Enable two factor authentication for your web application to use with Google Authenticator App.

kalpads.blogspot.tw/2013/10/enable-two-factor-authentication-for.html

Introduction.

If you have already enabled “2-Step Verification” for your Google account (Or any other)you may aware of what we are going to discuss today. For all others “2-Step Verification” is simply another mechanism to keep safe your Google account from the “bad guys”.

Basically what happens is when you enable the 2-Step Verification (more generically two factor authentication) addition to the your user name password combination it will ask for a verification code as the second gateway to your account. As the primary option Google offers SMS or voice call as the delivery mechanism for the verification code.

Or we can switch to “Google Authenticator App” which is available freely on Play Store and App Store. In this post I am going to discuss how we can implement our own 2-Step Verification for Google Authenticator App is the client side of this story, and google has their own server side to enable service for google users and we are going to have our own today.

This mechanism powered by TOTP- Time Based **One Time Password** Algorithm .You can find the specification here.

Below set of images will explain you what is happening behind there.

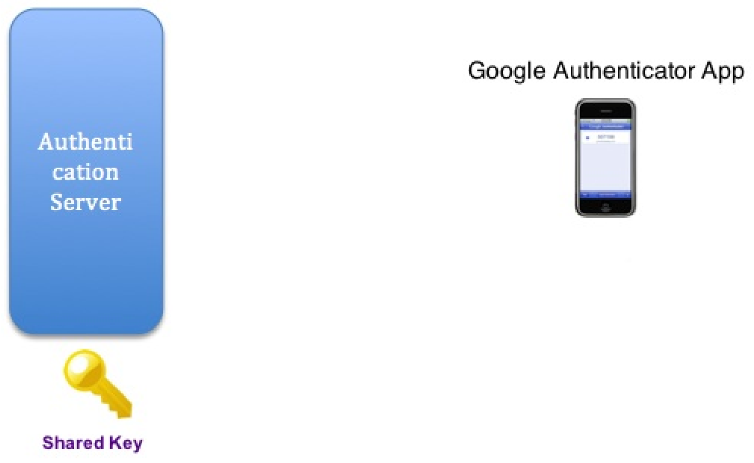


Figure 1.1- Authentication Server Generates an key to share with the GA app.

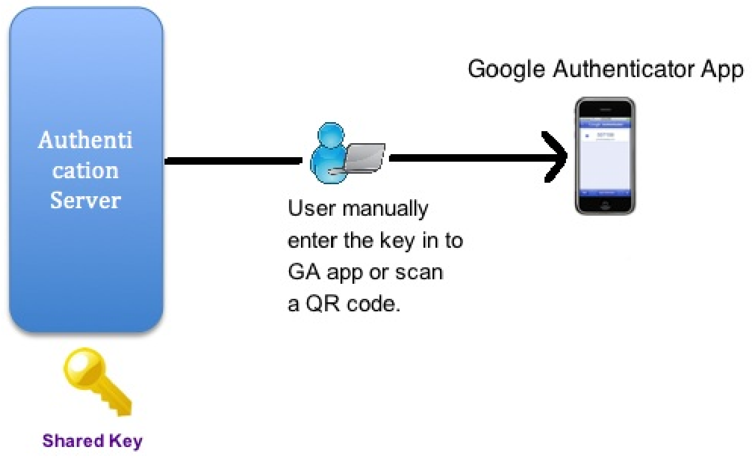
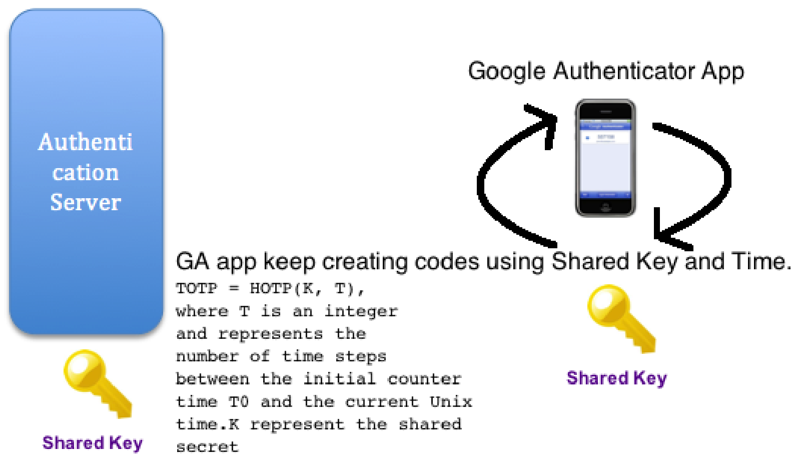


Figure 1.2- User enter the key to GA app- No communication between server and the GA app.



