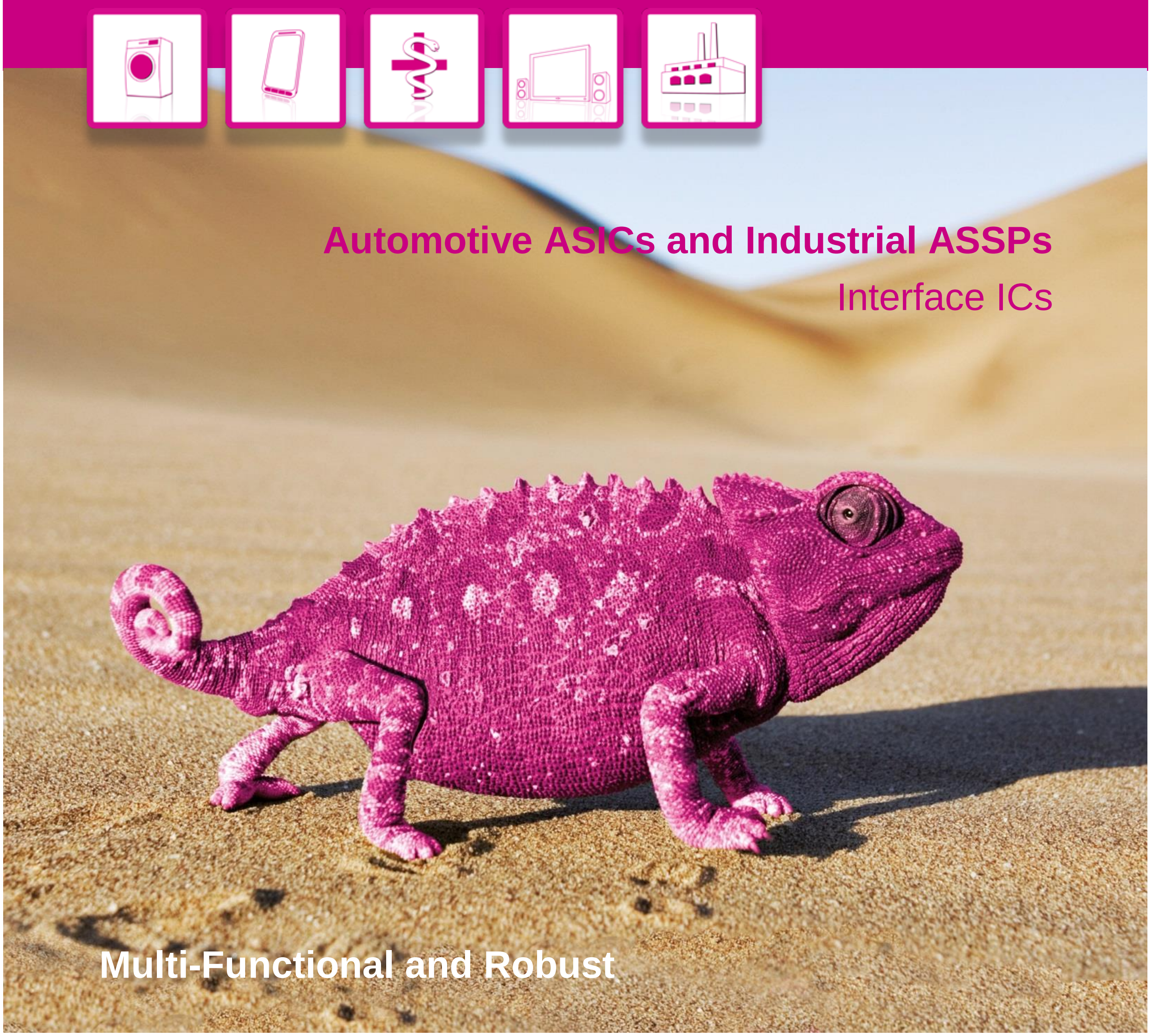
The Analog Mixed Signal Company

Application Note

Rev. 1.30 / September 2014

ZWIR45xx

Guide for Using IPsec and IKEv2 in 6LoWPANs



Contents

1 Introduction 4
2 IPsec and IKEv2 Overview 4
2.1. IPsec 4
2.1.1. IPsec Processing 5
2.2. IKEv2 7
2.2.1. Establishing a Secured Connection and Creating One IPsec SA 8
2.2.2. Creating a New IPsec SA Pair 8
2.2.3. Rekeying the IKE_SA 9
2.2.4. Rekeying an IPsec SA 9
2.2.5. Closing an SA 9
3 Setup and Configure IPsec 10
3.1. Enable IPsec Processing 10
3.2. Configure SPD and SAD Entries 10
3.2.1. Adding an SA Entry 10
3.2.2. Adding an SP Entry 12
3.2.3. Error Codes 13
3.2.4. Removing an SA Entry 13
3.2.5. Removing an SP Entry 13
3.3. IPsec Configuration Example 14
4 Setup and Configure IKEv2 16
4.1. Enable IKEv2 16
4.2. Configuring the IKEv2 Daemon 16
4.2.1. IKEv2 Configuration Example 18
5 Interfacing to Other Operating Systems 19
5.1. Linux 19
5.2. Connecting with the IPsec Linux Kernel Implementation 19
5.2.1. Example 20
5.3. Connecting with strongSwan 22
5.3.1. Example 23
5.4. Windows 25
5.4.1. Example 25
6 Security Considerations for IPsec and IKEv2 with the ZWIR45xx 26
7 Related Documents 27
8 Glossary 27
9 Document Revision History 28

List of Figures

Figure 2.1 Working Principles of IPsec 6

List of Tables

Table 2.1 Authentication Header Format 5
Table 2.2 Encapsulating Security Payload Format 5
Table 2.3 Sequence for Establishing a Secure Connection and Creating an IPsec SA 8
Table 2.4 Sequence for Creating a New IPsec SA Pair 8
Table 2.5 Sequence for Rekeying the IKD_SA 9
Table 2.6 Sequence for Rekeying the IPsec SA 9
Table 2.7 Sequence for Closing the SA 9
Table 3.1 Error Codes Generated by the IPsec Libraries 13
Table 3.2 Example Packets and Processing 15
Table 4.1 IKEv2 Default Timing Periods 17
Table 4.2 Cryptographic Algorithms for IKEv2 18

1 Introduction

6LoWPAN is a protocol to integrate wireless low-power networks into the global Internet. For security-relevant applications, protection is required to prevent unauthorized access to data packets. IPsec is a standardized security method for the internet protocol. In addition to protection of the data provided by encryption, a key exchange and rekeying method that allows identification and authentication is also necessary. Currently, the internet key exchange protocol version 2 (IKEv2) is the recommended standardized key management method for IPsec.

