AVR-Kryptoknight Reference Manual

Generated by Doxygen 1.4.2

Wed Jun 1 20:52:32 2005

Contents

Chapter 1

AVR-Kryptoknight

1.1 Introduction

1.1.1 What's in this package

This program consists of five smaller projects put together in a single project. The first project is the Neo-RS232-protocol. This is an idle-RQ stop and wait protocol with a CRC16-checksum. The second and following project all have more or less to do with cryptograpy. The second project is an implementation of a hardware number generator. Secure hardware numbers are very important in cryptography for generating keys and authentication. The third project is the SHA-1 hash calculation algorithm. This has been implemented in assembly language. The fourth project, the HMAC calculation makes use of the SHA-1 calculation to sign messages for authentication purposes. The fifth project, namely the Kryptoknight authentication algorithm combines all of the above projects in a process that enables two machines to authenticate one another.

1.1.2 Why did I make this project?

The main incentive to start this project was to improve security in wireless systems for garagedoors. People lock there houses with sophisticated mechanical keys, but leave their garagedoor nearly unprotected. Current wireless garagedoor systems are actually very simple and easy to hack when one has a sender and a transmitter.

Chapter 2

AVR-Kryptoknight Class Index

2.1 AVR-Kryptoknight Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

AVR-Kryptoknight File Index

3.1 AVR-Kryptoknight File List

Here is a list of all documented files with brief descriptions:

Chapter 4

AVR-Kryptoknight Class Documentation

4.1 FLAGS Struct Reference

Structure combining some flags with different function.

`#include <define.h>`

Public Attributes

- byte waittimer1:1
  becomes 1 when timer1 finished
- byte waitflow2:1
  becomes 1 when flow2 package is received
- byte commError:1
  becomes 1 when an communication error occurred

4.1.1 Detailed Description

Structure combining some flags with different function.

The documentation for this struct was generated from the following file:

- define.h

Chapter 5

AVR-Kryptoknight File Documentation

5.1 authenticate.c File Reference

```
#include "define.h"
```

Functions

- AUTH_ERROR authenticate (byte me[4], byte other[4], byte message[19])
- void dataRXready (byte *buffer, byte length)
- void retransmit (void)
- void noAnswer (void)

Variables

- volatile FLAGS flags

  contains some flags (see define.h)

5.1.1 Detailed Description

Routines that implement the IBM-Kryptoknight authentication protocol.

5.1.2 Function Documentation

5.1.2.1 AUTH_ERROR authenticate (byte *me*[4], byte *other*[4], byte *message*[19])

This function contains the authentication algorithm according to the IBM-Kryptoknight algorithm. This function can only perform the function of ALICE, i.e. this function always starts the authentication procedure. The procedure consists of three flows or messages. Every flow contains the address of the sender, followed by the address of the receiver. In the first flow, a nonce (=large random number) and a MESSAGE are attached. This message, generated by our ALICE is sent to BOB. In the second flow, this function receives a packet from BOB and checks the contents. This flow 2 package contains the two addresses, followed by a nonce nB generated by BOB, followed by an HMAC. This HMAC is calculated over the nonce nA, the MESSAGE, nB and the and the BOB address. If the HMAC is correct, then generate the

third flow package. This package is again consituted of the two addresses, followed by an HMAC. Now the HMAC is calculated over nA and nB

Parameters:

me my address consisting of four bytes (e.g. IP-number)

other address of the other party, also four bytes

message message to send, consisting of nineteen bytes

Returns:

an error code indicating the status of this function

5.1.2.2 void dataRXready (byte * *buffer*, byte *length*)

Function called by the NeoRS232-routines that take care of the lower level RS232 communications. DataRXready sets the flag that lets function authenticate know that data has arrived. Pointer p_RX is also set to point to the received data buffer. When the size of the data is wrong, then a communication error flag is set.

Parameters:

buffer Buffer containing received data

length length of received data

5.1.2.3 void noAnswer (void)

Function called by NeoRS232-routines when no acknowledge is received after several trials.

5.1.2.4 void retransmit (void)

Function called by NeoRS232-routines when no acknowledge is received after sending a package. This function could resend the data.