GA app keep creating codes using Shared Key and Time.

```
TOTP = HOTP(K, T),
where T is an integer
and represents the
number of time steps
between the initial counter
time T0 and the current Unix
time.K represent the shared
secret
```

Figure 1.3- Now Both sides have the shared key.

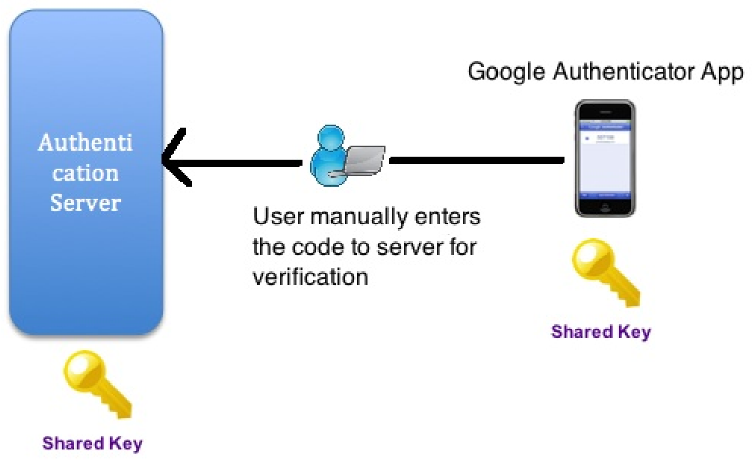


Figure 1.4 - User authentication using the TOTP.

***Note that the app and the server did not communicate by any means via network. Thus the possibility for leak the secret is minimized.**
Ok now we know what happens there and it is time to get into the code and implement our own two factor authentication. For this part we will develop a simple web application to demonstrate server side authentication part.

Using Eclipse Juno and Apache Tomcat for this sample development.

1. Create Dynamic Web Application Project.
#. Add index.jsp with simple login form in the body

```
  <%@ page language="java" contentType="text/html; charset=UTF-
8"
    pageEncoding="UTF-8"%>


*Note that checking the checkbox means its for setting up request Otherwise its
verification request.AuthController will be the servlet for handling the login
action.
```

2. Create Servlet - AuthController and override doPost() as follows.

```
package home.test.auth;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class AuthController
 */
@WebServlet("/AuthController")
public class AuthController extends HttpServlet {
 private static final long serialVersionUID = 1L;

 /**
  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
  */
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
  // TODO Auto-generated method stub
 }

 /**
  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
  */
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
  String username = request.getParameter("username");
  String password = request.getParameter("password");
  boolean is2faSetup  = true;

  if(request.getParameter("setup") == null)
  {
   is2faSetup = false;
  }

  if (username != null && password != null) {

   if(is2faSetup){
    // user want to set up 2fa
    request.setAttribute("username", username);
    request.getRequestDispatcher("/SetUpController").forward(request,
      response);
   }else{
   // forward to verify code
    request.setAttribute("username", username);
    request.getRequestDispatcher("/auth.jsp").forward(request,
      response);
   }

  } else {

   request.setAttribute("error", "Unknown user, please try again");
   request.getRequestDispatcher("/index.jsp").forward(request,
     response);
  }
 }

}
```

3. Create Class call GoogleAuthenticator.This is the Server side class for Google Authenticator's TOTP generated.Thanks to this excellent blog I can reuse most of the code from his blog post for this.

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Random;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base32;

/**
 * Java Server side class for Google Authenticator's TOTP generator
 * Thanks to Enrico's blog for the sample code:
 * @see http://thegreyblog.blogspot.com/2011/12/google-authenticator-using-it-in-
your.html
 *
 *
 * @see http://code.google.com/p/google-authenticator
 * @see http://tools.ietf.org/id/draft-mraihi-totp-timebased-06.txt
 */
public class GoogleAuthenticator {

 // taken from Google pam docs - we probably don't need to mess with these
 final static int secretSize = 10;
 final static int numOfScratchCodes = 5;
 final static int scratchCodeSize = 8;

 int window_size =3;  // default 3 - max 17 (from google docs)