The 6LoWPAN software stack running on the ZWIR45xx is IPsec and IKEv2 ready. This application note shows how to configure, set up, and use the IPsec and IKEv2 functionalities.

2 IPsec and IKEv2 Overview

2.1. IPsec

Internet Protocol Security (IPsec) is a protocol suite for securing the Internet Protocol (IP) communications by authenticating and encrypting each IP packet. IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite.

IPsec is officially specified by the Internet Engineering Task Force (IETF) in various RFC publications:

- *RFC 4301: Security Architecture for the Internet Protocol*
- *RFC 4302: IP Authentication Header*
- *RFC 4303: IP Encapsulating Security Payload*
- *RFC 4308: Cryptographic Suites for IPsec*

To secure packets between two endpoints, security associations (SA) are used. An SA is a unidirectional virtual channel that describes which packets should be secured. Furthermore, the methods of authentication and encryption are defined for each SA. Each incoming and outgoing SA is identified by its security parameter index (SPI) for each endpoint.

How packets are processed by IPsec is shown in Figure 2.1.

The Security Association Database (SAD) contains all relevant information for encrypting, decrypting, and verifying the content of a packet. The entries in the Security Policy Database (SPD) define how an incoming or outgoing packet should be processed. Possible options are to discard, bypass, or secure the packet. If an outgoing packet should be secured, the SPD returns the associated SA. If no associated SA can be found in the SPD, the key exchange daemon (IKEv2) must become active to create a valid SA.

IPsec uses the authentication header (AH) to protect the integrity of a packet or the encapsulating security payload (ESP) to encrypt and authenticate the payload. Both the AH and ESP are next layer protocols and are placed between layer 3 and 4.

Table 2.1 Authentication Header Format

Byte	0	1	2	3
0	Next Header	Payload Length	Reserved	
4	SPI			
8	Sequence Number			
C	Integrity Check Vector (ICV)			
…	…			

Table 2.2 Encapsulating Security Payload Format

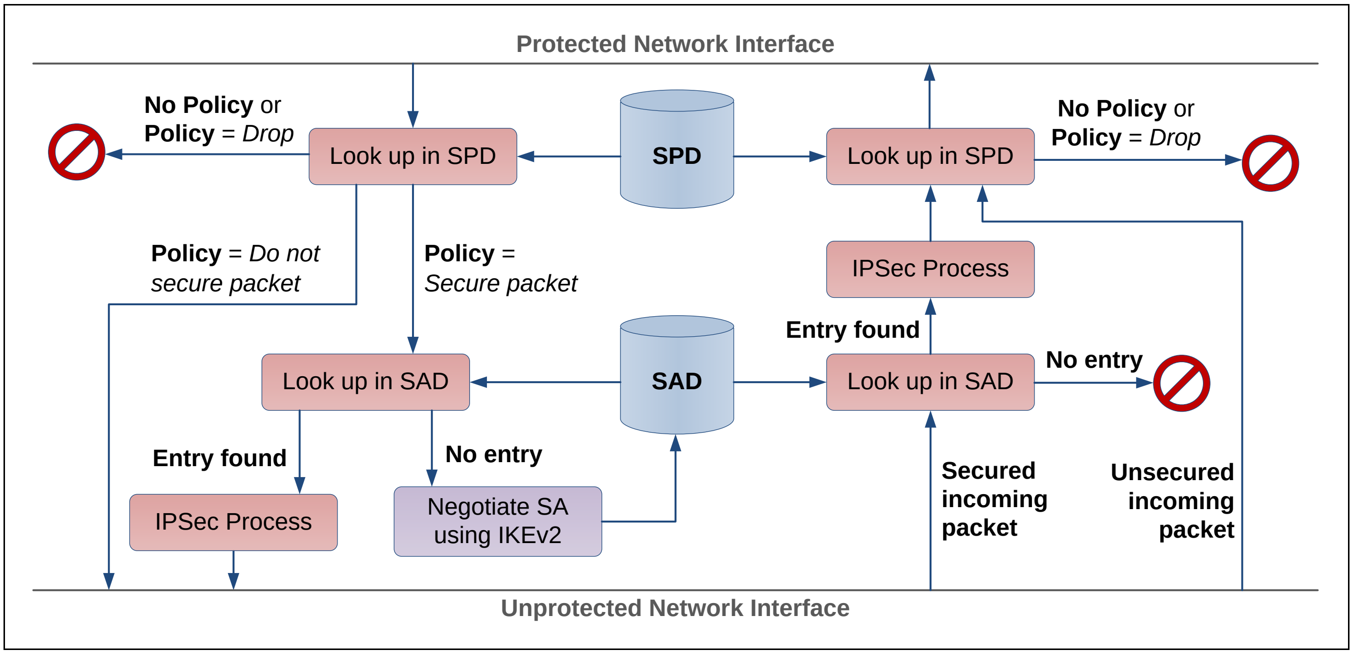
<table>
<tr><th>Byte</th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td colspan="4">SPI</td></tr>
<tr><td>4</td><td colspan="4">Sequence Number</td></tr>
<tr><td>8</td><td colspan="4" rowspan="2">Payload Data</td></tr>
<tr><td>…</td></tr>
<tr><td>…</td><td></td><td colspan="3">Padding (0-255 octets)</td></tr>
<tr><td>…</td><td colspan="2"></td><td>Pad Length</td><td>Next Header</td></tr>
<tr><td>…</td><td colspan="4" rowspan="2">Integrity Check Vector (ICV)
…</td></tr>
<tr><td>…</td></tr>
</table>

2.1.1. IPsec Processing

Incoming and outgoing packets are processed different by IPsec. If a packet arrives from an unprotected interface or lower layer, IPsec first checks the type of the packet. If it is a normal unprotected IP packet with a UDP, TCP, or ICMPv6 payload, the packet is directly passed to SP processing, which searches for a suitable entry in the SPD. If no matching address and protocol entry is found, the packet is dropped. Otherwise, the policy defines what must happen with the packet. Only if it is a bypass entry, the packet is passed to the next layer. Unsecured packets matching a security policy are dropped. If an AH or ESP secured packet arrives from the upper layer, the SPI included helps to locate the encryption and authentication suite in the SA database. Packets with an unknown SPI cannot be decrypted and must be dropped. Packets belonging to a valid SA are decrypted by IPsec and authenticated with the proper keys. Only packets with a positive integrity check are passed to SP processing.

