# Java-Kryptoknight Reference Manual

Generated by Doxygen 1.4.2

Mon Apr 25 20:29:53 2005

# Contents

# Chapter 1

# Java-Kryptoknight

## 1.1 Introduction

### 1.1.1 What's in this package

In this package you can find the Java implementation of the IBM-Kryptoknight protocol. IBM developed this protocol for authentication especially for devices with limited calculation abilities such as small microcontrollers. Here the implementation is made in Java. This will make it possible to authenticate several PC to eachother or to authenticate a microcontroller to a PC. In contrast with the AVR-Kryptoknight package for the Atmel ATMEGA8535 microcontroller, this package can function as a station waiting for an authentication request. This is the so called BOB-mode. The other available mode is the ALICE-mode. Alice always starts the authentication procedure.

## 1.2 References

There is a lot information available on the internet about authentication. I focused on the Kryptoknight-protocol because it's a lightweight protocol that suited my needs best.

### 1.2.1 Authentication


- BIRD R., et al., Systematic Design of a Family of Attack-Resistant Authentication Protocols
- BIRD R., et al., The Kryptoknight Family of Light-weight Protocols for Authentication and Key Distribution, December 1993
- CLARK J., JACOB J. A Survey of Authentication Protocol Literature: Version 1.0
- COMPTECHDOC, Authentication protocols http://www.comptechdoc.org/ independent/networking/protocol/protauthen.html, December 3, 2004
- CRISPO B., et al., Symmetric Key Authentication Services Revisited
- INFOSYSSEC, The Security Portal for Information System Security Professionals http://www.infosyssec.com/infosyssec/secauthen1.htm
- JANSON P., et al. Scalability and Flexibility in Authentication Services: The Kryptoknight approach, 1997

- MOLVA K., et al., Kryptoknight Authentication and Key Distribution System
- MOLVA K., et al., Authentication of Mobile Users, August 20, 1993
- RFC1994 - PPP Challenge Handshake Authentication Protocol (CHAP)
- TSUDIK G. VAN HERREWEGHEN E., On Simple and Secure Key Distribution


### 1.2.2 Hmac


- BELLARE M., et al., Keying Hash Functions for Message Authentication
- FIPS PUB 198 The Keyed-Hash Message Authentication Code (HMAC)
- IETF, RFC2104 HMAC: Keyed-Hashing for Message Authentication, February 1997


### 1.2.3 Hashing


- FIPS PUB 180-2 Announcing the Secure Hash Standard, August 1, 2002
- HUSNI A. Implementation Secure Hash Standard with AVR Microcontroller
- IETF, RFC3174 US Secure Hash Algorithm 1 (SHA1)


### 1.2.4 Random Number Generators


- BERNSTEIN G.M., et al., Method and Apparatus for generating Secure Random Numbers using Chaos, US Patent US5007087, April 9, 1991.
- DAVIES R., Hardware Random Number Generators, http://www.robertnz.net/hwrng.htm
- GILLEY J.E., Method and Apparatus for generating truly random numbers, US Patent US5781458, July 14, 1998
- GOLBECK E. C., Random Bit Stream Generator and Method, US Patent US5239494, August 24, 1993.
- HOFFMAN E.J., Random Number Generator, US Patent US6061702, May 9 2000
- INTEL, http://web.archive.org/web/20040229005150/http://developer.intel.com/design
- JAKOBSSON M., et al. A Physical Secure Random Bit Generator
- JUN B.,et al., The Intel Random Number Generator, April 22, 1999
- LOGUE A., Hardware Random Number Generator, May 2002 http://www.cryogenius.com/hardware/rng/
- MESSINA M. et al., Random Bit Sequence Generator, US Patent US2003/0093455 A1, May 15, 2003
- RSA, Hardware-Based Random Number Generation, An RSA Data Security White Paper
- SKLAVOS N., Random Number Generator Architecture And Vlsi Implementation, May 2002
- UNER E., Generating Random Numbers, Embedded.com http://www.embedded.com/showArticle.jhtml?articleID=20900500

- WELLS S.E., Programmable Random Bit Source, US Patent US6795837, September 21, 2004
- WELLS S.E., et al.Secure Handware Random Number Generator, US Patent US6792438, 14 sept 2004
- WIKIPEDIA, Hardware Random Number Generator, http://en.wikipedia.org/wiki/Hardware_-random_number_generator
- WILLWARE, Hardware Random Bit Generator, http://willware.net:8080/hw-rng.html


### 1.2.5 Random Noise Sources


- ELECTRONIC DESIGN, Wide-Band Analog White-Noise Generator, November 3, 1997, http://www.elecdesign.com/Articles/Index.cfm?AD=1&ArticleID=6356
- SHUPE C.D., Random Voltage Source With Subtstantially Uniform Distribution, US Patent US4578649

