ZMDI
The Analog Mixed Signal Company

Application Note

Rev. 1.11 / August 2012

# ZWIR45xx

## Guide for Using IPsec and IKEv2 in 6LoWPANs

# Contents



# List of Figures



# List of Tables

# 1 Introduction

6LoWPAN is a protocol to integrate wireless low power networks in to the global internet. For security relevant application there must be a possibility to protect data packets for not allowed access an eavesdropping. Therefore IPsec is a standardized security method for the internet protocol. Like the protection of the data realized by encryption, a key exchange and rekeying method and identification and authentication are necessary too. At this time the internet key exchange protocol version 2 (IKEv2) is the recommended standardized key management method for IPsec.

The 6LoWPAN software stack running at the ZWIR45xx is IPsec and IKv2 ready. This application note shows how to configure, to set up and to 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:

- 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. A 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. All incoming and outgoing SA are identified by its security parameter index (SPI) for each endpoint.

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

The SAD contains all relevant information for en- and decrypting and verifying the content of a packet. The entries in the SPD define how a incoming or outgoing packet should be processed. Possible options are to discard, to bypass or to secure a packet. If an outgoing packet should be secured, the SPD returns the according SA. I case of returning a non existing SA the key exchange daemon (IKEv2) must become active to create a valid SA.

IPsec is using 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 Len | Reserved | |
| 4 | SPI | | | |
| 8 | Sequence Number | | | |
| C | Integrity Check Vector (ICV) | | | |
| … | … | | | |

*Table 2.2 Encapsulating Security Payload Format*

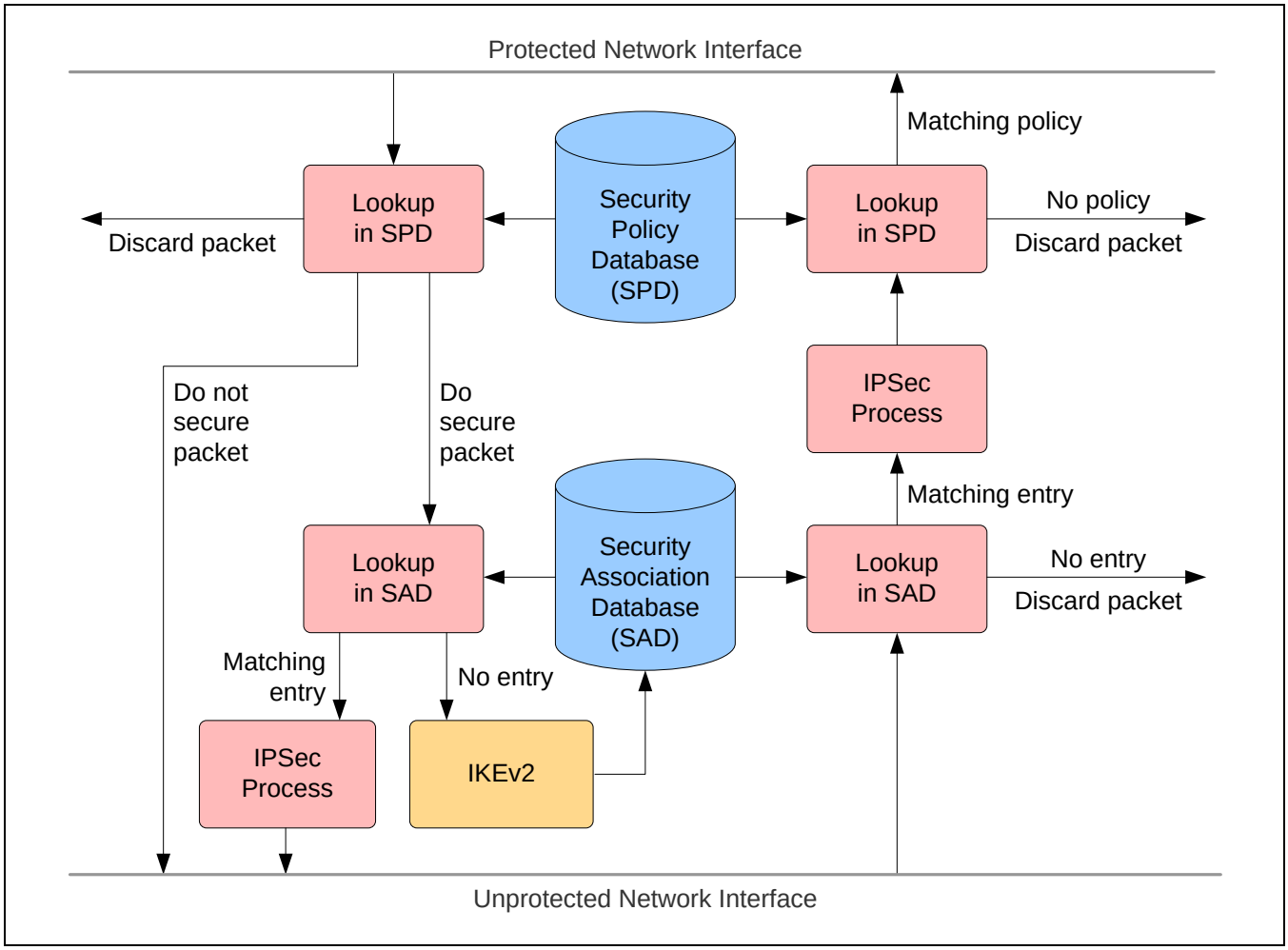
| Byte | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | SPI | | | |
| 4 | Sequence Number | | | |
| 8 | Payload Data | | | |
| … | | | | |
| … | | Padding (0-255 octets) | | |
| … | | | Pad Length | Next Header |
| … | Integrity Check Vector (ICV) | | | |
| … | … | | | |