Outgoing packets from a higher layer are handled similarly but in reverse order. First the SP processing determines if a packet must be dropped (policy missing or policy dictates dropping packet), can be passed unprotected to the upper layer, or must be secured first. The SP contains a reference to the SA of the packet. However, if the referenced SA does not exist, the key exchange daemon must create a new SA to the communication partner. The packet can now be protected with the security suite from the SA and be sent to the lower layer.

Figure 2.1 Working Principles of IPsec



2.2. IKEv2

The Internet Key Exchange version 2 (IKEv2) is a protocol used to set up a security association (SA) in the IPsec protocol suite. It is the successor of IKE and improves and simplifies establishing the connection. It uses an Elliptic-Curves-based key exchange to set up a shared session secret from which cryptographic keys are derived. Public key techniques or a pre-shared key is used to mutually authenticate the communicating parties.

IKEv2 is described in the following RFCs:

- *RFC 5996: Internet Key Exchange (IKEv2) Protocol*
- *RFC 4307: Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)*
- *RFC 5903: Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2*

IKEv2 uses UDP packets at port 500 to communicate. An IKE communication consists of packet pairs. Each request is answered by a response packet and only the initiator is allowed to retry a transfer.

There are four different IKEv2 message types:

- IKE_SA_INIT: Initiate the IKE – a key exchange is performed to communicate over a secured connection
- IKE_AUTH: Authenticate the opposite communication partner and create first IPsec SA
- CREATE_CHILD_SA: Create additional IPsec SAs or rekey the IPsec SA
- INFORMATIONAL: Contains status error and termination messages

The four message types contain different payloads:

- AUTH: Authentication data
- CERT: Certificate
- CERTREQ: Certificate request
- CP: Configuration parameter
- D: Delete
- EAP: Extensible authentication
- HDR: IKE header – first part of each IKE message
- IDi: Identification data from the initiator
- IDr: Identification data from the responder
- KEi: Key exchange payload from the initiator
- KEr: Key exchange payload from the responder
- Ni: Nonce from initiator
- Nr: Nonce from responder
- N: Notify data
- SA: New security association
- SAi1: IKE security association offer from the initiator
- SAr1: IKE security association choice from the responder
- SAi2: Child security association (for IPsec) offer from the initiator
- SAr2: Child security association (for IPsec) choice from the responder
- SK: Begin of the authenticated and encrypted payload
- TSi: Traffic selector of the initiator
- TSr: Traffic selector of the responder
- V: Vendor ID

2.2.1. Establishing a Secured Connection and Creating One IPsec SA

For the initial key exchange, two **IKE_SA_INIT** packets are necessary. The initiator sends a packet with the following payloads: HDR, SAi1, KEi, Ni. The responder answers with payloads: HDR, SAr1, KEr, Nr and an optional [CERTREQ]. After this, packets can both derive a shared secret and protect all subsequent transmitted packets. All other keys will be derived from the shared secret key. To increase the security, random nonces from both sides are used to calculate the encryption and authentication keys as well as keys for the IPsec SAs.

During the next communication phase, the nodes identify and authenticate the communication partners and negotiate one IPsec SA. Therefore only two **IKE_AUTH** packets are needed. The initiator sends HDR, SK {IDi, AUTH, SAi2, TSi, TSr, and optional [CERT], [CERTREQ], [IDr]}. After the responder has answered with the payloads HDR, SK {IDr, AUTH, SAr2, TSi, TSr, [CERT]}, the IKE connection and a pair of IPsec SAs are established.

Table 2.3 Sequence for Establishing a Secure Connection and Creating an IPsec SA

Packet	Initiator	←→	Responder
1	HDR, SAi1, KEi, Ni	→	
2		←	HDR, SAr1, [CERTREQ], KEr, Nr
3	HDR, SK {IDi, AUTH, SAi2, TSi, TSr [CERT], [CERTREQ], [IDr]}	→	
4		←	HDR, SK {IDr, AUTH, SAr2, TSi, TSr, [CERT]}

2.2.2. Creating a New IPsec SA Pair

To create a new IPsec SA, a **CREATE_CHILD_SA** request with the payload HDR, SK {SA, Ni, [KEi], TSi, TSr} is sent. It is answered by a **CREATE_CHILD_SA** message with the payload HDR, SK {SA, Nr, [KEr], TSi, TSr}.

Table 2.4 Sequence for Creating a New IPsec SA Pair

Packet	Initiator	←→	Responder
1	HDR, SK {SA, Ni, [KEi], TSi, TSr}	→	
2		←	HDR, SK {SA, Nr, [KEr], TSi, TSr}

2.2.3. Rekeying the IKE_SA

The IKE_SA is rekeyed with a pair of **CREATE_CHILD_SA** messages: HDR, SK {SA, Ni, KEi} and HDR, SK {SA, Nr, KEr}. During the rekeying process, both communication partners perform a new key exchange to refresh all keying material.

Table 2.5 Sequence for Rekeying the IKD_SA

Packet	Initiator	←→	Responder
1	HDR, SK {SA, Ni, KEi}	→	
2		←	HDR, SK {SA, Nr, KEr}

2.2.4. Rekeying an IPsec SA

For rekeying an IPsec SA, a pair of **CREATE_CHILD_SA** messages is used: HDR, SK {N(REKEY_SA), SA, Ni, [KEi], TSi, TSr} and HDR, SK {SA, Nr, [KEr], TSi, TSr}. It is not necessary to do a key exchange during the IPsec-SA rekeying process.

Table 2.6 Sequence for Rekeying the IPsec SA

Packet	Initiator	←→	Responder
1	HDR, SK {N(REKEY_SA), SA, Ni, [KEi], TSi, TSr}	→	
2		←	HDR, SK {SA, Nr, [KEr], TSi, TSr}

2.2.5. Closing an SA

Closing an SA is performed by sending an **INFORMATIONAL** message pair: HDR, SK {D} and HDR, SK {D}.

Table 2.7 Sequence for Closing the SA

Packet	Initiator	←→	Responder
1	HDR, SK {D}	→	
2		←	HDR, SK {D}

3 Setup and Configure IPsec

3.1. Enable IPsec Processing

All IPsec relevant functions are centralized in the C library file **libZWIR45xx-IPSec.a**.