5.1.3 Variable Documentation

5.1.3.1 volatile FLAGS flags

contains some flags (see define.h)

See also:

define.h)

5.2 CRC_Test.c File Reference

```
#include "define.h"
#include <avr/crc16.h>
```

Functions

- BOOL stripCRC (byte *buffer, byte length)
- void appendCRC (byte *buffer, byte length)

5.2.1 Detailed Description

The protocol makes use of checksums, these functions implement these.

5.2.2 Function Documentation

5.2.2.1 void appendCRC (byte * *buffer*, byte *length*)

This function calculates the CRC of the bytes in the buffer, up to length bytes. Afterwards the CRC is appended to the end of the buffer. As a result, the buffer is two bytes longer than before the call.

Parameters:

buffer contains the data of which the CRC is calculated. Upon return the buffer will be two bytes longer, because the CRC is appended.

length number of bytes in the buffer

5.2.2.2 BOOL stripCRC (byte * *buffer*, byte *length*)

This routine strips the two CRC-bytes off at the end of the buffer. The LSB CRC-byte is followed by the MSB CRC-byte (little endian order). The routine calculates the CRC of the remaining bufferdata. When the calculated and stripped CRC are equal, TRUE is returned, else FALSE is returned.

Parameters:

buffer A buffer with CRC attached at the end (little endian)

length Total number of bytes in the buffer, CRC included

Returns:

TRUE when CRC is correct, otherwise FALSE

5.3 define.h File Reference

```
#include <avr/signal.h>
#include <avr/interrupt.h>
```

Defines

- #define _DEFINE_H_ 1

Typedefs

- typedef unsigned long dword

  Unsigned 32-bit data.

- typedef unsigned short word

  Unsigned 16-bit data.

- typedef unsigned char byte

  Unsigned 8-bit data.

Enumerations

- enum AUTH_ERROR {
  ALL_OK = 0,
  FROM_ERROR,
  TO_ERROR,
  HMAC_ERROR,
  COMM_ERROR }
- enum BOOL {
  FALSE = 0,
  TRUE }

  Boolean constants (in fact integers), substituting one and zero.

5.3.1 Detailed Description

This file contains the declarations of several functions

5.3.2 Define Documentation

5.3.2.1 #define _DEFINE_H_ 1

to prevent multiple definitions of the same constant in one file

5.3.3 Enumeration Type Documentation

5.3.3.1 enum AUTH_ERROR

Enumeration of possible errors returned from the authenticate function.

Enumeration values:

ALL_OK No error in authentication routine.

FROM_ERROR Source address incorrect.

TO_ERROR Destination address incorrect.

HMAC_ERROR HMAC not correct.

COMM_ERROR Communication error in NeoRS232 routines.

5.3.3.2 enum BOOL

Boolean constants (in fact integers), substituting one and zero.

Enumeration values:

FALSE An constant integer that is zero.

TRUE A constant integer that is one.

5.4 hmac.c File Reference

```
#include <avr/eeprom.h>
#include "define.h"
```

Defines

- #define NEED_HASH

  HMAC is based on hashing, so needs hashing routines.

- #define IPAD 0x36

  defined in RFC2104

- #define OPAD 0x5C

  defined in RFC2104

Functions

- byte * getHmac (byte text[55])

5.4.1 Detailed Description

This file contains functions to implement the HMAC calculation. A hmac provides authentication of data, based on a shared secret key. Authentication is a way to make sure the users really are who they say they are. So if data with an HMAC comes in. You compute the HMAC yourself using the secret key you share with the other party. If this computed HMAC is exactly the same as the received HMAC, then you are sure that the data came from the other communicating party. Offcourse you must be sure that only you and the other communicating party has the secret key.

5.4.2 Function Documentation

5.4.2.1 byte* getHmac (byte *text*[55])

Calculates an HMAC (RFC2104) of 55-byte data and a 512-bit key, a 160-bit (20-byte) HMAC is returned

Parameters:

text The 55-byte data of which the HMAC will be calculated.

5.5 main.c File Reference

```
#include "define.h"
```

Functions

- int main (void)

5.5.1 Detailed Description

This is the main file of the project, containing the main-loop.

5.5.2 Function Documentation

5.5.2.1 int main (void)

Main function. After boot-code finishes, it jumps to this routine. This function configures PORTB as an output port. I connected eight leds to this port. It also initializes the USART and calls a test- routine, in this case the authentication test.