# Chapter 2

# Java-Kryptoknight Class Index

## 2.1 Java-Kryptoknight Class List

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

# Chapter 3

# Java-Kryptoknight Class Documentation

## 3.1 AuthPacket Class Reference

### Package Functions

- AuthPacket (byte[ ] from, byte[ ] to, byte[ ] nonce, byte[ ] x)
- AuthPacket (byte[ ] from, byte[ ] to, byte[ ] hmac)
- AuthPacket (byte[ ] array, byte flow)
- byte[ ] getFrom ()
- byte[ ] getNa ()
- byte[ ] getNb ()
- byte[ ] getMessage ()
- byte[ ] toBytes (byte flow)
- boolean checkFlow1 (byte[ ] me)
- boolean checkFlow2 (byte[ ] me, byte[ ] other, byte[ ] na, byte[ ] message, byte[ ] secretKey)
- boolean checkFlow3 (byte[ ] me, byte[ ] other, byte[ ] na, byte[ ] nb, byte[ ] secretKey)
- boolean compare (byte[ ] a, byte[ ] b)
- void showArray (byte[ ] array, String name)

### 3.1.1 Detailed Description

Class that determines the lower OSI-level structure of the Kryptoknight messages. This class also checks the validity of those packages.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 AuthPacket.AuthPacket (byte[ ] *from*, byte[ ] *to*, byte[ ] *nonce*, byte[ ] *x*) `[package]`

Constructor for FLOW1 and FLOW2 packages.

**Parameters:**

***from*** Source address

***to*** Destination address

***nonce*** NonceA for FLOW1, NonceB for FLOW2

***x*** Message in FLOW1, Hmac in FLOW2

#### 3.1.2.2 AuthPacket.AuthPacket (byte[ ] *from*, byte[ ] *to*, byte[ ] *hmac*) [package]

Constructor for FLOW3 packages.

**Parameters:**

***from*** Source address

***to*** Destination address

***hmac*** Hmac

#### 3.1.2.3 AuthPacket.AuthPacket (byte[ ] *array*, byte *flow*) [package]

Constructor that can make a package out of a byte array. Packages for all flows can be constructed with this method.

**Parameters:**

***array*** Array containing an incoming package

***flow*** Flow number (FLOW1, FLOW2 or FLOW3)

### 3.1.3 Member Function Documentation

#### 3.1.3.1 boolean AuthPacket.checkFlow1 (byte[ ] *me*) [package]

Checks a flow1 package

**Parameters:**

***me*** source address

**Returns:**

true when package is correct

#### 3.1.3.2 boolean AuthPacket.checkFlow2 (byte[ ] *me*, byte[ ] *other*, byte[ ] *na*, byte[ ] *message*, byte[ ] *secretKey*) [package]

Checks a flow2 package

**Parameters:**

***me*** source address

***other*** destination address

***na*** nonceA

***message*** message of flow1

***secretKey*** bytearray containing secret key

**Returns:**

true when package is a correct flow2 package

#### 3.1.3.3 boolean AuthPacket.checkFlow3 (byte[ ] *me*, byte[ ] *other*, byte[ ] *na*, byte[ ] *nb*, byte[ ] *secretKey*) [package]

Checks a flow3 package

**Parameters:**

***me*** Expected source address

***other*** Expected destination address

***na*** Expected nonceA

***nb*** Expected nonceB

***secretKey*** secretKey that will be used to calculate HMAC

**Returns:**

true when package is a correct flow3 package

#### 3.1.3.4 boolean AuthPacket.compare (byte[ ] *a*, byte[ ] *b*) [package]

Compares two arrays for equality

**Parameters:**

***a*** first array

***b*** second array

**Returns:**

true when arrays are equal

#### 3.1.3.5 byte [ ] AuthPacket.getFrom () [package]

Get source address of a package

**Returns:**

bytearray containing source address

#### 3.1.3.6 byte [ ] AuthPacket.getMessage () [package]

Get message out of a package

**Returns:**

bytearray containing message

#### 3.1.3.7 byte [ ] AuthPacket.getNa () [package]

Get nonceA of a package

**Returns:**

bytearray containing nonceA.

#### 3.1.3.8 byte [ ] AuthPacket.getNb () [package]

Get nonceB of a package

**Returns:**
bytearray containing nonceB

#### 3.1.3.9 void AuthPacket.showArray (byte[ ] *array*, String *name*) [package]

Prints the contents of an array. Only used for debugging purposes.

