRTUAL Mitel-3300-ICP-12.0.1.99
      Content-Length: 0

    ]]>
  </send>

  <!-- NOTE:  The INVITE's Via/CSeq headers are used explicitly here -->
  <send retrans="500">
    <![CDATA[

      SIP/2.0 200 OK
      Via: [$2]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
```

```
      CSeq: [$1]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Server: VIRTUAL Mitel-3300-ICP 12.0.1.99
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=16002 0 0 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[media_ip_type] [media_ip]
      t=0 0
      m=audio [media_port] RTP/AVP 0 18 9 103 8 101
      a=rtcp:[media_port+1]
      a=sendrecv
      a=rtpmap:0 PCMU/8000
      a=rtpmap:18 G729/8000
      a=rtpmap:9 G722/16000
      a=fmtp:9 bitrate=64000
      a=rtpmap:103 G7221/16000
      a=fmtp:103 bitrate=32000
      a=rtpmap:8 PCMA/8000
      a=rtpmap:101 telephone-event/8000
      a=fmtp:101 0-11,16
      m=video [media_port+2] RTP/AVP 99 98 96 97
      b=TIAS:1536000
      b=AS:1597
      a=rtcp:[media_port+3]
      a=maxprate:192.0
      a=rtpmap:99 H264/90000
      a=fmtp:99 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1
      a=rtpmap:98 H264/90000
      a=fmtp:98 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
      a=rtpmap:96 H264/90000
      a=fmtp:96 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
      a=rtpmap:97 H264/90000
      a=fmtp:97 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1

    ]]>
  </send>

  <recv request="ACK">
  </recv>

  <recv request="BYE">
  </recv>

  <send>
    <![CDATA[
      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Server: VIRTUAL Mitel-3300-ICP 12.0.1.99
      Content-Length: 0
    ]]>
  </send>

  <!-- definition of the response time repartition table (unit is ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms)     -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>
```

NOTE the use of "[media_port]" – which is replaced by the value passed on the command line with the "-mp <value>" parameter; "[media_port+1]", "[media_port+2]", "[media_port+3]" are replaced by the mp value plus one, mp value plus two, mp value plus three respectively.

COMMANDLINE (UAS SCENARIO):

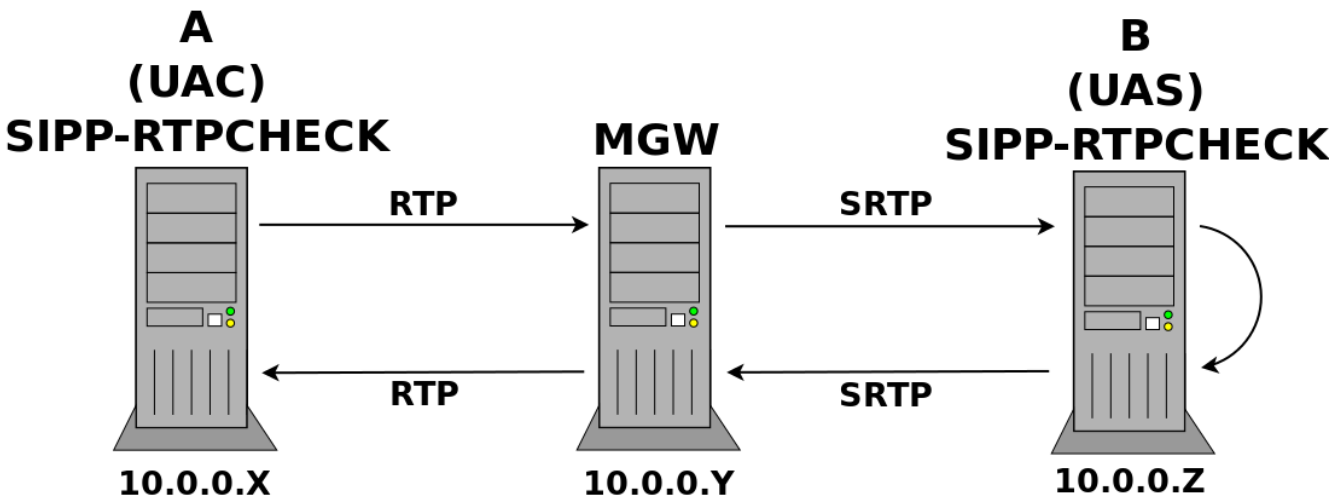
The following command line can be used to launch the UAS scenario (assuming your scenario is contained in uas_scenario.xml):