 /**
  * set the windows size. This is an integer value representing the number of 30
second windows we allow
  * The bigger the window, the more tolerant of clock skew we are.
  *
  * @param s window size - must be >=1 and <=17.  Other values are ignored
  */
 public void setWindowSize(int s) {
  if( s >= 1 && s <= 17 )
   window_size = s;
 }

 /**
  * Generate a random secret key. This must be saved by the server and associated
with the
  * users account to verify the code displayed by Google Authenticator.
  * The user must register this secret on their device.
  * @return secret key
  */
 public static String generateSecretKey() {
  // Allocating the buffer
  byte[] buffer = new byte[secretSize + numOfScratchCodes* scratchCodeSize];

  // Filling the buffer with random numbers.
  // Notice: you want to reuse the same random generator
  // while generating larger random number sequences.
  new Random().nextBytes(buffer);

  // Getting the key and converting it to Base32
  Base32 codec = new Base32();
  byte[] secretKey = Arrays.copyOf(buffer, secretSize);
  byte[] bEncodedKey = codec.encode(secretKey);
  String encodedKey = new String(bEncodedKey);
  return encodedKey;
 }

 /**
  * Return a URL that generates and displays a QR barcode. The user scans this bar
code with the
  * Google Authenticator application on their smartphone to register the auth code.
They can also manually enter the
  * secret if desired
  *
  * @param user   user id (e.g. fflinstone)
  * @param host   host or system that the code is for (e.g. myapp.com)
  * @param secret the secret that was previously generated for this user
  * @return the URL for the QR code to scan
  */
 public static String getQRBarcodeURL(String user, String host, String secret) {
  String format = "https://chart.googleapis.com/chart?
chs=200x200&chld=M%%7C0&cht=qr&chl=otpauth://totp/%s@%s%%3Fsecret%%3D%s";
  return String.format(format, user, host, secret);
 }

 /**
  * Check the code entered by the user to see if it is valid
  * @param secret  The users secret.
  * @param code  The code displayed on the users device
  * @param t  The time in msec (System.currentTimeMillis() for example)
  * @return
  */
 public boolean check_code(String secret, long code, long timeMsec) {
  Base32 codec = new Base32();
  byte[] decodedKey = codec.decode(secret);

  // convert unix msec time into a 30 second "window"
  // this is per the TOTP spec (see the RFC for details)
  long t = (timeMsec / 1000L) / 30L;
  // Window is used to check codes generated in the near past.
  // You can use this value to tune how far you're willing to go.

  for (int i = -window_size; i <= window_size; ++i) {
   long hash;
   try {
    hash = verify_code(decodedKey, t + i);
   } catch (Exception e) {
    // Yes, this is bad form - but
    // the exceptions thrown would be rare and a static configuration problem
    e.printStackTrace();
    throw new RuntimeException(e.getMessage());
    //return false;
   }
   if (hash == code) {
    return true;
   }
  }
  // The validation code is invalid.
  return false;
 }


 private static int verify_code(byte[] key, long t)
   throws NoSuchAlgorithmException, InvalidKeyException {
  byte[] data = new byte[8];
  long value = t ;
  for (int i = 8; i-- > 0; value >>>= 8) {
   data[i] = (byte) value;
  }

  SecretKeySpec signKey = new SecretKeySpec(key, "HmacSHA1");
  SecretKeySpec signKey = new SecretKeySpec(key, "HmacSHA1");
  Mac mac = Mac.getInstance("HmacSHA1");
  mac.init(signKey);
  byte[] hash = mac.doFinal(data);

  int offset = hash[19] & 0xF;

  // We're using a long because Java hasn't got unsigned int.
  long truncatedHash = 0;
  for (int i = 0; i < 4; ++i) {
   truncatedHash <<= 8;
   // We are dealing with signed bytes:
   // we just keep the first byte.
   truncatedHash |= (hash[offset + i] & 0xFF);
  }

  truncatedHash &= 0x7FFFFFFF;
  truncatedHash %= 1000000;

  return (int) truncatedHash;
 }
}
```

***Note that according to the specification Secret key must be base32 encoded one so we are using org.apache.commons.codec.binary.Base32 thus you need to add the jar to WEB-INF/lib**

4.Create Servlet -SetUpController and override doPost() as follows.

