vt-middleware / cryptacular Public

Code Issues 5 Pull requests 2 Actions Projects Wiki

master

cryptacular / src / main / java / org / cryptacular / CiphertextHeader.java / Jump to

dfish3r Fix API compatibility for v1.2.4 … History

2 contributors

263 lines (234 sloc) | 7 KB

```java
/* See LICENSE for licensing and NOTICE for copyright. */
package org.cryptacular;

import java.io.IOException;
import java.io.InputStream;
import java.nio.BufferUnderflowException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import org.cryptacular.util.ByteUtil;

/**
 * Cleartext header prepended to ciphertext providing data required for decryption.
 *
 * <p>Data format:</p>
 *
 * <pre>
     +-----+----------+-------+------------+---------+
     | Len | NonceLen | Nonce | KeyNameLen | KeyName |
     +-----+----------+-------+------------+---------+
 * </pre>
 *
 * <p>Where fields are defined as follows:</p>
 *
 * <ul>
 *   <li>Len - Total header length in bytes (4-byte integer)</li>
 *   <li>NonceLen - Nonce length in bytes (4-byte integer)</li>
 *   <li>Nonce - Nonce bytes (variable length)</li>
 *   <li>KeyNameLen (OPTIONAL) - Key name length in bytes (4-byte integer)</li>
 *   <li>KeyName (OPTIONAL) - Key name encoded as bytes in platform-specific encoding (variable length)</li>
 * </ul>
 *
 * <p>The last two fields are optional and provide support for multiple keys at the encryption provider. A common case
 * for multiple keys is key rotation; by tagging encrypted data with a key name, an old key may be retrieved by name to
 * decrypt outstanding data which will be subsequently re-encrypted with a new key.</p>
 *
 * @author  Middleware Services
 *
 * @deprecated Superseded by {@link CiphertextHeaderV2}
 */
@Deprecated
public class CiphertextHeader
{
  /** Maximum nonce length in bytes. */
  protected static final int MAX_NONCE_LEN = 255;

  /** Maximum key name length in bytes. */
  protected static final int MAX_KEYNAME_LEN = 500;

  /** Header nonce field value. */
  protected final byte[] nonce;

  /** Header key name field value. */
  protected String keyName;

  /** Header length in bytes. */
  protected int length;


  /**
   * Creates a new instance with only a nonce.
   *
   * @param  nonce  Nonce bytes.
   */
  public CiphertextHeader(final byte[] nonce)
  {
    this(nonce, null);
  }


  /**
   * Creates a new instance with a nonce and named key.
   *
   * @param  nonce  Nonce bytes.
   * @param  keyName  Key name.
   */
  public CiphertextHeader(final byte[] nonce, final String keyName)
  {
    if (nonce.length > MAX_NONCE_LEN) {
        throw new IllegalArgumentException("Nonce exceeds size limit in bytes (" + MAX_NONCE_LEN + ")");
      }
      if (keyName != null) {
        if (ByteUtil.toBytes(keyName).length > MAX_KEYNAME_LEN) {
          throw new IllegalArgumentException("Key name exceeds size limit in bytes (" + MAX_KEYNAME_LEN + ")");
        }
      }
      this.nonce = nonce;
      this.keyName = keyName;
      length = computeLength();
    }

    /**
     * Gets the header length in bytes.
     *
     * @return  Header length in bytes.
     */
    public int getLength()
    {
      return this.length;
    }

    /**
     * Gets the bytes of the nonce/IV.
     *
     * @return  Nonce bytes.
     */
    public byte[] getNonce()
    {
      return this.nonce;
    }

    /**
     * Gets the encryption key name stored in the header.
     *
     * @return  Encryption key name.
     */
    public String getKeyName()
    {
      return this.keyName;
    }


    /**
     * Encodes the header into bytes.
     *
     * @return  Byte representation of header.
     */
    public byte[] encode()
    {
      final ByteBuffer bb = ByteBuffer.allocate(length);
      bb.order(ByteOrder.BIG_ENDIAN);
      bb.putInt(length);
      bb.putInt(nonce.length);