### 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 checks firstly the type of the packet. If it's a normal unprotected IP packet with a UDP, TCP or ICMPv6 payload the packet is directly passed to the SP processing. There a suitable entry in the SPD is tried to find. In case of no address and protocol matching entry the packet is dropped. Otherwise the policy defines what has to happen with the packet. Only if it's a bypass entry the packet is passed to the next layer. Unsecured packets matching to a secure policy are dropped as well. If an AH or ESP secured packet arrives from the upper layer the included SPI helps to find the encryption and authentication suit in the SA database. Packets with an unknown SPI can't be decrypted and must be dropped. Packets belonging to a valid SA are decrypted by IPsec and authenticated with the according keys. Only packets with a positive integrity check are passed to the SP processing.

Outgoing packets from a higher layer are handled vice versa. First the SP processing finds out if a packet must be thrown away, can be passed unprotected to the upper layer or has to be secured. The SP contains a reference to the belonging SA. But if the referenced SA does not exist the key exchange daemon has to 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 IPsec Processing***



## 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 the connection establishing. It uses a Diffie–Hellman key exchange to set up a shared session secret, from which cryptographic keys are derived. Public key techniques or a pre-shared key are 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)

IKEv2 uses UDP packets at port 500 to communicate. An IKE communication contains of packet pairs. Eche 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 creates first IPsec SA
- CREATE_CHILD_SA: create additional IPsec SAs and for rekeying
- INFORMATIONAL: contains status error and termination messages

Those 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
- KE: Key exchange payload
- Ni: Noce from initiator
- Nr: Nonce from responder
- N: Notify data
- SA: Security association
- 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 are two **IKE_SA_INIT** packets necessary. The initiator 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 both can derive a shared secret and protect all followed 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 knots 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 answered with the payloads HDR, SK {IDr, AUTH, SAr2, TSi, TSr, [CERT]} the IKE connection and a pair of IPsec SAs are established.

| 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 answered by a **CREATE_CHILD_SA** message with the payload: HDR, SK {SA, Nr, [KEr], TSi, TSr}.

| 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

IKE_SA are 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 doing a new key exchange to refresh all keying material.

| 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 as well: 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.

| 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 a SA realized by sending a **INFORMATIONAL** message pair: HDR, SK {D} and HDR, SK {D}

| 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 into your project. While creating a new project CrossStudio adds the required library if selected. Otherwise choose in an existing CrossStudio project “Add Existing Item” for the “System Files” folder and add the file **`libZWIR45xx-IPSec.a`**. Don’t forget to set the file type to library.

If the IPsec library is included in the project IPsec processing is always enabled. Without configuring the SPD and SAD all traffic, in- and outbound, is blocked by IPsec. Only neighbor solicitations and advertisements are processed as well as key exchange packets.