```
./sipp -sf uas_scenario.xml -i 10.0.0.Z -t u1 -p 5060 -mp 5000 -m 1 -s 16001 -rtp_echo
```

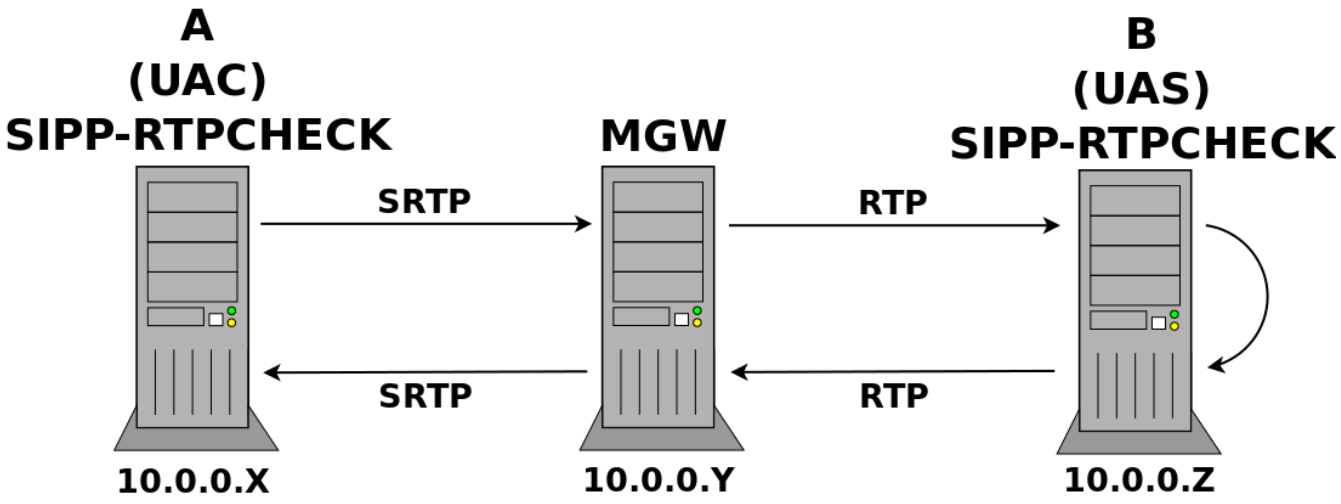
BIDIRECTIONAL SRTP CHECKING XML SYNTAX

Versions >= **3.7.0** of **SIPP** support bidirectional SRTP checking with any of the following setups:

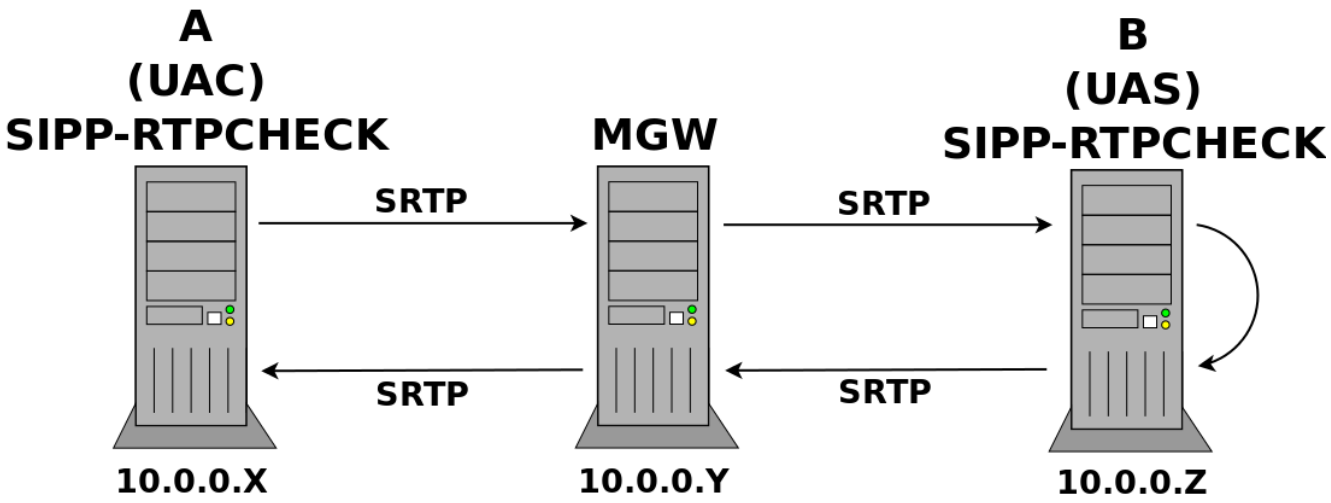
SETUP #1:



SETUP #2:



SETUP #3:



Typically...

- Endpoint A is either a TRUNK (FIXED or SRV) or a SET – simulated by SIPP running on a VM.
- Endpoint B is a PBX – simulated by SIPP running on a VM.

...but the roles of Endpoint A and B can also be reversed.

However – the endpoint acting as UAC is ALWAYS the one sending/receiving the packets used for the test while the endpoint acting as UAS is ALWAYS the one echoing back the packets.

SETUP #1:

Endpoint A sends RTP packets which go through MGW who then encrypts them and then forwards them as SRTP packets to Endpoint B.

Endpoint B decrypts the SRTP packets then re-encrypts them as SRTP packets then forwards them back to MGW who then decrypts them and finally forwards them back as RTP packets to Endpoint A.

SETUP #2:

Endpoint A encrypts SRTP packets which it sends to MGW who then decrypts them and forwards them as RTP packets to Endpoint B.

Endpoint B then echos the RTP packets back to MGW who then re-encrypts them as SRTP packets and finally forwards them as SRTP packets to Endpoint A which decrypts them.

SETUP #3:

Endpoint A encrypts SRTP packets which it sends to MGW who then decrypts them then re-encrypts them and finally forwards them as SRTP packets to Endpoint B.

Endpoint B decrypts the SRTP packets then re-encrypts them as SRTP packets then forwards them back to MGW who then decrypts them then re-encrypts them as SRTP packets and finally forwards them back as SRTP packets to Endpoint A which decrypts them.

The actual unencrypted payload of the SRTP packets is a simple bit pattern that can have one of six possible hexadecimal values (used by the tool to check that what is received is actually what was sent) – there are six possible values since various permutations of SIP signalling callflows may involve up to six different SRTP streams:

1. 0xAA
2. 0xBB
3. 0xCC
4. 0xDD
5. 0xEE
6. 0xFF

In any of these setups in normal circumstances when two-way RTP/SRTP media is forwarded back and forth by MGW as expected the exact same packets initially sent by Endpoint A (before encryption) would be received back by Endpoint A (after decryption) – test success.

Any discrepencies between packets are logged as errors by Endpoint A – too many lost/incorrect packets result in failure.

Each of the two SIPP instances indicated in the diagram above requires an XML scenario file. The syntax used for the general SIP/SDP signalling is the same as the regular SIPP syntax. However for anything specific to SRTP checking the following additional constructs are supported:

XML SYNTAX (UAC SCENARIO):

A. Here is the syntax used for a PRIMARY crypto attribute part of an AUDIO SDP OFFER m-line:

```
a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1AUTHTAGBITS1audio] inline:
[cryptokeyparams1audioREUSEPREVIOUS]
```

Where:

"[cryptotag1audio]" is replaced by the PRIMARY AUDIO cryptotag: 1

"[cryptosuiteaescm128sha1AUTHTAGBITS1audio]" is replaced by a string that depends on the AUTHTAGBITS value:

AUTHTAGBITS is one of the following:

- 80 : AES_CM_128_HMAC_SHA1_80
- 32 : AES_CM_128_HMAC_SHA1_32

"[cryptokeyparams1audioREUSEPREVIOUS]" is replaced by the auto-generated (or reused) base64-encoded concatenation of the PRIMARY AUDIO master key + salt

REUSEPREVIOUS is one of the following:

- "" : Empty string – auto-generate a new base64-encoded concatenation of PRIMARY AUDIO master key + salt
- <negative_value> – reuse previously-generated base64-encoded concatenation of PRIMARY AUDIO master key + salt

Example of a PRIMARY AUDIO crypto attribute using AES_CM_128_HMAC_SHA1_80 auto-generating a new base64-encoded concatenation of its master key + salt:

```
a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1801audio] inline:[cryptokeyparams1audio]
```

Example of a PRIMARY AUDIO crypto attribute using AES_CM_128_HMAC_SHA1_32 reusing an existing base64-encoded concatenation of its master key + salt from 4 messages earlier:

```
a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1321audio] inline:[cryptokeyparams1audio-4]
```

B. Here is the syntax used for a SECONDARY crypto attribute part of an AUDIO SDP OFFER m-line:

```
a=crypto:[cryptotag2audio] [cryptosuiteaescm128sha1AUTHTAGBITS2audio] inline:
[cryptokeyparams2audioREUSEPREVIOUS]
```

"[cryptotag2audio]" is replaced by the SECONDARY AUDIO cryptotag: 2

"[cryptosuiteaescm128sha1AUTHTAGBITS2audio]" is replaced by a string that depends on the AUTHTAGBITS value:

AUTHTAGBITS is one of the following:

- 80 : AES_CM_128_HMAC_SHA1_80
- 32 : AES_CM_128_HMAC_SHA1_32

"[cryptokeyparams2audioREUSEPREVIOUS]" is replaced by the auto-generated (or reused) base64-encoded concatenation of the SECONDARY AUDIO master key + salt

REUSEPREVIOUS is one of the following:

- "" : Empty string – auto-generate a new base64-encoded concatenation of SECONDARY AUDIO master key + salt
- <negative_value> – reuse previously-generated base64-encoded concatenation of SECONDARY AUDIO master key + salt

Example of a SECONDARY AUDIO crypto attribute using AES_CM_128_HMAC_SHA1_80 auto-generating a new base64-encoded concatenation of its master key + salt:

```
a=crypto:[cryptotag2audio] [cryptosuiteaescm128sha1802audio] inline:[cryptokeyparams2audio]
```

Example of a SECONDARY AUDIO crypto attribute using AES_CM_128_HMAC_SHA1_32 reusing an existing base64-encoded concatenation of its master key + salt from 4 messages earlier:

```
a=crypto:[cryptotag2audio] [cryptosuiteaescm128sha1322audio] inline:[cryptokeyparams2audio-4]
```

C. Here is the syntax used for a PRIMARY crypto attribute part of a VIDEO SDP OFFER m-line:

```
a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1AUTHTAGBITS1video] inline:
[cryptokeyparams1videoREUSEPREVIOUS]
```

"[cryptotag1video]" is replaced by the PRIMARY VIDEO cryptotag: 1

"[cryptosuiteaescm128sha1AUTHTAGBITS1video]" is replaced by a string that depends on the AUTHTAGBITS value:

AUTHTAGBITS is one of the following:

- 80 : AES_CM_128_HMAC_SHA1_80
- 32 : AES_CM_128_HMAC_SHA1_32

"[cryptokeyparams1videoREUSEPREVIOUS]" is replaced by the auto-generated (or reused) base64-encoded concatenation of the PRIMARY VIDEO master key + salt

REUSEPREVIOUS is one of the following:

- "" : Empty string – auto-generate a new base64-encoded concatenation of PRIMARY VIDEO master key + salt
- <negative_value> – reuse previously-generated base64-encoded concatenation of PRIMARY VIDEO master key + salt

Example of a PRIMARY VIDEO crypto attribute using AES_CM_128_HMAC_SHA1_80 auto-generating a new base64-encoded concatenation of its master key + salt:

```
a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1801video] inline:[cryptokeyparams1video]
```

Example of a PRIMARY VIDEO crypto attribute using AES_CM_128_HMAC_SHA1_32 reusing an existing base64-encoded concatenation of its master key + salt from 4 messages earlier:

```
a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1321video] inline:[cryptokeyparams1video-4]
```

D. Here is the syntax used for a SECONDARY crypto attribute part of a VIDEO SDP OFFER m-line:

```
a=crypto:[cryptotag2video] [cryptosuiteaescm128sha1AUTHTAGBITS2video] inline:
[cryptokeyparams2videoREUSEPREVIOUS]
```

"[cryptotag2video]" is replaced by the SECONDARY AUDIO cryptotag: 2

"[cryptosuiteaescm128sha1AUTHTAGBITS2video]" is replaced by a string that depends on the AUTHTAGBITS value:

AUTHTAGBITS is one of the following:

- 80 : AES_CM_128_HMAC_SHA1_80
- 32 : AES_CM_128_HMAC_SHA1_32

"[cryptokeyparams2videoREUSEPREVIOUS]" is replaced by the auto-generated (or reused) base64-encoded concatenation of the SECONDARY VIDEO master key + salt

REUSEPREVIOUS is one of the following:

- "" : Empty string – auto-generate a new base64-encoded concatenation of SECONDARY VIDEO master key + salt
- <negative_value> – reuse previously-generated base64-encoded concatenation of SECONDARY VIDEO master key + salt

Example of a SECONDARY VIDEO crypto attribute using AES_CM_128_HMAC_SHA1_80 auto-generating a new base64-encoded concatenation of its master key + salt:

```
a=crypto:[cryptotag2video] [cryptosuiteaescm128sha1802video] inline:[cryptokeyparams2video]
```

Example of a SECONDARY VIDEO crypto attribute using AES_CM_128_HMAC_SHA1_32 reusing an existing base64-encoded concatenation of its master key + salt from 4 messages earlier:

```
a=crypto:[cryptotag2video] [cryptosuiteaescm128sha1322video] inline:[cryptokeyparams2video-4]
```

E. Here is the XML syntax to generate outgoing RTP packets with bit patterns – the tool will automatically expect incoming RTP packets with identical bit patterns to come back to proceed with comparison:

```
<nop>
  <action>
    <exec rtp_stream="MEDIA_PATTERN_TYPE,MEDIA_PATTERN_ID,MEDIA_PATTERN_PAYLOAD_ID,
MEDIA_PATTERN_PAYLOAD_NAME_RATE"
  </action>
</nop>
```

Where:

MEDIA_PATTERN_TYPE indicates what media type the pattern is for:

- apattern : AUDIO
- vpattern : VIDEO

MEDIA_PATTERN_ID indicates which bit pattern to use for the media type:

- 1 : 0xAA
- 2 : 0xBB
- 3 : 0xCC
- 4 : 0xDD
- 5 : 0xEE
- 6 : 0xFF

MEDIA_PATTERN_PAYLOAD_ID indicates the static (0-95) / dynamic (96-127) payload id to use for the media type:

- 0 : G.711 u-law (PCMU/8000)
- 8 : G.711 a-law (PCMA/8000)
- 9 : G.722/8000
- 18 : G.729/8000
- 96-127 : H264/90000

NOTE: H264/90000 is the ONLY supported VIDEO media type – which also happens to be the ONLY one that uses a dynamic payload id

MEDIA_PATTERN_PAYLOAD_NAME_RATE indicates the encoding name as well as clock rate of the payload:

- PCMU/8000
- PCMA/8000
- G722/8000
- G729/8000
- H264/90000

Here's an example of an AUDIO stream which uses bit pattern #1 with static payload id 0 (described as "PCMU/8000" in the SDP):

```
<nop>
  <action>
    <exec rtp_stream="apattern,1,0,PCMU/8000" />
  </action>
</nop>
```

Here's an example of a VIDEO stream which uses bit pattern #1 with dynamic payload id 99 (described as "H264/90000" in the SDP):

```
<nop>
  <action>
    <exec rtp_stream="vpattern,1,99,H264/90000" />
  </action>
</nop>
```

F. Here is the XML syntax to pause an RTP stream (useful to simulate a "PUT-ON-HOLD" operation):

```
<nop>
  <action>
    <exec rtp_stream="MEDIA_PAUSE_TYPE" />
  </action>
</nop>
```

Where:

MEDIA_PAUSE_TYPE indicates what media type the pause is for:

- pauseapattern : AUDIO
- pausevpattern : VIDEO

Here's an example showing how to pause an AUDIO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="pauseapattern" />
  </action>
</nop>
```