      bb.put(nonce);
      if (keyName != null) {
        final byte[] b = keyName.getBytes();
        bb.putInt(b.length);
        bb.put(b);
      }
      return bb.array();
    }


    /**
     * @return  Length of this header encoded as bytes.
     */
    protected int computeLength()
    {
      int len = 8 + nonce.length;
      if (keyName != null) {
        len += 4 + keyName.getBytes().length;
      }
      return len;
    }


    /**
     * Creates a header from encrypted data containing a cleartext header prepended to the start.
     *
     * @param  data  Encrypted data with prepended header data.
     *
     * @return  Decoded header.
     *
     * @throws  EncodingException  when ciphertext header cannot be decoded.
     */
    public static CiphertextHeader decode(final byte[] data) throws EncodingException
    {
      final ByteBuffer bb = ByteBuffer.wrap(data);
      bb.order(ByteOrder.BIG_ENDIAN);

      final int length = bb.getInt();
      if (length < 0) {
        throw new EncodingException("Bad ciphertext header");
      }

      final byte[] nonce;
      int nonceLen = 0;
    try {
      nonceLen = bb.getInt();
      if (nonceLen > MAX_NONCE_LEN) {
        throw new EncodingException("Bad ciphertext header: maximum nonce length exceeded");
      }
      nonce = new byte[nonceLen];
      bb.get(nonce);
    } catch (IndexOutOfBoundsException | BufferUnderflowException e) {
      throw new EncodingException("Bad ciphertext header");
    }

    String keyName = null;
    if (length > nonce.length + 8) {
      final byte[] b;
      int keyLen = 0;
      try {
        keyLen = bb.getInt();
        if (keyLen > MAX_KEYNAME_LEN) {
          throw new EncodingException("Bad ciphertext header: maximum key length exceeded");
        }
        b = new byte[keyLen];
        bb.get(b);
        keyName = new String(b);
      } catch (IndexOutOfBoundsException | BufferUnderflowException e) {
        throw new EncodingException("Bad ciphertext header");
      }
    }

    return new CiphertextHeader(nonce, keyName);
  }


  /**
   * Creates a header from encrypted data containing a cleartext header prepended to the start.
   *
   * @param  input  Input stream that is positioned at the start of ciphertext header data.
   *
   * @return  Decoded header.
   *
   * @throws  EncodingException  when ciphertext header cannot be decoded.
   * @throws  StreamException  on stream IO errors.
   */
  public static CiphertextHeader decode(final InputStream input) throws EncodingException, StreamException
  {
    final int length = ByteUtil.readInt(input);
    if (length < 0) {
      throw new EncodingException("Bad ciphertext header");
    }

    final byte[] nonce;
    int nonceLen = 0;
    try {
      nonceLen = ByteUtil.readInt(input);
      if (nonceLen > MAX_NONCE_LEN) {
        throw new EncodingException("Bad ciphertext header: maximum nonce size exceeded");
      }
      nonce = new byte[nonceLen];
      input.read(nonce);
    } catch (ArrayIndexOutOfBoundsException e) {
      throw new EncodingException("Bad ciphertext header");
    } catch (IOException e) {
      throw new StreamException(e);
    }

    String keyName = null;
    if (length > nonce.length + 8) {
      final byte[] b;
      int keyLen = 0;
      try {
        keyLen = ByteUtil.readInt(input);
        if (keyLen > MAX_KEYNAME_LEN) {
          throw new EncodingException("Bad ciphertext header: maximum key length exceeded");
        }
        b = new byte[keyLen];
        input.read(b);
      } catch (ArrayIndexOutOfBoundsException e) {
        throw new EncodingException("Bad ciphertext header");
      } catch (IOException e) {
        throw new StreamException(e);
      }
      keyName = new String(b);
    }

    return new CiphertextHeader(nonce, keyName);
  }

}
```