## 3.2. Configure SPD and SAD Entries

For managing both databases the APIMD6 provides add entries functions. It is only possible to add entries and the priority is given be the order of adding entries. Removing, manipulating or order-changing functions are not provided.

It is recommended to 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. Till now for encryption the AES-CTR mode is available and for data authentication the AES-XCBC-96 mode.

### 3.2.1. Adding an SA Entry

SA entries can be added manually with the API function:

```
ZWIRSEC_SecurityAssociation_t*
  ZWIRSEC_AddSecurityAssociation ( unsigned long                    securityParamID,
                                   ZWIRSEC_EncryptionSuite_t*       encSuite,
                                   ZWIRSEC_AuthenticationSuite_t*   authSuite )
```

This function expected three parameters. The first one is the unique security parameter index (SPI) – **securityParamID** of the security association. The others are pointer to the encryption ***encSuite** and authentication ***authSuite** suit, containing the corresponding keys and encryption functions, which are used for this security association. The return value is a pointer to the created security association.

The encryption parameters are set up in the following structure:

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

This structure carries all encryption related information. Required are the algorithm, a 16 Byte long array containing the encryption key (**key**) and a 4 Byte nonce value (**nonce**).

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 base encryption

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 long **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 auth. |

### 3.2.2. Adding an SP Entry

SP entries are added by calling the following API function:

```
unsigned char
  ZWIRSEC_AddSecurityPolicy ( ZWIR_PolicyType_t              type,
                              ZWIR_IPv6Address_t*            remoteAddress,
                              unsigned char                  prefix,
                              ZWIR_Protocol_t                proto,
                              unsigned short                 lowerPort,
                              unsigend short                 upperPort,
                              ZWIRSEC_SecurityAssociation_t* securityAssociation )
```

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

| | |
|---|---|
| **ZWIRSEC_ptOutputApply** | secure outbound traffic |
| **ZWIRSEC_ptOutputBypass** | bypass outbound traffic |
| **ZWIRSEC_ptOutputDrop** | drop outbound traffic |
| **ZWIRSEC_ptInputApply** | secure incoming traffic |
| **ZWIRSEC_ptInputBypass** | bypass incoming traffic |
| **ZWIRSEC_ptInputDrop** | drop incoming 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 have to 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 |

At last the to be utilized security association (***securityAssociation**) must be given. For not secured or all incoming traffic this parameter can be null.

SP-entries can be overlapping at which the first matching entry will be used for securing.