Here's an example showing how to pause a VIDEO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="pausevpattern" />
  </action>
</nop>
```

G. Here is the XML syntax to resume an RTP stream (useful to simulate a "RETRIEVE-FROM-HOLD" operation):

```
<nop>
  <action>
    <exec rtp_stream="MEDIA_RESUME_TYPE" />
  </action>
</nop>
```

Where:

MEDIA_RESUME_TYPE indicates what media type the resume is for:

- resumeapattern : AUDIO
- resumevpattern : VIDEO

Here's an example showing how to resume an AUDIO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="resumeapattern" />
  </action>
</nop>
```

Here's an example showing how to resume a VIDEO RTP stream:

```
<nop>
  <action>
    <exec rtp_stream="resumevpattern" />
  </action>
</nop>
```

SAMPLE UAC SCENARIO:

Here's a working sample UAC scenario demonstrating SRTP checking for BOTH AUDIO / VIDEO bit streams (using SRTP on **BOTH** sides of MGW as described by setup #3):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- UC360 INVITE/200/ACK/BYE/200 sample UAC scenario -->

<scenario name="Basic UC360 UAC">

  <send retrans="500">
    <![CDATA[

      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: 16001 <sip:16001@[remote_ip]:[remote_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 10 INVITE
      Contact: <sip:16001@[local_ip]:[local_port]>
      Content-Type: application/sdp
      Max-Forwards: 70
      User-Agent: VIRTUAL Mitel-UC-Endpoint (Mitel UC360 Collaboration Point/2.1.0.99;  08:00:0F:74:80:E1)
      Subject: Conference
```

```
    Session-Expires: 3600;refresher=uas
    Min-SE: 90
    Supported: 100rel
    Require: 100rel
    Content-Length: [len]

    v=0
    o=16001 0 0 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[media_ip_type] [media_ip]
    t=0 0
    m=audio [rtpstream_audio_port] RTP/AVP 0 18 9 103 8 101
    a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1801audio] inline:[cryptokeyparams1audio]
    a=crypto:[cryptotag2audio] [cryptosuiteaescm128sha1322audio] inline:[cryptokeyparams2audio]
    a=rtcp:[rtpstream_audio_port+1]
    a=sendrecv
    a=rtpmap:0 PCMU/8000
    a=rtpmap:18 G729/8000
    a=rtpmap:9 G722/16000
    a=fmtp:9 bitrate=64000
    a=rtpmap:103 G7221/16000
    a=fmtp:103 bitrate=32000
    a=rtpmap:8 PCMA/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16
    m=video [rtpstream_video_port] RTP/AVP 99 98 96 97
    a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1801video] inline:[cryptokeyparams1video]
    a=crypto:[cryptotag2video] [cryptosuiteaescm128sha1322video] inline:[cryptokeyparams2video]
    b=TIAS:1536000
    b=AS:1597
    a=maxprate:192.0
    a=rtcp:[rtpstream_video_port+1]
    a=sendrecv
    a=rtpmap:99 H264/90000
    a=fmtp:99 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1
    a=rtpmap:98 H264/90000
    a=fmtp:98 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
    a=rtpmap:96 H264/90000
    a=fmtp:96 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
    a=rtpmap:97 H264/90000
    a=fmtp:97 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1

  ]]>
</send>

<recv response="100"
      optional="true">
</recv>

<recv response="180">
  <action>
    <!-- NOTE:  Save the 180 Ringing's CSeq header for later reuse -->
    <ereg regexp=".*" search_in="hdr" header="CSeq:" check_it="true" assign_to="1" />
    <!-- NOTE:  Save the 180 Ringing's RSeq header for later reuse -->
    <ereg regexp=".*" search_in="hdr" header="RSeq:" check_it="true" assign_to="2" />
  </action>
</recv>

<!-- Send PRACK for 180 Ringing -->
<send retrans="500">
  <![CDATA[

    PRACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
    From: 16001 <sip:16001@[local_ip]:[local_port]>;tag=[call_number]
    To: <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 11 PRACK
    RAck: [$2][$1]
    Contact: <sip:16001@[local_ip]:[local_port]>
    Max-Forwards: 70
```

```
      Subject: Conference
      Content-Length: 0

    ]]>
  </send>

  <!-- receive 200 OK / PRACK (180 Ringing) -->
  <recv response="200">
  </recv>

  <!-- receive 200 OK / INVITE -->
  <recv response="200">
  </recv>

  <!-- NOTE:  [branch-5] is used to specify reuse of same [branch] value that was used five messages earlier (e.
g. INVITE) -->
  <send>
    <![CDATA[

      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch-5]
      From: "16001" <sip:16001@[remote_ip]:[remote_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 10 ACK
      Content-Length: 0

    ]]>
  </send>

  <nop>
      <action>
          <exec rtp_stream="apattern,1,0,PCMU/8000" />
          <exec rtp_stream="vpattern,1,99,H264/90000" />
      </action>
  </nop>