**Parameters:**
***array*** Array to show
***name*** Name of the array to show

#### 3.1.3.10 byte [ ] AuthPacket.toBytes (byte *flow*) [package]

Converts a package to a bytearray.

**Parameters:**
***flow*** Flow number (FLOW1, FLOW2 or FLOW3)

**Returns:**
bytearray containing the package

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

- AuthPacket.java

# 3.2 ByteArray Class Reference

## Package Functions

- ByteArray ()
- void add (byte[ ] b)
- byte[ ] toByte ()

### 3.2.1 Detailed Description

A helper class that makes it possible to have an array with variable length.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 ByteArray.ByteArray () [package]

Constructor. Makes a new ByteArray-object with an empty array.

### 3.2.3 Member Function Documentation

#### 3.2.3.1 void ByteArray.add (byte[ ] *b*) [package]

Method to add an array to the current object.

**Parameters:**

***b*** Array to add to the ByteArray object

#### 3.2.3.2 byte [ ] ByteArray.toByte () [package]

Method that converts the array of this object to an array of bytes

**Returns:**

the array of bytes.

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

- ByteArray.java

## 3.3 FlowState Interface Reference

### Public Attributes

- byte FLOW1 = 1

  *Flow1, first flow (generated by ALICE) of the Kryptoknight protocol.*

- byte FLOW2 = 2

  *Flow2, second flow (generated by BOB) of the Krypoknight protocol.*

- byte FLOW3 = 3

  *Flow3, third flow (generated by ALICE) of the Kryptoknight protocol.*

### 3.3.1 Detailed Description

Interface grouping the possible flows

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

- Kryptoknight.java

# 3.4 Hmac Class Reference

## Static Package Functions

- static SecretKey generateKey ()
- static byte[ ] selfHash (byte[ ] key, byte[ ] text)
- static void showArray (byte[ ] array, String name)

## 3.4.1 Detailed Description

A class that implements the HMAC-algorithm described in RFC2104

## 3.4.2 Member Function Documentation

### 3.4.2.1 static SecretKey Hmac.generateKey () [static, package]

Method that generates a secret key for use in HMAC

**Returns:**
The secret key

### 3.4.2.2 static byte [ ] Hmac.selfHash (byte[ ] *key*, byte[ ] *text*) [static, package]

Calculate the HMAC. Java also implements this function "javax.crypto.Mac" The disadvantage of the java implementation is that it's impossible to pass the secret key as a byte-array.

**Parameters:**
***key*** Secret key (64-bytes)
***text*** Message to authenticate

**Returns:**
the 20-byte (160-bit) HMAC

### 3.4.2.3 static void Hmac.showArray (byte[ ] *array*, String *name*) [static, package]

Shows an array

**Parameters:**
***array*** Array to show
***name*** Name of the array to show

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

- Hmac.java

## 3.5 Kryptoknight Class Reference

### Public Member Functions

- Kryptoknight (byte[ ] me, byte[ ] sk, String port)
- void actionPerformed (ActionEvent e)
- String toString ()

### Static Public Member Functions

- static void main (String[ ] args)

### Package Functions

- void close ()
- boolean authenticate (byte[ ] other, byte[ ] message)
- boolean authenticateLowLevel (byte[ ] other, byte[ ] message)

### Static Package Functions

- static void showArray (byte[ ] array, String name)
- static void threading ()

### Static Package Attributes

- static final int MESSAGE_SIZE = 19

  *A variable specifying the length of the message.*

- static final int ADDRESS_SIZE = 4

  *Address contained in the packets consists of 4 bytes.*

- static final int NONCE_SIZE = 16

  *The random number (=nonce) consists of 16bytes = 128bit.*

- static final int MAC_SIZE = 20

  *HMAC with SHA-1 is used, so length of the hash is 20bytes = 160bit.*

### Classes

- class TijdKlok

### 3.5.1 Detailed Description

This class implements the Kryptoknight protocol. It specifies which action to undertake when a certain packet arrives.

## 3.5.2 Constructor & Destructor Documentation

### 3.5.2.1 Kryptoknight.Kryptoknight (byte[ ] *me*, byte[ ] *sk*, String *port*)

The constructor for the Kryptoknight class

**Parameters:**

***me*** Address of this object (4 bytes)

***sk*** 64-byte secret key

***port*** COM-port that must be used for authentication (COM1 or COM2)

## 3.5.3 Member Function Documentation

### 3.5.3.1 void Kryptoknight.actionPerformed (ActionEvent *e*)

Routine that will be called by the lower level OSI-routines when data comes in. It calls routines that check the incoming packets. When the packets are correct a response can be sent.

**Parameters:**

***e*** The ActionEvent

### 3.5.3.2 boolean Kryptoknight.authenticate (byte[ ] *other*, byte[ ] *message*) [package]

Command used for authenticating another party.