### 3.3. IPsec Configuration Example

The following example shows the variety of configuration possibilities:

```
01 ZWIR_IPv6Address_t testAddress =
02  {0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x11, 0x7d, 0x00, 0x00, 0x21, 0x14, 0x98};
03 ZWIRSEC_EncryptionSuite_t es0 =
04  { ZWIRSEC_EncAESCTR,
05   {0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02},
06   {0x01, 0x23, 0x45, 0x67} };
07 ZWIRSEC_AuthenticationSuite_t as0 =
08  { ZWIRSEC_AuthAESXCBC96,
09   {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, &es0, &as0 );
21 ZWIRSEC_SecurityAssociation_t *out = ZWIRSEC_AddSecurityAssociation( 0x0000bade, &es1, &as1 );
22
23 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputApply, &testAddress, 128, ZWIR_UDP, 1000, 3000, out );
24 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputApply, &testAddress, 128, ZWIR_UDP, 1000, 3000, in );
25
26 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputDrop, &testAddress, 0, ZWIR_AnyProtocol, 0, 2999, 0);
27 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputDrop, &testAddress,0, ZWIR_AnyProtocol, 0, 2999, 0);
28
29 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputBypass, &testAddress, 64, ZWIR_UDP, 0x0, 0xffff, 0);
30 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputBypass, &testAddress, 64, ZWIR_UDP, 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).

Following table shows for example IP packets with different addresses, protocols and ports the IPsec action and corresponding rule.

***Table 3.1 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 | 15 |
| out | fe80::11:7d00:21:1498 | UDP/1001 | secure with SA-out | 15 |
| out | fe80::11:7d00:21:1498 | UDP/999 | drop | 18 |
| out | fe80::11:7d00:21:1498 | TCP/1000 | drop | 18 |
| out | fe80::11:7d00:21:1498 | UDP/3000 | secure with SA-out | 15 |
| out | fe80::11:7d00:21:1498 | UDP/3001 | bypass | 22 |
| out | fe80::11:7d00:21:1500 | UDP/3001 | bypass | 22 |
| out | fe80::11:7d00:21:1500 | UDP/3000 | bypass | 22 |
| out | fe80::11:7d00:21:1500 | UDP/2999 | drop | 18 |
| out | fe80::11:7d00:21:1500 | UDP/1 | drop | 18 |
| out | 1234::11:7d00:21:1500 | UDP/1 | drop | 18 |
| out | 1234::11:7d00:21:1500 | UDP/2999 | drop | 18 |
| out | 1234::11:7d00:21:1500 | UDP/3000 | drop | No matching rule |
| in | fe80::11:7d00:21:1498 | UDP/1001 | secure with SA-in | 16 |
| in | fe80::11:7d00:21:1498 | UDP/999 | drop | 19 |
| in | fe80::11:7d00:21:1498 | UDP/3001 | bypass | 23 |

# 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 into your project While creating a new project CrossStudio adds the required library if selected. Otherwise choose in an existing CrossStudio project “Add Existing Item” for the “System Files” folder and add the file **`libZWIR45xx-IKEv2.a`**. Don’t forget to set the file type to library.

The IkEv2 Daemon is only working if IPsec processing is enabled.

## 4.2. Configuring the IKEv2 Daemon

The IKEv2 daemon registers itself as 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. The periods are adjustable by overwriting the weak constants. All time constants are defined in seconds:

**`uint8_t  ZWIR_ikeRetransmitTime = 10`**

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. Thus, set the value accordingly. The predefined value of 10 seconds is suitable for a receiver clock frequency of 32 MHz or 64 MHz only. The value can be redefined by definition of the variable **`ZWIR_ikeRetransmitTime`** with an appropriate value in application code.

**`uint32_t  ZWIR_ikeRekeyTime = 86400`**

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

**`uint32_t  ZWIR_ikeSARekeyTime = 604800`**

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

***Table 4.1 IKEv2 Default Timing Periods***

| Event | Period | Description |
|---|---|---|
| `Retransmit` | `10 seconds` | Time till retransmitting an IKE packet if no answer packet was received |
| `Rekeying child SA` | `24 hours` | Time till the child SAs will be rekeyed |
| `Rekeying IKE SA` | `7 days` | Time till the IKE SAs will be rekeyed |

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

```
unsigned char
  ZWIR_AddIKEAuthenticationEntry (  ZWIR_IPv6Address_t*  remoteAddress,
                                    unsigned char        prefix,
                                    unsigned char*       id,
                                    unsigned char        idLength,
                                    unsigned char*       presharedKey )
```

The address range to match the IPv6 address of the communication partner is defined by the first two parameter **`*remoteAddress`** and **`prefix`**. 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 width field (**`*presharedKey`**).

During the secured connection negotiation both endpoints are identifying and authenticating 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.

The key exchange is initialized automatically if a SP matching packet is sent. All sent packets will be dropped until key exchange is not 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 |
|---|---|
| IKE encryption | AES-CBC |
| IKE authentication | AES-XCBC-96 |
| IKE pseudo random function | AES-128-XCBC |
| Key exchange | Diffie Hellman 768 Bit |
| IPsec encryption | AES-CTR |
| IPsec authentication | AES-XCBC-96 |

### 4.2.1. IKEv2 Configuration Example

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

```
01 ZWIR_IPv6Address_t testAddress = { 0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02                                    0x00, 0x11, 0x7d, 0x00, 0x00, 0x00, 0x00, 0x00 };
03
04 char psk[17] = {"abcdefghijklmnop"};
05 char id[3] = {'I','d','0'};
06
07 ZWIRSEC_AddIkeAuthenticationEntry(&testAddress,64,id,3,psk);
08
09 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputApply, &testAddress, 64, ZWIR_UDP, 1111, 1111, NULL );
10 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputApply,  &testAddress, 64, ZWIR_UDP, 1111, 1111, NULL );
```

# 5 Interfacing to Other Operating Systems

## 5.1. Linux

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

How to connect the ZWIR 45xx whit the IPsec Linux kernel implementation or with the StrongSwan IKEv2 key exchange daemon are described below.

For all 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 IPsec connection under Ubuntu can be found under:

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

### 5.2.1. Example

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

/etc/ipsec-tools.conf