To enable IPsec processing, include this library file in the project. When creating a new project, Rowley CrossStudio adds the required library if selected. For an existing CrossStudio project, choose “Add Existing Item” for the “System Files” folder and add the file **libZWIR45xx-IPSec.a**. Do not forget to set the file type to library.

If the IPsec library is included in the project, IPsec processing is always enabled. If the SPD and SAD are not configured, all inbound and outbound traffic is blocked by IPsec. Only neighbor solicitations, neighbor advertisements, and key exchange packets are processed.

3.2. Configure SPD and SAD Entries

For managing both databases, the API provides add-entries functions. It is only possible to add entries, and the priority is given by the order of adding entries. Manipulating, or order-changing functions are not provided.

Recommendation: Configure both databases after system start-up in the function **ZWIR_AppInitNetwork**.

IPsec provides end-to-end security. Therefore each connection (between sensor node and sensor node or between sensor node and server) should have its own SA and SP for each direction.

The Advanced Encryption Standard (AES) with a block size of 128 bits is used for encrypting and authentication. The AES-CTR mode is available for encryption, and the AES-XCBC-96 mode is available for data authentication.

3.2.1. Adding an SA Entry

SA entries can be added manually with the API function:

```
ZWIRSEC_SecurityAssociation_t*
  ZWIRSEC_AddSecurityAssociation ( uint32_t                        securityParamIdx,
                                   uint8_t                         replayCheck,
                                   ZWIRSEC_EncryptionSuite_t*      encSuite,
                                   ZWIRSEC_AuthenticationSuite_t*  authSuite )
```

This function adds a security association to the security association database manually. Use this function before calling **ZWIRSEC_AddSecurityPolicy** if not using IKEv2 for automatic key exchange.

The ***securityParamIdx*** argument is a unique number identifying the security association. The ***encSuite*** and ***authSuite*** parameters specify the encryption and authentication algorithms and keys. The ***replayCheck*** parameter determines if replay checks must be performed for this security association. The internal replay check counter can only be reset by recreating the specific security association. The others are pointers to the encryption ***encSuite** and authentication ***authSuite** suite containing the corresponding keys and encryption functions that are used for this security association.

The function returns a pointer to the security association descriptor if it was created successfully. In the event of an error, the function returns NULL.

The encryption parameters are set up in the following structure:

```
typedef struct {
    ZWIRSEC_EncryptionAlgorithm_t  algorithm
    uint8_t                        key   [ 16 ]
    uint8_t                        nonce [  4 ]
} ZWIRSEC_EncryptionSuite_t
```

This structure carries all encryption related information. The algorithm, a 16-byte array containing the encryption key (**key**), and a 4-byte nonce value (**nonce**) are required.

All implemented encryption algorithms are defined in the following typedef:

```
typedef enum { ... } ZWIRSEC_EncryptionAlogrithm_t
```

Enumeration of algorithms available for authentication; possible values include

ZWIRSEC_encNull = 11	no encryption
ZWIRSEC_encAESCTR = 13	AES Counter Mode based encryption
ZWIRSEC_encAESGCM16 = 20	AES Galois/Counter Modes with a 16 octet ICV Note: This algorithm requires **ZWIRSEC_authNull** as an authentication algorithm. Thus AES-GCM includes packet authentication.

This structure carries all authentication-related information:

```
typedef struct {
    ZWIRSEC_AuthenticationAlgorithm_t  algorithm
    unsigned char                      key [ 16 ]
} ZWIRSEC_AuthenticationSuite_t
```

The only two parameters are the authentication **algorithm** and the 16-byte **key**.

Possible authentication algorithms are specified in the following enum:

```
typedef enum { ... } ZWIRSEC_AuthenticationAlgorithm_t
```

Enumeration of algorithms available for authentication; possible values include:

ZWIRSEC_authNull = 0	No authentication
ZWIRSEC_authAESXCBC96 = 5	Extended AES128 CBC Mode based authentication.

3.2.2. Adding an SP Entry

SP entries are added by calling the following API function:

```
uint8_t
  ZWIRSEC_AddSecurityPolicy ( ZWIR_PolicyType_t                type,
                              ZWIR_IPv6Address_t*              remoteAddress,
                              uint8_t                          prefix,
                              ZWIR_Protocol_t                  proto,
                              uint16_t                         lowerPort,
                              uint16_t                         upperPort,
                              ZWIRSEC_SecurityAssociation_t*   securityAssociation )
```

The first parameter (**type**) of this function defines the direction and action of the policy. Possible values are

ZWIRSEC_ptOutputApply	= 0x11	Secure outbound traffic with IPSec
ZWIRSEC_ptOutputBypass	= 0x12	Bypass outbound traffic unsecured
ZWIRSEC_ptOutputDrop	= 0x13	Drop outbound traffic
ZWIRSEC_ptInputApply	= 0x21	Unsecure inbound traffic with IPSec
ZWIRSEC_ptInputBypass	= 0x22	Bypass inbound traffic unsecured
ZWIRSEC_ptInputDrop	= 0x23	Drop inbound traffic

The next parameters are a pointer to the remote IPv6 Address (***remoteAddress**) and a prefix (**prefix**) to define an address range. The prefix defines how many bits of the remote address must match, starting from the most significant address bit. The next layer protocol and a port range are configured by the parameters **proto**, **lowerPort** and **upperPort**. Possible protocols are

ZWIR_protoAny	= 0	any protocol
ZWIR_protoTCP	= 6	TCP
ZWIR_protoUDP	= 17	UDP
ZWIR_protoICMP6	= 58	ICMPv6

Last, the security association to be utilized (***securityAssociation**) must be given. For unsecured and all incoming traffic, this parameter can be null.

SP entries can be overlapping, in which case, the first matching entry will be used for securing.

3.2.3. Error Codes

The error codes listed in Table 3.1 are generated by the IPsec libraries and passed to the **ZWIR_Error** hook if it is implemented in the application code. **ZWIRSEC_eDroppedICMP** indicates that an ICMP packet was dropped due to an IPsec rule, and **ZWIRSEC_eDroppedPacket** indicates that any other (non-ICMP) packet was discarded due to an IPsec rule. **ZWIRSEC_eUnknownSPI** indicates that an IPsec packet was received but no associated security association was found. With active replay check, **ZWIRSEC_eReplayedPacket** indicates a replayed packet.