**Parameters:**

***other*** Address of the other party (4bytes)

***message*** Message to authenticate

**Returns:**

true when authentication passed, otherwise false

### 3.5.3.3 boolean Kryptoknight.authenticateLowLevel (byte[ ] *other*, byte[ ] *message*) [package]

This function takes care of the message flow in the Kryptoknight protocol. This is by far the most important function.

**Parameters:**

***other*** Address of the other to authenticate

***message*** Message that must be authenticated

**Returns:**

true when authentication passed, otherwise false.

### 3.5.3.4 void Kryptoknight.close () [package]

This command closes the communication port. This must be called before closing the program.

#### 3.5.3.5 static void Kryptoknight.main (String[ ] *args*) [static]

The main function initiates two new objects of the Kryptoknight class and lets them authenticate eachother.

**Parameters:**

***args*** Command line parameters

#### 3.5.3.6 static void Kryptoknight.showArray (byte[ ] *array*, String *name*) [static, package]

A useful routine for debugging purposes. It shows an array on the output capture window.

**Parameters:**

***array*** Array to show

***name*** Name of the array that will be shown

#### 3.5.3.7 static void Kryptoknight.threading () [static, package]

A debug routine that shows which threads are currently running.

#### 3.5.3.8 String Kryptoknight.toString ()

The toString method that shows the elements of an object of the Kryptoknight class.

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

- Kryptoknight.java

# 3.6 Kryptoknight.TijdKlok Class Reference

## Public Member Functions

- void actionPerformed (ActionEvent e)

## Package Functions

- TijdKlok (int waittime)
- void timerStarten ()
- void timerStoppen ()

## Package Attributes

- Timer timer

  *The timer object holding the time.*

- int waittime

  *Time in seconds that must be waited.*

### 3.6.1 Detailed Description

Inner class for keeping time.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 Kryptoknight.TijdKlok.TijdKlok (int *waittime*) [package]

Initialize the timer with a waiting interval "waittime" in seconds.

**Parameters:**

***waittime*** Time that must pass before an interrupt is called

### 3.6.3 Member Function Documentation

#### 3.6.3.1 void Kryptoknight.TijdKlok.actionPerformed (ActionEvent *e*)

Method that will be called when the "waittime" interval has passed.

**Parameters:**

***e*** The ActionEvent

#### 3.6.3.2 void Kryptoknight.TijdKlok.timerStarten () [package]

Start the timer

#### 3.6.3.3 void Kryptoknight.TijdKlok.timerStoppen () [package]

Stop the timer

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

- Kryptoknight.java

# 3.7 ModeState Interface Reference

## Public Attributes

- boolean BOB_MODE = false

## Package Attributes

- boolean ALICE_MODE = true

### 3.7.1 Detailed Description

Interface grouping the possible communicating states

### 3.7.2 Member Data Documentation

#### 3.7.2.1 boolean ModeState.ALICE_MODE = true [package]

This program now works in ALICE_MODE. This is the other possible mode. In ALICE_MODE, this program generates the flow1 messages and then waits for a flow3 message to come in.

#### 3.7.2.2 boolean ModeState.BOB_MODE = false

Two modes are possible. BOB_MODE is the mode in which this program acts as BOB in the Kryptoknight protocol. i.e. The program waits for a flow2 package to come in. It will send a flow3 package when the flow2 package was found correct.

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

- Kryptoknight.java

## 3.8 Nonce Class Reference

### Package Functions

- Nonce ()
- byte[ ] generateNonce (int length)

### 3.8.1 Detailed Description

A helper class to generate the nonces. Nonces are cryptographically strong random numbers so pseudorandom numbers are not suited.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 Nonce.Nonce () [package]

Constructor initiating the random number generator

### 3.8.3 Member Function Documentation

#### 3.8.3.1 byte [ ] Nonce.generateNonce (int *length*) [package]

Method that generates a nonce of the given length.

**Parameters:**

***length*** Length of the nonce in bytes.

**Returns:**

an array of bytes containing the random number.

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

- Nonce.java

# 3.9 WaitingState Interface Reference

## Public Attributes

- byte NOT_WAITING = 0

  *ALICE is not waiting.*

- byte WAITING = 1

  *ALICE is not waiting ALICE is waiting for a response of BOB.*

- byte NO_RESPONSE = 2

  *ALICE is not waiting ALICE is waiting for a response of BOB BOB doesn't respond.*

### 3.9.1 Detailed Description

Interface that groups some constants. These constants are used for defining in which state ALICE is when waiting for a flow3 package.

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

- Kryptoknight.java

# Index