```
01 # Flush the SAD and SPD
02 flush;
03 spdflush;
04
05 # Add out SA
06 add fe80::21d:e0ff:fe20:253 fe80::11:7d00:21:1498 esp 0x0000affe
07         -E aes-ctr      0x02020202020202020202020202020201234567
08         -A aes-xcbc-mac 0x01010101010101010101010101010101;
09
10 # Add in SA
11 add fe80::11:7d00:21:1498 fe80::21d:e0ff:fe20:253 esp 0x0000bade
12         -E aes-ctr      0x04040404040404040404040404040489abcdef
13         -A aes-xcbc-mac 0x03030303030303030303030303030303;
14
15 # Add out SP
16 spdadd fe80::21d:e0ff:fe20:253 fe80::11:7d00:21:1498[1000] any -P out ipsec
17        esp/transport//require;
18
19 # Add in SP
20 spdadd fe80::11:7d00:21:1498 fe80::21d:e0ff:fe20:253 [1000] any -P in ipsec
21        esp/transport//require;
```

```
// ZWIR45xx configuration

01 ZWIR_IPv6Address_t testAddress = { 0xfe, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02                                    0x02, 0x1d, 0xe0, 0xff, 0xfe, 0x20, 0x02, 0x53 };
03 ZWIRSEC_EncryptionSuite_t es0 =
04  { ZWIRSEC_EncAESCTR,
05   {0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02},
06   {0x01, 0x23, 0x45, 0x67} };
07 ZWIRSEC_AuthenticationSuite_t as0 =
08  { ZWIRSEC_AuthAESXCBC96,
09   {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, &es0, &as0 );
21 ZWIRSEC_SecurityAssociation_t *out = ZWIRSEC_AddSecurityAssociation( 0x0000bade, &es1, &as1 );
22
23 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputApply, &testAddress, 0, ZWIR_UDP, 1000, 3000, out );
24 ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputApply, &testAddress, 0, ZWIR_UDP, 1000, 3000, in );
```

## 5.3. Connecting with StrongSwan

StrongSwan is a complete IPsec implementation for Linux 2.4 and 2.6 kernels and supports IKEv1 and IKEv2.

Since StrongSwan supports IKEv2 with PSK authentication it is possible to establish a secure connection between the ZWIR45xx and a Linux/Strongswan running device, like a PC.

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

After installing StrongSwan two config files must be edit:

```
ipsec.conf          //contains all connection parameters
ipsec.secrets       //contains all secrets like PSKs
```

For more information, visit the StrongSwan wiki: http://wiki.strongswan.org/projects/show/strongswan

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

### 5.3.1. Example

In this example the StrongSwan IKEv2 Daemon should establish a secured connection to the wireless sensor node. Both need to know the ID and PSK of the opposite. In this case both have the same ID and PSK.

The key exchange is based on two computationally intensive calculations. Thus it is recommended to increase the speed of the microcontroller.

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

01 config setup
02          charonstart=yes //start the IKEv2 Daemon
03          charondebug="ike 4, dmn 4, mgr 4, chd 4, job 4, cfg 4, knl 4, net 4" //write debug infos
04
05 conn test
06          keyexchange=ikev2                       //use IKEv2
07          left=2001:db8:1:0:21e:bff:fe57:656c     //local IPv6 Address
08          right=2001:db8:1:0:11:7d00:21:15aa      //remote IPv6 Address
09          rightid=id0                             //remote ID
10          auto=start                              //establish connection while startup
11          auth=esp                                //use ESPs for IPsec
12          authby=psk                              //select PSK authentication
13          esp=aes128ctr-aesxcbc!                  //specify security algorithms for IPsec
14          ike=aes128-aesxcbc-modp768!             //specify security algorithms for IKEv2
15          ikelifetime=7d                          //time till IKEv2 rekeying
16          keylifetime=1d                          //time till IPsec SA rekeying
17.         leftprotoport=UDP/1111                  //local upper protocol and port specification
18.         rightprotoport=UDP/1111                 //remote upper protocol and port specification
```

```
/etc/ipsec.secrets

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