  <pause milliseconds="2000" />

  <send retrans="500">
    <![CDATA[

      BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: "16001" <sip:16001@[remote_ip]:[remote_port]>;tag=[call_number]
      To: <sip:[service]@[remote_ip]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 12 BYE
      Contact: <sip:16001@[local_ip]:[local_port]>
      Max-Forwards: 70
      Subject: Conference
      User-Agent: VIRTUAL Mitel-UC-Endpoint (Mitel UC360 Collaboration Point/2.1.0.99;  08:00:0F:74:80:E1)
      Content-Length: 0

    ]]>
  </send>

  <recv response="200">
  </recv>

  <!-- definition of the response time repartition table (unit is ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms)     -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>
```

NOTE the use of "[rtpstream_audio_port]" – which is replaced by the value passed on the command line with the "-mp <value>" parameter; "[rtpstream_audio_port+1]" is replaced by the mp value plus one.

NOTE the use of "[rtpstream_video_port]" – which is replaced by the value passed on the command line with the "-mp <value>" parameter; "[rtpstream_video_port+1]" is replaced by the mp value plus one.

NOTE that if BOTH AUDIO and VIDEO SRTP streams are being used at once the value passed on the command line with the "-mp <value>" parameter is uniquely distributed evenly in the order of the SDP m-lines such that there is no overlap in the port numbers across the various m-lines of different media types.

COMMANDLINE (UAC SCENARIO):

The following command line can be used to launch the UAC scenario (assuming your scenario is contained in uac_scenario.xml):

```
./sipp 10.0.0.Y:5060 -sf uac_scenario.xml -i 10.0.0.X -t u1 -p 5060 -mp 4000 -m 1 -s 16002 -rtpcheck_debug -
srtpcheck_debug -audiotolerance x -videotolerance y
```

Where:

The "-rtpcheck_debug" parameter is used to write statistics files about the RTP streams in the current directory:

- "debugafile" logs statistics about the last AUDIO RTP stream check
- "debugvfile" logs statistics about the last VIDEO RTP stream check

x is a floating point number in the 0.0-1.0 range indicating the acceptable error threshold which, when reached, causes testing to be considered a FAILURE (defaults to 1.0 which means ALL AUDIO packets must be invalid/lost for testing to be considered a FAILURE)

y is a floating point number in the 0.0-1.0 range indicating the acceptable error threshold which, when reached, causes testing to be considered a FAILURE (defaults to 1.0 which means ALL VIDEO packets must be invalid/lost for testing to be considered a FAILURE)

By example – using:

```
-audiotolerance 0.4 -videotolerance 0.6
```

Would mean that any of (or both) the following conditions are true then testing would be considered a FAILURE:

- if 40% or more AUDIO SRTP packets are incorrect/lost
- if 60% or more VIDEO SRTP packets are incorrect/lost

The "-srtpcheck_debug" parameter is used to write statistics files about the SRTP streams in the current directory:

- "srtpctxdebugfile_uac" logs ALL SRTP-related information generated on the UAC side
- "debuglsrtpafile_uac" logs the LAST AUDIO SRTP session parameters sent by the LOCAL (UAC) side
- "debuglsrtpvfile_uac" logs the LAST VIDEO SRTP session parameters sent by the LOCAL (UAC) side
- "debugrsrtpafile_uac" logs the LAST AUDIO SRTP session parameters received from the REMOTE (UAS) side
- "debugrsrtpvfile_uac" logs the LAST VIDEO SRTP session parameters received from the REMOTE (UAS) side

The pass/fail criteria for SRTP streaming check is customizable for each media type using the "-audiotolerance x" as well as the "-videotolerance y" parameters.

Upon SUCCESS SIPP returns the "0" status code.

Upon FAILURE of RTP checking SIPP returns the "253" status code.

XML SYNTAX (UAS SCENARIO):

A. Here is the syntax used for a PRIMARY crypto attribute part of an AUDIO SDP ANSWER m-line:

```
a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1AUTHTAGBITS1audio] inline:
[cryptokeyparams1audioREUSEPREVIOUS]
```

"[cryptotag1audio]" is replaced by the PRIMARY AUDIO cryptotag: 1

"[cryptosuiteaescm128sha1AUTHTAGBITS1audio]" is replaced by a string that depends on the AUTHTAGBITS value:

AUTHTAGBITS is one of the following:

- 80 : AES_CM_128_HMAC_SHA1_80
- 32 : AES_CM_128_HMAC_SHA1_32

"[cryptokeyparams1audioREUSEPREVIOUS]" is replaced by the auto-generated (or reused) base64-encoded concatenation of the PRIMARY AUDIO master key + salt

REUSEPREVIOUS is one of the following:

- "" : Empty string – auto-generate a new base64-encoded concatenation of PRIMARY AUDIO master key + salt
- <negative_value> – reuse previously-generated base64-encoded concatenation of PRIMARY AUDIO master key + salt

Example of a PRIMARY AUDIO crypto attribute using AES_CM_128_HMAC_SHA1_80 auto-generating a new base64-encoded concatenation of its master key + salt:

```
a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1801audio] inline:[cryptokeyparams1audio]
```

Example of a PRIMARY AUDIO crypto attribute using AES_CM_128_HMAC_SHA1_32 reusing an existing base64-encoded concatenation of its master key + salt from 4 messages earlier:

```
a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1321audio] inline:[cryptokeyparams1audio-4]
```

B. Here is the syntax used for a PRIMARY crypto attribute part of a VIDEO SDP ANSWER m-line:

```
a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1AUTHTAGBITS1video] inline:
[cryptokeyparams1videoREUSEPREVIOUS]
```

"[cryptotag1video]" is replaced by the PRIMARY VIDEO cryptotag: 1

"[cryptosuiteaescm128sha1AUTHTAGBITS1video]" is replaced by a string that depends on the AUTHTAGBITS value:

AUTHTAGBITS is one of the following:

- 80 : AES_CM_128_HMAC_SHA1_80
- 32 : AES_CM_128_HMAC_SHA1_32

"[cryptokeyparams1videoREUSEPREVIOUS]" is replaced by the auto-generated (or reused) base64-encoded concatenation of the PRIMARY VIDEO master key + salt

REUSEPREVIOUS is one of the following:

- "" : Empty string – auto-generate a new base64-encoded concatenation of PRIMARY VIDEO master key + salt
- <negative_value> – reuse previously-generated base64-encoded concatenation of PRIMARY VIDEO master key + salt

Example of a PRIMARY VIDEO crypto attribute using AES_CM_128_HMAC_SHA1_80 auto-generating a new base64-encoded concatenation of its master key + salt:

```
a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1801video] inline:[cryptokeyparams1video]
```

Example of a PRIMARY VIDEO crypto attribute using AES_CM_128_HMAC_SHA1_32 reusing an existing base64-encoded concatenation of its master key + salt from 4 messages earlier:

```
a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1321video] inline:[cryptokeyparams1video-4]
```

C. Here is the syntax used for STARTING SRTP echo for an SRTP AUDIO stream:

```
<nop>
  <action>
    <exec rtp_echo="startaudio,MEDIA_PATTERN_PAYLOAD_ID,MEDIA_PATTERN_PAYLOAD_NAME_RATE" />
  </action>
<nop>
```

D. Here is the syntax used for UPDATING SRTP echo for an SRTP AUDIO stream:

```
<nop>
  <action>
    <exec rtp_echo="updateaudio,MEDIA_PATTERN_PAYLOAD_ID,MEDIA_PATTERN_PAYLOAD_NAME_RATE" />
  </action>
<nop>
```

E. Here is the syntax used for STOPPING SRTP echo for an SRTP AUDIO stream:

```
<nop>
  <action>
    <exec rtp_echo="stopaudio,MEDIA_PATTERN_PAYLOAD_ID,MEDIA_PATTERN_PAYLOAD_NAME_RATE" />
  </action>
<nop>
```

F. Here is the syntax used for STARTING SRTP echo for an SRTP VIDEO stream:

```
<nop>
  <action>
    <exec rtp_echo="startvideo,MEDIA_PATTERN_PAYLOAD_ID,MEDIA_PATTERN_PAYLOAD_NAME_RATE" />
  </action>
<nop>
```

G. Here is the syntax used for UPDATING SRTP echo fro an SRTP VIDEO stream:

```
<nop>
  <action>
    <exec rtp_echo="updatevideo,MEDIA_PATTERN_PAYLOAD_ID,MEDIA_PATTERN_PAYLOAD_NAME_RATE" />
  </action>
<nop>
```

H. Here is the syntax used for STOPPING SRTP echo for an SRTP VIDEO stream:

```
<nop>
  <action>
    <exec rtp_echo="stopvideo,MEDIA_PATTERN_PAYLOAD_ID,MEDIA_PATTERN_PAYLOAD_NAME_RATE" />
  </action>
<nop>
```

Where:

MEDIA_PATTERN_PAYLOAD_ID indicates the static (0-95) / dynamic (96-127) payload id to use for the media type:

- 0 : G.711 u-law (PCMU/8000)
- 8 : G.711 a-law (PCMA/8000)
- 9 : G.722/8000
- 18 : G.729/8000
- 96-127 : H264/90000

NOTE: H264/90000 is the ONLY supported VIDEO media type – which also happens to be the ONLY one that uses a dynamic payload id

MEDIA_PATTERN_PAYLOAD_NAME_RATE indicates the encoding name as well as clock rate of the payload:

- PCMU/8000
- PCMA/8000
- G722/8000
- G729/8000
- H264/90000

SAMPLE UAS SCENARIO:

Here's a working sample UAS scenario demonstrating SRTP checking for BOTH AUDIO / VIDEO bit streams (using SRTP on **BOTH** sides of MGW as described by setup #3):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<scenario name="Basic MCD UAS">
  <recv request="INVITE" crlf="true">
    <action>
      <!-- Save the INVITE's CSeq header for later reuse -->
      <ereg regexp=".*" search_in="hdr" header="CSeq:" check_it="true" assign_to="1" />
      <!-- Save the INVITE's Via header for later reuse -->
      <ereg regexp=".*" search_in="hdr" header="Via:" check_it="true" assign_to="2" />
    </action>
  </recv>

  <!-- NOTE:  The INVITE's Via/CSeq headers are used explicitly here -->
  <send>
    <![CDATA[

      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:];tag=[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Require: 100rel
      RSeq: 1
      Server: VIRTUAL Mitel-3300-ICP 12.0.1.99
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>

  <recv request="PRACK">
```

```
</recv>

<send>
  <![CDATA[

    SIP/2.0 200 OK
    [last_Via:]
    [last_From:]
    [last_To:]
    [last_Call-ID:]
    [last_CSeq:]
    Contact: <sip:[local_ip]:[local_port];transport=[transport]>
    Server: VIRTUAL Mitel-3300-ICP-12.0.1.99
    Content-Length: 0

  ]]>
</send>

<!-- NOTE:  The INVITE's Via/CSeq headers are used explicitly here -->
<send retrans="500">
  <![CDATA[

    SIP/2.0 200 OK
    Via: [$2]
    [last_From:]
    [last_To:]
    [last_Call-ID:]
    CSeq: [$1]
    Contact: <sip:[local_ip]:[local_port];transport=[transport]>
    Server: VIRTUAL Mitel-3300-ICP 12.0.1.99
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=16002 0 0 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[media_ip_type] [media_ip]
    t=0 0
    m=audio [rtpstream_audio_port] RTP/AVP 0 18 9 103 8 101
    a=crypto:[cryptotag1audio] [cryptosuiteaescm128sha1801audio] inline:[cryptokeyparams1audio]
    a=rtcp:[rtpstream_audio_port+1]
    a=sendrecv
    a=rtpmap:0 PCMU/8000
    a=rtpmap:18 G729/8000
    a=rtpmap:9 G722/16000
    a=fmtp:9 bitrate=64000
    a=rtpmap:103 G7221/16000
    a=fmtp:103 bitrate=32000
    a=rtpmap:8 PCMA/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16
    m=video [rtpstream_video_port] RTP/AVP 99 98 96 97
    a=crypto:[cryptotag1video] [cryptosuiteaescm128sha1801video] inline:[cryptokeyparams1video]
    b=TIAS:1536000
    b=AS:1597
    a=maxprate:192.0
    a=rtcp:[rtpstream_video_port+1]
    a=rtpmap:99 H264/90000
    a=fmtp:99 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1
    a=rtpmap:98 H264/90000
    a=fmtp:98 profile-level-id=64000d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
    a=rtpmap:96 H264/90000
    a=fmtp:96 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=0
    a=rtpmap:97 H264/90000
    a=fmtp:97 profile-level-id=42800d; max-mbps=108000; max-fs=3600; max-br=1280; packetization-mode=1

  ]]>
</send>

<nop>
  <action>
```

```
        <exec rtp_echo="startaudio,0,PCMU/8000" />
        <exec rtp_echo="startvideo,99,H264/90000" />
    </action>
  </nop>

  <recv request="ACK">
  </recv>

  <nop>
    <action>
        <exec rtp_echo="updateaudio,0,PCMU/8000" />
        <exec rtp_echo="updatevideo,99,H264/90000" />
    </action>
  </nop>

  <recv request="BYE">
  </recv>

  <send>
    <![CDATA[
      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Server: VIRTUAL Mitel-3300-ICP 12.0.1.99
      Content-Length: 0
    ]]>
  </send>

  <nop>
    <action>
        <exec rtp_echo="stopaudio,0,PCMU/8000" />
        <exec rtp_echo="stopvideo,99,H264/90000" />
    </action>
  </nop>



  <!-- definition of the response time repartition table (unit is ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms)     -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>
```

NOTE the use of "[rtpstream_audio_port]" – which is replaced by the value passed on the command line with the "-mp <value>" parameter; "[rtpstream_audio_port+1]" is replaced by the mp value plus one.

NOTE the use of "[rtpstream_video_port]" – which is replaced by the value passed on the command line with the "-mp <value>" parameter; "[rtpstream_video_port+1]" is replaced by the mp value plus one.

NOTE that if BOTH AUDIO and VIDEO SRTP streams are being used at once the value passed on the command line with the "-mp <value>" parameter is uniquely distributed evenly in the order of the SDP m-lines such that there is no overlap in the port numbers across the various m-lines of different media types.

COMMANDLINE (UAS SCENARIO):

The following command line can be used to launch the UAS scenario (assuming your scenario is contained in uas_scenario.xml):

```
./sipp -sf uas_scenario.xml -i 10.0.0.Z -t u1 -p 5060 -mp 5000 -m 1 -s 16001 -srtpcheck_debug
```

Where:

The "-srtpcheck_debug" parameter is used to write statistics files about the SRTP streams in the current directory:

- "srtpctxdebugfile_uas" logs ALL SRTP-related information generated on the UAS side
- "debuglsrtpafile_uas" logs the LAST AUDIO SRTP session parameters sent by the LOCAL (UAS) side
- "debuglsrtpvfile_uas" logs the LAST VIDEO SRTP session parameters sent by the LOCAL (UAS) side
- "debugrsrtpafile_uas" logs the LAST AUDIO SRTP session parameters received from the REMOTE (UAC) side
- "debugrsrtpvfile_uas" logs the LAST VIDEO SRTP session parameters received from the REMOTE (UAC) side
- "debugrefileaudio" logs ALL the AUDIO SRTP echo-related information generated on the UAS side
- "debugrefilevideo" logs ALL the VIDEO SRTP echo-related information generated on the UAS side