5.6 protocol.h File Reference

Defines

- #define LEADING_FLAG 0xAA
  Start of package.
- #define DLE_FLAG 0x10
  Byte stuffing flag.
- #define TRAILING_FLAG 0xFF
  End of package.
- #define ACK0_FLAG 0x30
  Acknowledge flag (separate packet).
- #define RS232LENGTH 50
  USART receive buffer size.

Enumerations

- enum FRAME {
  INFO_FRAME,
  ACK_FRAME }
  Possible frametypes in the NeoRS232-protocol.

5.6.1 Detailed Description

This file contains the declarations of constants used in the NeoRS232 routines.

5.6.2 Enumeration Type Documentation

5.6.2.1 enum FRAME

Possible frametypes in the NeoRS232-protocol.

Enumeration values:

INFO_FRAME Constant defining a frame containing data.

ACK_FRAME Constant defining a frame containing an acknowledge

5.7 random.c File Reference

```
#include "define.h"
```

Functions

- void getRandom (byte bit128Random[16])

Variables

- volatile FLAGS flags

  contains some flags (see define.h)

5.7.1 Detailed Description

File containing routines to generate truly random numbers, based on a hardware random number generator.

5.7.2 Function Documentation

5.7.2.1 void getRandom (byte *bit128Random*[16])

This routine calculates 128-bit random numbers. The random number generating engine is seeded with 32-bit random numbers from a hardware random number generator. The design of the random number generator engine is that of the Intel RNG.

Parameters:

bit128Random a pointer to a 16-byte array. It will contain the 128-bit randomword when the function returns.

5.7.3 Variable Documentation

5.7.3.1 volatile FLAGS flags

contains some flags (see define.h)

See also:

define.h)

5.8 serieData.c File Reference

```
#include "define.h"
#include "protocol.h"
```

Functions

- void schrijfString (byte *buffer, byte lengte, byte typeFrame)
- void writeData (byte *buffer, byte grootte)
- void bewerkDatagram (byte *buffer, byte teller)

5.8.1 Detailed Description

Higher OSI-level function of the NeoRS232-protocol for writing and receiving the data.

5.8.2 Function Documentation

5.8.2.1 void bewerkDatagram (byte * *buffer*, byte *teller*)

When data comes in. This function checks if the checksum is correct. If it is, then the dataRSReady routine is called. The user must implement the dataRXReady-routine him-/herself. The user should never call the bewerkDatagram-routine. This function is called by the lower OSI-level when data comes in.

Parameters:

buffer A buffer containing the received data

teller The number of bytes that is received

5.8.2.2 void schrijfString (byte * *buffer*, byte *lengte*, byte *typeFrame*)

This function receives a buffer with a length "lengte". This function converts the buffer to the NeoRS232-protocol by inserting a startflag, bytestuffing flags and also a trailing flag.

Parameters:

buffer Buffer containing the data to send

lengte Number of bytes out of buffer to be sent

typeFrame Identifier indicating what type of frame must be sent (INFO_FRAME or ACK_FRAME)

5.8.2.3 void writeData (byte * *buffer*, byte *grootte*)

A user wanting to use the NeoRS232-protocol must call this function to send data. This upper OSI-layer function sends the data down to the lower OSI-levels. The functions send the data then to the RS232.

Parameters:

buffer Buffer is a buffer containing the data to send

grootte The number of bytes to send

5.9 simpleRead.c File Reference

```
#include "define.h"
#include "protocol.h"
```

Functions

- void bewerkDatagram (byte *buffer, byte teller)
- SIGNAL (SIG_UART_RECV)

5.9.1 Detailed Description

Functions implementing the low-level read functions of the NeoRS232- protocol.

5.9.2 Function Documentation

5.9.2.1 void bewerkDatagram (byte * *buffer*, byte *teller*)

When data comes in. This function checks if the checksum is correct. If it is, then the dataRSReady routine is called. The user must implement the dataRXReady-routine him-/herself. The user should never call the bewerkDatagram-routine. This function is called by the lower OSI-level when data comes in.

Parameters:

buffer A buffer containing the received data

teller The number of bytes that is received

5.9.2.2 SIGNAL (SIG_UART_RECV)

This is the ISR-routine that will be executed when data arrives at the USART. It calls one of the NeoRS232-routines that takes further care of the received data.