IPsec indicates authentication vector mismatches (corrupted packet) with **ZWIRSEC_eCorruptedPacket**.

Table 3.1 Error Codes Generated by the IPsec Libraries

C – Identifier	Code	Default Handling
libZWIR45xx-IPsec.a		
ZWIRSEC_eDroppedICMP	500_{HEX}	Ignore
ZWIRSEC_eDroppedPacket	501_{HEX}	Ignore
ZWIRSEC_eUnknownSPI	502_{HEX}	Ignore
ZWIRSEC_eReplayedPacket	503_{HEX}	Ignore
ZWIRSEC_eCorruptedPacket	504_{HEX}	Ignore

3.2.4. Removing an SA Entry

```
void
  ZWIRSEC_RemoveSecurityAssociation( ZWIRSEC_SecurityAssociation_t*  sa )
```

This function removes the security association pointed to by ***sa***.

3.2.5. Removing an SP Entry

```
void
  ZWIRSEC_RemoveSecurityPolicy( uint8_t  spi )
```

This function removes the security policy with index ***spi*** from the security policy database. If no index is stored at ***spi***, the function does nothing.

3.3. IPsec Configuration Example

The following example demonstrates the variety of configuration possibilities:

```
1     ZWIR_IPv6Address_t testAddress =
2      {0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x11, 0x7d, 0x00, 0x00, 0x21, 0x14, 0x98};
3     ZWIRSEC_EncryptionSuite_t es0 =
4      { ZWIRSEC_encAESCTR,
5       {0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02},
6       {0x01, 0x23, 0x45, 0x67} };
7     ZWIRSEC_AuthenticationSuite_t as0 =
8      { ZWIRSEC_authAESXCBC96,
9       {0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}
10     };
11    ZWIRSEC_EncryptionSuite_t es1 =
12     { ZWIRSEC_encAESCTR,
13      {0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04},
14      {0x89, 0xab, 0xcd, 0xef} };
15    ZWIRSEC_AuthenticationSuite_t as1 =
16     { ZWIRSEC_authAESXCBC96,
17      {0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03}
18     };
19
20    ZWIRSEC_SecurityAssociation_t *in = ZWIRSEC_AddSecurityAssociation( 0x0000affe, 0, &es0, &as0 );
21    ZWIRSEC_SecurityAssociation_t *out = ZWIRSEC_AddSecurityAssociation( 0x0000bade, 0, &es1, &as1 );
22
23    ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputApply, &testAddress, 128, ZWIR_protoUDP, 1000, 3000, out );
24    ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputApply, &testAddress, 128, ZWIR_protoUDP, 1000, 3000, in );
25
26    ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputDrop, &testAddress, 0, ZWIR_protoAny, 0, 2999, 0);
27    ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputDrop, &testAddress,0, ZWIR_protoAny, 0, 2999, 0);
28
29    ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputBypass, &testAddress, 64, ZWIR_protoUDP, 0x0, 0xffff, 0);
30    ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputBypass, &testAddress, 64, ZWIR_protoUDP, 0x0, 0xffff, 0);
```

This configuration creates a pair of SA (in-0xaffe and out-0xbade) in line 20 and 21. The in-SA uses the AES-CTR mode with the key 0x0202020202020202 and 0x01234567 as nonce. For authentication, the in-SA is using the AES-XCBC-96 with the key 0x0101010101010101.

The policies, added in line 23 and 24, specify that all UDP packets transmitted and received from the sender with IP address fe80::11:7d00:21:1498 between port 1000 and 3000 must be secured with the corresponding SA.

UDP packets from and to any other IP addresses with a port equal to or less than 2999 are dropped by IPsec (Line 26 and 27).

All other UDP packets in the same subnet (fe80::/64) are bypassed without any IPsec protection (defined in line 29 and 30).

Table 3.2 shows the corresponding rule in the previous code example and resulting IPsec action for example IP packets with different addresses, protocols, and ports.

Table 3.2 Example Packets and Processing

Direction	IP Address	Protocol/Port	Action	Line of the policy
Out	fe80::11:7d00:21:1498	UDP/1000	Secure with SA-out	23
Out	fe80::11:7d00:21:1498	UDP/1001	Secure with SA-out	23
Out	fe80::11:7d00:21:1498	UDP/999	Drop	26
Out	fe80::11:7d00:21:1498	TCP/1000	Drop	26
Out	fe80::11:7d00:21:1498	UDP/3000	Secure with SA-out	23
Out	fe80::11:7d00:21:1498	UDP/3001	Bypass	29
Out	fe80::11:7d00:21:1500	UDP/3001	Bypass	29
Out	fe80::11:7d00:21:1500	UDP/3000	Bypass	29
Out	fe80::11:7d00:21:1500	UDP/2999	Drop	26
Out	fe80::11:7d00:21:1500	UDP/1	Drop	26
Out	1234::11:7d00:21:1500	UDP/1	Drop	26
Out	1234::11:7d00:21:1500	UDP/2999	Drop	26
Out	1234::11:7d00:21:1500	UDP/3000	Drop	No matching rule
In	fe80::11:7d00:21:1498	UDP/1001	Secure with SA-in	24
In	fe80::11:7d00:21:1498	UDP/999	Drop	27
In	fe80::11:7d00:21:1498	UDP/3001	Bypass	30

4 Setup and Configure IKEv2

4.1. Enable IKEv2

The IKEv2 Daemon is sourced out in the C library file **`libZWIR45xx-IKEv2.a`**.

To use IKEv2 processing, include this library file in the project. When creating a new project, Rowley CrossStudio adds the required library if selected. For an existing CrossStudio project, choose "Add Existing Item" for the "System Files" folder and add the file **`libZWIR45xx-IKEv2.a`**. Do not forget to set the file type to library.

The IKEv2 daemon functions only if IPsec processing is enabled.

Note: As of stack version 1.10, IKEv2 has used Elliptic-Curves-based key exchange instead of Diffie-Hellman key exchange. Furthermore, AES-CTR has been used for encrypted IKE payloads. This allows a faster, more efficient and secure key exchange. The code size and the IKEv2 payload size could be minimized as well.

The former IKEv2 implementation has been moved in the library file **`libZWIR45xx-IKEv2-deprecated.a.`**

4.2. Configuring the IKEv2 Daemon

The IKEv2 daemon registers itself as the key exchange daemon for IPsec.