```
package com.clault.auth;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.OutputStream;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import net.glxn.qrgen.QRCode;
import net.glxn.qrgen.image.ImageType;

import com.google.auth.GoogleAuthenticator;

/**
 * Servlet implementation class SetUpController
 */
@WebServlet("/SetUpController")
public class SetUpController extends HttpServlet {
 private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public SetUpController() {
        super();
        // TODO Auto-generated constructor stub
    }

 /**
  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
  */
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

 }

 /**
  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
  */
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

 String username = request.getParameter("username");

  String secretKey = GoogleAuthenticator.generateSecretKey();

  request.getSession().setAttribute( "secretKey", secretKey );

  String s = "otpauth://totp/"+username+"?secret="+secretKey;

  ByteArrayOutputStream outs = QRCode.from(s).to(ImageType.PNG).stream();

  response.setContentType("image/png");
  response.setContentLength(outs.size());

  OutputStream outStream = response.getOutputStream();

  outStream.write(outs.toByteArray());

  outStream.flush();
  outStream.close();

 }

}
```

***Note that above I have user QR code generator library form QRGen and Zebra Crossing (ZXing) all three jars should be put in to WEB-INF/lib**

5.Create a jsp call - auth.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Auth JSP</title>

</head>

<body>
  <form action="VerifyController" method="post">
 <table>
  <tr>
   <td>Enter Your Key</td>
   <td><input type="text" name="code"></td>
  </tr>
 </table>
 <input type="submit" value="submit">
 </form>

</body>
</html>
```

6. It submits the code for verification to VerifyController servlet let's create it.

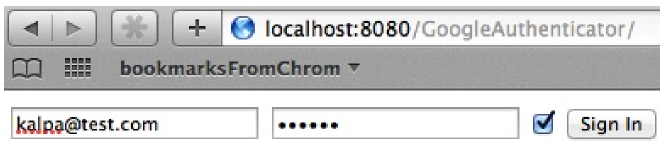
```
package com.clault.auth;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class VerifyController
 */
@WebServlet("/VerifyController")
public class VerifyController extends HttpServlet {
 private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public VerifyController() {
        super();
        // TODO Auto-generated constructor stub
    }

 /**
  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
```

```
    */
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // TODO Auto-generated method stub
  }

  /**
    * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
    */
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    String codestr = request.getParameter("code");
    long code = Long.parseLong(codestr);

    long t = System.currentTimeMillis();
    com.google.auth.GoogleAuthenticator ga = new com.google.auth.GoogleAuthenticator();
    ga.setWindowSize(5);  //should give 5 * 30 seconds of grace...

    String savedSecret = (String)request.getSession().getAttribute("secretKey");

    boolean r = ga.check_code(savedSecret, code, t);

    PrintWriter pw = response.getWriter();

    if(r){
    pw.write("OK");
    }else{
    pw.write("Fail");
    }

  }

}
```

Deploy the web application to tomcat.All done. It's time to test the application with Google Authenticator App.

7.1 When application deployed, go to http://localhost:8080/GoogleAuthenticator/ and enter some user name and password. And make sure the checkbox checked before send request. If you see the AuthController code it is the parameter to identify whether this is an account set up action or just verification login.

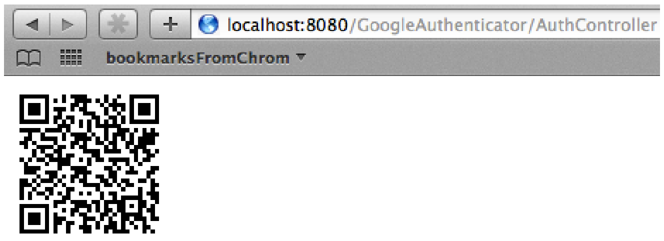
7.2 When you submit with the checkbox checked , the server will respond QR code which holds the Your user name SecretKey
(Shared Key)



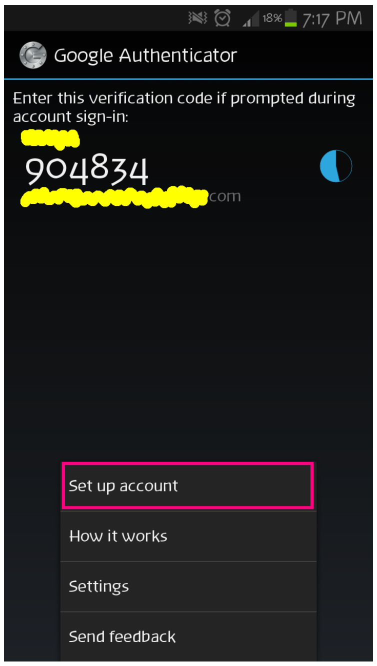
7.3 Now open up the Google Authenticator app on your phone. Select “Set up account” from the option menu.