5.10 simpleWrite.c File Reference

```
#include "define.h"
#include "protocol.h"
```

Functions

- void schrijfString (byte *buffer, byte lengte, byte typeFrame)

5.10.1 Detailed Description

Functions implementing the lowlevel write functions of the NeoRS232-protocol.

5.10.2 Function Documentation

5.10.2.1 void schrijfString (byte * *buffer*, byte *lengte*, byte *typeFrame*)

This function receives a buffer with a length "lengte". This function converts the buffer to the NeoRS232-protocol by inserting a startflag, bytestuffing flags and also a trailing flag.

Parameters:

buffer Buffer containing the data to send

lengte Number of bytes out of buffer to be sent

typeFrame Identifier indicating what type of frame must be sent (INFO_FRAME or ACK_FRAME)

5.11 test.c File Reference

```
#include "define.h"
```

Functions

- void testAuthenticate (void)

5.11.1 Detailed Description

This file contains testroutines, comment them out when compiling the final realization.

5.11.2 Function Documentation

5.11.2.1 void testAuthenticate (void)

You can test the authentication routine with this routine. The protocol is instantiated with a source address of {0,1,2,2}, a destination address of {0,1,2,3} and a message of {0,1,2,3, ... , 17, 18,19} The returned error codes will be shown on PORTB.

5.12 timer1.c File Reference

```
#include "define.h"
```

Functions

- SIGNAL (SIG_INPUT_CAPTURE1)
- void timer1_Activate (void)
- void timer1_DeActivate (void)

  Disable timer1 input capture interrupts.

5.12.1 Detailed Description

Here some routines are defined to control timer1

5.12.2 Function Documentation

5.12.2.1 SIGNAL (SIG_INPUT_CAPTURE1)

Interrupt service routine of input capture on timer1 generating 32-bit random-numbers. When a falling edge on ICP1 occurs, the value of timer1 is copied to the ICR1 register and this signal input capture interrupt is triggered. This ISR then resets timer1. The result is that ICR1 always contains the time between two input capture interrupts, i.e. the period. Period N, is compared to the period N-2 of two interrupts ago. If period N-2 $<$ period N then shift in a one in our random dword, otherwise shift in a zero. After 32 bits are shift in, the waittimer1-flag is set. The ICP-pin of the microcontroller is connected to a simple astable multivibrator, consisting of two NOT-ports, two resistors and two capacitors. The small variations in the period of such a square-wave generator are the basis for the randomness of this hardware random number generator.

5.12.2.2 void timer1_Activate (void)

Activate timer1: normal operation (no PWM), interrupts on falling edge of IP1, IRQ on input capture

5.13 timer2.c File Reference

```
#include "define.h"
```

Functions

- void initTimer2 (void)
- void stopTimer2 (void)
- INTERRUPT (SIG_OUTPUT_COMPARE2)

5.13.1 Detailed Description

Functions for controlling timer2. NeoRS232-routines use timer2 as a timeout-timer with a period of five seconds.

5.13.2 Function Documentation

5.13.2.1 void initTimer2 (void)

This routine sets timer2. Interrupt when timer2 register=OCR2, enable interrupts on timer2 set period=5s

5.13.2.2 INTERRUPT (SIG_OUTPUT_COMPARE2)

This ISR-routine is called when the value in timer2 reaches OCR2. This happens normally after five seconds. In good communications the timeout doesn't occur. When the acknowledge is not received within five seconds, this function is called for the first time. A retransmit function (implemented elsewhere) is called when timeouts occur. After 25 seconds, i.e. after five timeouts, the sender gives up. A function is called to let the other functions know that no data has been received.

5.13.2.3 void stopTimer2 (void)

Stop timer2 (disable IRQ)

5.14 usart.c File Reference

```
#include "define.h"
```

Functions

- void init_USART (void)
- void send_USART (byte character)

5.14.1 Detailed Description

This function takes care of the USART. This is the lowest possible OSI-layer on the MCU.

5.14.2 Function Documentation

5.14.2.1 void init_USART (void)

Set the USART with 9600baud, enable RX and TX, enable IRQ on receiving data.

5.14.2.2 void send_USART (byte *character*)

Send a character to the USART

Parameters:

character a byte to send

Index