The IKEv2 daemon comes with its own event scheduler. Both the IKEv2 connection (IKE SA) and the derived IPsec SA (child SA) are rekeyed periodically. See Table 4.1 for the default schedule. The periods are adjustable by overwriting the weak constants. All time constants are defined in seconds:

`uint8_t  ZWIRSEC_ikeRetransmitTime = 10`

> This is a weak constant defining the how many seconds IKE waits for a reply before retransmission is initiated. The time should be large enough to enable IKE processing at the receiver. This value largely depends on the clock frequency. Set the value accordingly. The predefined value of 10 seconds is only suitable for a receiver clock frequency of 32MHz or 64MHz. The value can be redefined by definition of the variable **`ZWIRSEC_ikeRetransmitTime`** with an appropriate value in the application code.

`uint32_t  ZWIRSEC_ikeRekeyTime = 86400`

> This is a weakly defined variable that controls the interval at which the IKE connection must be rekeyed. The default setting corresponds to one week. In order to change this value, the variable **`ZWIRSEC_ikeRekeyTime`** must be defined with an appropriate value in the application code.

`uint32_t  ZWIRSEC_ikeSARekeyTime = 604800`

> This weakly defined variable controls the interval during which security associations remain valid before rekeying is required. The default setting corresponds to one day. In order to change this value, a **`ZWIRSEC_ikeSARekeyTime`** variable must be defined with an appropriate value in the application code.

Table 4.1 IKEv2 Default Timing Periods

Event	Period	Description
Retransmit	10 seconds	Time until retransmission of an IKE packet if no answer packet was received
Rekeying child SA	24 hours	Time until the child SAs will be rekeyed
Rekeying IKE SA	7 days	Time until the IKE SAs will be rekeyed

If an outgoing IP packet must be secured according to a SP and if there is no corresponding SA, IPsec calls the key exchange daemon to create a matching SA. In addition to the IPsec SPD and SAD configuration, the IKE authentication configuration is needed. Such entries are added by an APIIKE function:

```
unsigned char
  ZWIRSEC_AddIKEAuthenticationEntry (  ZWIR_IPv6Address_t*  remoteAddress,
                                       uint8_t              prefixLength,
                                       uint8_t              id,
                                       uint8_t              idLength,
                                       uint8_t              presharedKey )
```

The address range to match the IPv6 address of the communication partner is defined by the first two parameters **`*remoteAddress`** and **`prefixLength`**. The next parameter is a pointer the ID (**`*id`**) and the parameter **`idLength`** defines the length of the ID. The pre-shared key is defined by a pointer to a 16-bit wide field (**`*presharedKey`**).

During the secured connection negotiation, both endpoints identify and authenticate each other. Each endpoint sends its ID and authentication string derived from the PSK.

The first added authentication **`IkeAuthenticationEntry`** is used for generating the authentication data to send them to the opposite communication partner. For identifying the communication partner, all entries are used.

For IPsec SAs negotiated by IKEv2, the replay check is always enabled.

The key exchange is initialized automatically if an SP matching packet is sent. All sent packets will be dropped until the key exchange is finished.

To save memory, it is not possible to start to key exchanges at the same time.

By default the IKEv2 daemon uses and offers to the communication partner the following cryptographic algorithms:

Table 4.2 Cryptographic Algorithms for IKEv2

Type	Algorithm	Algorithm – Deprecated LIB
IKE encryption	**AES-CTR**	AES-CBC
IKE authentication	AES-XCBC-96	AES-XCBC-96
IKE pseudo-random function	AES-128-XCBC	AES-128-XCBC
Key exchange	**ECP 256 Bit**	Diffie-Hellman 768 Bit
IPsec encryption	AES-CTR	AES-CTR
IPsec authentication	AES-XCBC-96	AES-XCBC-96

4.2.1. IKEv2 Configuration Example

The configuration in the following example allows two nodes in the same sub-network to establish a secured UDP connection at port 1111. Both must have the same ID to identify the connection and the same PSK as the initial secret to secure the key exchange (first id is the local ID).

```
ZWIR_IPv6Address_t testAddress = { 0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                                   0x00, 0x11, 0x7d, 0x00, 0x00, 0x00, 0x00, 0x00 };

char psk[17] = {"abcdefghijklmnop"};
char id[3] = {'I','d','0'};

ZWIRSEC_AddIkeAuthenticationEntry(&testAddress, 64, id, 3, psk);

ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputApply, &testAddress, 64, ZWIR_protoUDP, 1111, 1111, NULL );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputApply,  &testAddress, 64, ZWIR_protoUDP, 1111, 1111, NULL );
```

5 Interfacing to Other Operating Systems

5.1. Linux

Both IPsec and IKEv2 with PSK are compatible with implementation for Linux.

Section 5.2 describes how to connect the ZWIR45xx with the IPsec Linux kernel implementation, and section 5.3 describes connecting with the strongSwan IKEv2 key exchange daemon.

For all of the following examples, the Linux distribution Ubuntu 9.10 is utilized.

5.2. Connecting with the IPsec Linux Kernel Implementation

IPsec-secured connections are setup in the config file:

`/etc/ipsec-tools.conf`

It is possible to start this configuration (instead of rebooting) immediately:

```
sudo /etc/init.d/setkey start
```

Further information about how to configure the IPsec connection under Ubuntu can be found on these websites:

https://help.ubuntu.com/community/IPSecHowTo

or

http://www.freebsd.org/cgi/man.cgi?query=setkey&sektion=8

5.2.1. Example

The following configuration gives an example of how an IPsec connection that secures UDP port 1000 and pinging can be arranged between a PC and the ZWIR45xx.