```
// Example C-Code for ZWIR 45xx

01 #include <stdio.h>
02 #include <ZWIR45xx-6LoWPAN.h>
03 #include <ZWIR45xx-IKEv2.h>
04 #include <ZWIR45xx-IPsec.h>
05
06 void ZWIR_AppInitNetwork ( void ) {
07   ZWIR_IPv6Address_t testAddress = { 0x20, 0x01, 0x0d, 0xb8, 0x00, 0x01, 0x00, 0x00,
08                                      0x02, 0x1e, 0x0b, 0xff, 0xfe, 0x57, 0x65, 0x6c };
09   ZWIR_SetFrequency(ZWIR_mf64MHz);
10   char psk[17] = {"abcdefghijklmnop"};
11   char id[3] = {'I','d','0'};
12
13   ZWIRSEC_AddIkeAuthenticationEntry(&testAddress,64,id,3,psk);
14   ZWIRSEC_AddSecurityPolicy( ZWIRSEC_OutputApply, &testAddress, 128, ZWIR_UDP, 1111, 1111, NULL );
15   ZWIRSEC_AddSecurityPolicy( ZWIRSEC_InputApply,  &testAddress, 128, ZWIR_UDP, 1111, 1111, NULL );
16 }
```

# 6 Security Consideration for IPsec and IKEv2 at ZWIR45xx

Only IPsec with IKEv2 is the secure method providing key freshness for secured connections. As security algorithm AES in counter mode is used. To prevent identical encrypted pattern (as a result of same initialization vectors) it is necessary to change the key at regular intervals. But actually IKEv2 wasn't designed for low power low bandwidth networks like 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 Systems can provide enough energy. Usually IPsec provides enough security with the AES-CTR encryption for 6LoWPAN typical small data rates. It has to take care that the encrypting initialization vector will be reset to zero after a system reset.

# 7 Related Documents

| Document | File Name |
|---|---|
| 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 |
| Internet Key Exchange (IKEv2) Protocol | RFC 5996 - http://www.ietf.org/rfc/rfc5996 |
| ZWIR451x Programming Guide | ZWIR451x_ProgGuide_revX.xy.pdf |

Visit ZMDI's website www.zmdi.com or contact your nearest sales office for the latest version of these documents.

# 8 Glossary

| Term | Description |
|---|---|
| AES | Advanced Encryption Standard |
| IKE | Internet Key Exchange |
| IPsec | Internet Protocol Security |
| ID | Identifier |
| PSK | Pre Shared Key |
| SA | Security Association |
| SP | Security Policy |
| UDP | User Datagram Protocol |
| WPAN | Wireless Personal Area Network |

# 9 Document Revision History

| Revision | Date | Description |
|---|---|---|
| 1.00 | September 30, 2010 | Initial version |
| 1.10 | November 4, 2011 | Clarify prefix definition. Update contact information. |
| 1.11 | August 3, 2012 | Replaced deprecated enum-names<br>Minor edits |


## Sales and Further Information

www.zmdi.com wpan@zmdi.com

| | | | | |
|---|---|---|---|---|
| **Zentrum Mikroelektronik Dresden AG**<br>Grenzstrasse 28<br>01109 Dresden<br>Germany<br><br>Phone +49.351.8822.7476<br>Fax +49.351.8822.87476 | **ZMD America, Inc.**<br>1525 McCarthy Blvd., #212<br>Milpitas, CA 95035-7453<br>USA<br><br>Phone +855-ASK-ZMDI<br>(+855.275.9634) | **Zentrum Mikroelektronik Dresden AG, Japan Office**<br>2nd Floor, Shinbashi Tokyu Bldg.<br>4-21-3, Shinbashi, Minato-ku<br>Tokyo, 105-0004<br>Japan<br><br>Phone +81.3.6895.7410<br>Fax +81.3.6895.7301 | **ZMD FAR EAST, Ltd.**<br>3F, No. 51, Sec. 2,<br>Keelung Road<br>11052 Taipei<br>Taiwan<br><br>Phone +886.2.2377.8189<br>Fax +886.2.2377.8199 | **Zentrum Mikroelektronik Dresden AG, Korean Office**<br>POSCO Centre Building<br>West Tower, 11th Floor<br>892 Daechi, 4-Dong,<br>Kangnam-Gu<br>Seoul, 135-777<br>Korea<br><br>Phone +82.2.559.0660<br>Fax +82.2.559.0700 |