7.4 Then select “Scan a barcode” from the option menu.

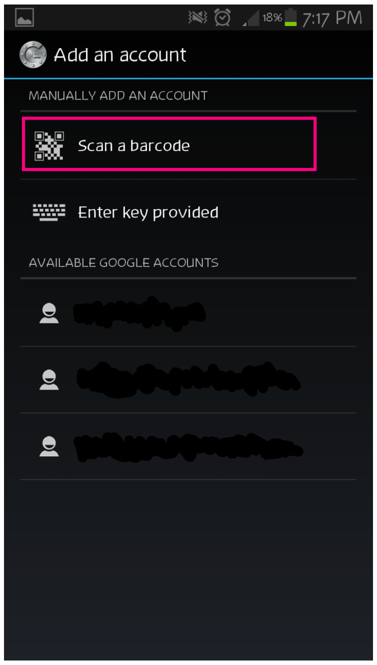
If you have installed QR code reader app GA will open up it for you otherwise you have to install a QR reader app.


localhost:8080/GoogleAuthenticator/AuthController

5. After successful scan GA will add a new account and generate a new 30 sec valid code for you.



Enter this verification code if prompted during account sign-in:
[redacted]
904834
[redacted]com



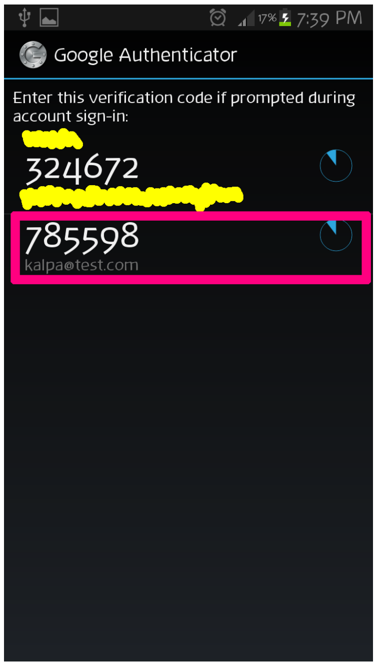
[redacted]
[redacted]
[redacted]

6. Now you can verify the code using a web application , Again go to http://localhost:8080/GoogleAuthenticator/ login without the checkbox checked. You will be redirected to code verification interface. Enter the code given by GA app.

The result will be
And check for fake number like
The result will be
Done. :)

Resources.
[1] - Eclipse Project file with all the dependencies can be found here.
References.

[1] - http://thegreyblog.blogspot.sg/2011/12/google-authenticator-using-it-in-your.html
[2] - https://code.google.com/p/google-authenticator/wiki/KeyUriFormat



Enter this verification code if prompted during account sign-in:

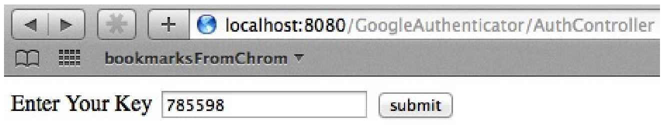
[redacted]

324672

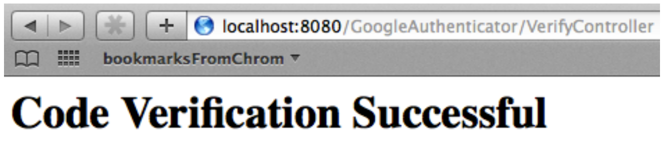
[redacted]

785598

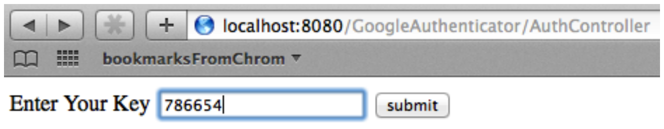
kalpa@test.com



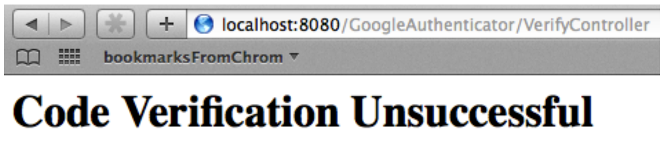
Enter Your Key 785598



Code Verification Successful



Enter Your Key 786654



Code Verification Unsuccessful