/etc/ipsec-tools.conf

```
# Flush the SAD and SPD
flush;
spdflush;

# Add out SA
add fe80::21d:e0ff:fe20:253 fe80::11:7d00:21:1498 esp 0x0000affe
        -E aes-ctr      0x0202020202020202020202020202020201234567
        -A aes-xcbc-mac 0x01010101010101010101010101010101;

# Add in SA
add fe80::11:7d00:21:1498 fe80::21d:e0ff:fe20:253 esp 0x0000bade
        -E aes-ctr      0x0404040404040404040404040404040489abcdef
        -A aes-xcbc-mac 0x03030303030303030303030303030303;

# Add out SP
spdadd fe80::21d:e0ff:fe20:253 fe80::11:7d00:21:1498[1000] any -P out ipsec
       esp/transport//require;

# Add in SP
spdadd fe80::11:7d00:21:1498 fe80::21d:e0ff:fe20:253 [1000] any -P in ipsec
       esp/transport//require;


# Enable IPsec for pinging

spdadd ::/0 ::/0 icmp6 128,0 -P out ipsec
       esp/transport//require;

spdadd ::/0 ::/0 icmp6 129,0 -P in ipsec
       esp/transport//require;
```

ZWIR45xx configuration

```
ZWIR_IPv6Address_t testAddress = { 0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                                   0x02, 0x1d, 0xe0, 0xff, 0xfe, 0x20, 0x02, 0x53 };
ZWIRSEC_EncryptionSuite_t es0 =
 { ZWIRSEC_encAESCTR,
  {0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02},
  {0x01, 0x23, 0x45, 0x67} };
ZWIRSEC_AuthenticationSuite_t as0 =
 { ZWIRSEC_authAESXCBC96,
  {0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}
 };
ZWIRSEC_EncryptionSuite_t es1 =
 { ZWIRSEC_encAESCTR,
  {0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04},
  {0x89, 0xab, 0xcd, 0xef}};
ZWIRSEC_AuthenticationSuite_t as1 =
 { ZWIRSEC_authAESXCBC96,
  {0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03}
 };

ZWIRSEC_SecurityAssociation_t *in = ZWIRSEC_AddSecurityAssociation( 0x0000affe, 0, &es0, &as0 );
ZWIRSEC_SecurityAssociation_t *out = ZWIRSEC_AddSecurityAssociation( 0x0000bade, 0, &es1, &as1 );

ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputApply, &testAddress, 0, ZWIR_protoUDP, 1000, 3000, out );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputApply, &testAddress, 0, ZWIR_protoUDP, 1000, 3000, in );

ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputApply, &testAddress, 0, ZWIR_protoICMPv6, 128, 129, out );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputApply, &testAddress, 0, ZWIR_protoICMPv6, 128, 129, in );
```

5.3. Connecting with strongSwan

strongSwan is a complete IPsec implementation for Linux 2.4 and 2.6 kernels, and it supports IKEv1 and IKEv2.

Since strongSwan supports IKEv2 with PSK authentication, it is possible to establish a secure connection between the ZWIR45xx and a device running Linux/strongSwan; e.g., a PC.

The only restriction is that link local addresses are currently not supported by strongSwan. Therefore a prefix must be distributed by a router advertisement daemon like radvd.

Two configuration files must be edited after installing strongSwan:

```
ipsec.conf         //contains all connection parameters

ipsec.secrets      //contains all secrets such as PSKs
```

For more information, visit the strongSwan wiki: https://wiki.strongswan.org/projects/strongswan

To reduce the algorithm negotiation overhead, all algorithms for the key exchange and IPsec should be specified in the configuration file.

5.3.1. Example

In this example, the strongSwan IKEv2 daemon should establish a secured connection to the wireless sensor node. Both must know the ID and PSK of the opposite node. In this case, both nodes have the same ID and PSK.

The key exchange is based on two computationally intensive calculations. Thus, increasing the speed of the microcontroller is recommended.

The following config files show how to establish a secure UDP connection at port 1111 between a wireless sensor node and a strongSwan daemon.

/etc/ipsec.conf

```
config setup
        charonstart=yes //start the IKEv2 Daemon
        # uncomment if required
        # charondebug="ike 4, dmn 4, mgr 4, chd 4, job 4, cfg 4, knl 4, net 4" //write debug infos

conn test
        keyexchange=ikev2       //use IKEv2
        left=2001:db8:1:0:21e:bff:fe57:656c     //local IPv6 Address
        right=2001:db8:1:0:11:7d00:21:15aa      //remote IPv6 Address
        rightid=id0     //remote ID
        leftid=id0      //local ID
        auto=start      //establish connection while startup
        auth=esp        //use ESPs for IPsec
        authby=psk      //select PSK authentication
        esp=aes128ctr-aesxcbc!  //specify security algorithms for IPsec
        ike=aes128ctr-aesxcbc-ecp256!   //specify security algorithms for IKEv2
        ikelifetime=7d  //time till IKEv2 rekeying
        keylifetime=1d  //time till IPsec SA rekeying
        leftprotoport=UDP/1111  //local upper protocol and port specification
        rightprotoport=UDP/1111 //remote upper protocol and port specification
```

/etc/ipsec.secrets

```
'id0' : PSK "abcdefghijklmnop" //PSK for specific local and remote IP
```

Example C-Code for ZWIR 45xx

```
#include <stdio.h>
#include <ZWIR45xx-6LoWPAN.h>
#include <ZWIR45xx-IKEv2.h>
#include <ZWIR45xx-IPsec.h>

void ZWIR_AppInitNetwork ( void ) {
  ZWIR_IPv6Address_t testAddress = { 0x20, 0x01, 0x0d, 0xb8, 0x00, 0x01, 0x00, 0x00,
                                     0x02, 0x1e, 0x0b, 0xff, 0xfe, 0x57, 0x65, 0x6c };
  char psk[17] = {"abcdefghijklmnop"};
  char id[3] = {'i','d','0'};

  ZWIRSEC_AddIkeAuthenticationEntry(&testAddress,64,id,3,psk);
  ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputApply, &testAddress, 64, ZWIR_protoUDP, 1111, 1111, 0 );
  ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputApply,  &testAddress, 64, ZWIR_protoUDP, 1111, 1111, 0 );
}
```

5.4. Windows

Since Windows 7®, Microsoft has introduced the Windows Filtering Platform (WFP). This platform enables IKEv2 and IPsec for Windows®. Unfortunately only a few security algorithms are supported. Thus it is not possible to use only IKEv2 with AES based algorithms.

However, the following example demonstrates IPsec-secured communication with the ZWIR4512 module.

5.4.1. Example

The following configuration gives an example of how an IPsec connection that secures UDP port 32799 can be arranged between a Windows® PC and the ZWIR45xx. The encryption and authentication algorithm is **`ZWIRSEC_encAESGCM16`.**

The example code for configuring the WFP with a small C program and the ZWIRxxxx project can be found in the **`Examples/`** folder under **`IPsec_Windows/`**

`ZWIR45xx configuration`

```
#define PORT 32799

#define WIN_IN_SPI 0x19c3ba8c // assigned by Windows automatically

ZWIR_IPv6Address_t testAddress = { 0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                                   0x02, 0x1d, 0xe0, 0xff, 0xfe, 0x20, 0x02, 0x53 };
ZWIR_IPv6Address_t testAddress  = {  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1};
ZWIRSEC_EncryptionSuite_t esIn  = { ZWIRSEC_encAESGCM16,
                                   {0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
                                    0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b},
                                   {0x07, 0x08, 0x09, 0x0a} };
ZWIRSEC_EncryptionSuite_t esOut = { ZWIRSEC_encAESGCM16,
                                   {0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
                                    0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02},
                                   {0x03, 0x04, 0x05, 0x06} };
ZWIRSEC_AuthenticationSuite_t as = { ZWIRSEC_authNull,{} };


ZWIRSEC_SecurityAssociation_t *saIn = ZWIRSEC_AddSecurityAssociation( 0x55aabbcc, 0, &esIn, &as );
ZWIRSEC_SecurityAssociation_t *saOut = ZWIRSEC_AddSecurityAssociation( WIN_IN_SPI, 0, &esOut, &as );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputApply, &addrUnsp, 0, ZWIR_protoUDP, PORT, PORT, saOut );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputApply, &addrUnsp, 0, ZWIR_protoUDP, PORT, PORT, saIn );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptOutputBypass, &addrUnsp, 0, ZWIR_protoICMPv6, 0, 0xffff, 0 );
ZWIRSEC_AddSecurityPolicy( ZWIRSEC_ptInputBypass, &addrUnsp, 0, ZWIR_protoICMPv6, 0, 0xffff, 0 );
```

6 Security Considerations for IPsec and IKEv2 with the ZWIR45xx

Only IPsec with IKEv2 is the secure method providing key freshness for secured connections. As the security algorithm, AES in Counter Mode is used. To prevent identical encrypted patterns (as a result of identical initialization vectors), it is necessary to change the key at regular intervals. However, IKEv2 was not actually designed for low-power, low-bandwidth networks such as 6LoWPANs. The initial key exchange with its long keys consumes a lot of computing power and energy.

Thus IKEv2 should be used if high security is necessary and the system can provide enough energy. Usually IPsec provides enough security with the AES-CTR encryption for 6LoWPAN at typical small data rates. It must ensure that the encrypting initialization vector will be reset to zero after a system reset.

7 Related Documents

IETF Documents	Source
Security Architecture for the Internet Protocol	*RFC 4301 - http://www.ietf.org/rfc/rfc4301*
IP Authentication Header	*RFC 4302 - http://www.ietf.org/rfc/rfc4302*
IP Encapsulating Security Payload	*RFC 4303 - http://www.ietf.org/rfc/rfc4303*
Cryptographic Algorithms for IKEv2	*RFC 4307 - http://www.ietf.org/rfc/rfc4307*
Cryptographic Suites for IPsec	*RFC 4308 - http://www.ietf.org/rfc/rfc4308*
Elliptic Curve Groups modulo a Prime for IKE and IKEv2	*RFC 5903 - http://tools.ietf.org/html/rfc5903*
Internet Key Exchange (IKEv2) Protocol	*RFC 5996 - http://www.ietf.org/rfc/rfc5996*

Note: *X.xy.pdf* refers to the current version of the ZMDI document.

ZMDI Documents	File Name
*ZWIR451x Programming Guide**	*ZWIR451x_ProgGuide_revX.xy.pdf*

Visit the ZWIR4512 product page www.zmdi.com/zwir4512 on ZMDI's website www.zmdi.com or contact your nearest sales office for the latest version of these documents.

*Note: Documents marked with an asterisk (*) require a free customer login account for access. To setup a login account, click on Login in the upper right corner of the website and follow the instructions in the resulting dialog box. After login, addition document sections are available on the product pages.

8 Glossary

Term	Description
AES	Advanced Encryption Standard
AH	Authentication Header
API	Application Programming Interface
CBC	Cyclic Block Cipher
ESP	Encapsulating Security Payload
ICMPv6	Internet Control Message Protocol Version 6
ICV	Integrity Check Vector
IKE	Internet Key Exchange
IPsec	Internet Protocol Security
ID	Identifier
PSK	Pre Shared Key
RFC	Request for Comments (type of publication of the Internet Engineering Task Force (IETF) and the Internet Society)

Term	Description
SA	Security Association
SAD	Security Association Database
SP	Security Policy
SPD	Security Policy Database
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WFP	Windows Filtering Platform
WPAN	Wireless Personal Area Network

9 Document Revision History

Revision	Date	Description
1.00	September 30, 2010	Initial release.
1.10	November 4, 2011	Clarification of prefix definition. Update contact information.
1.11	August 3, 2012	Replaced deprecated enum names. Minor edits.
1.20	April 2, 2014	Updated code; replaced deprecated enums; updated ZWIRSEC_SecurityAssociation_t; added removing functions.
1.30	September 3, 2014	Added ECC-based key exchange; added windows IPsec example. Updated contacts. Updated imagery for cover and headers. Minor edits.

Sales and Further Information www.zmdi.com wpan@zmdi.com

Zentrum Mikroelektronik Dresden AG Global Headquarters Grenzstrasse 28 01109 Dresden, Germany Central Office: Phone +49.351.8822.306 Fax +49.351.8822.337	**ZMD America, Inc.** 1525 McCarthy Blvd., #212 Milpitas, CA 95035-7453 USA USA Phone 1.855.275.9634 Phone +1.408.883.6310 Fax +1.408.883.6358	**Zentrum Mikroelektronik Dresden AG, Japan Office** 2nd Floor, Shinbashi Tokyu Bldg. 4-21-3, Shinbashi, Minato-ku Tokyo, 105-0004 Japan Phone +81.3.6895.7410 Fax +81.3.6895.7301	**ZMD FAR EAST, Ltd.** 3F, No. 51, Sec. 2, Keelung Road 11052 Taipei Taiwan Phone +886.2.2377.8189 Fax +886.2.2377.8199	**Zentrum Mikroelektronik Dresden AG, Korea Office** U-space 1 Building 11th Floor, Unit JA-1102 670 Sampyeong-dong Bundang-gu, Seongnam-si Gyeonggi-do, 463-400 Korea Phone +82.31.950.7679 Fax +82.504.841.3026

European Technical Support
Phone +49.351.8822.7.772
Fax +49.351.8822.87.772

European Sales (Stuttgart)
Phone +49.711.674517.55
Fax +49.711.674517.87955