systemwerx

PassGen for Java

User Guide

SYSTEMWERX LIMITED

User Guide

Table of Contents

Introduction

PassGen is a toolkit to generate one-time passwords.

A One-Time Password is a password that is used only once. It can be used to authenticate users in a secure manner across insecure networks as once the password is used it can never be used again.

This technology is useful to allow:

- Users to authenticate across insecure networks in a secure manner. Facilities such as terminal sessions (3270, Telnet, FTP etc.), which use clear-text passwords, can be secured.
- Passwords to be generated for users without knowledge of a current password value. This is useful in environments such as portals where users need to be connected to other systems without prompting user for another password.

This release supports generating One Time Passwords using:

- The IBM Security Server (RACF) Passticket algorithm. IBM Servers using RACF, CA-Top Secret and CA-ACF/2 security systems support this algorithm.
- SKEY (RFC 1760)

What does this toolkit consist of?

This toolkit consists of:

- An application to request services using a REST interface, administration is performed using a browser.
- A user interface application that can be used to generate passwords. This is useful to try generating passwords quickly and easily. You can also manage the Keystore.
- A key storage facility (The Keystore) that stores your keys in an encrypted manner.
- Java Classes to generate passwords. You can write your own programs to access them.

With the interfaces provided you can securely store Secure Signon Keys and generate single-use IBM Security Server Passtickets in your applications. You can also generate S/Key one-time passwords.

How do One-Time Passwords work?

One-Time Passwords are passwords that are generated using a Cryptographic algorithm; they generate a unique password that will be accepted only once. The benefit of using these is that if passwords are sent across an insecure network they can be easily intercepted and used to gain unauthorised access to systems. If One-Time Passwords are sent across an insecure network they are useless to a potential attacker, as once they have been used they cannot be used again.

This current release supports the IBM Security Server (RACF) Passticket One-Time password algorithm, which was devised by IBM and operates on IBM Servers. The IETF One time password algorithm S/Key is also supported.

IBM Security Server (RACF) Passstickets

Security Server Passstickets can be used interchangeably with static passwords on any Server application that uses Security Server or compatible security system (CA-ACF/2 & CA-Top Secret)

PassGen and Security Server share a Secure Signon Key, which is an 8-byte set of characters that are used to generate the password. Unique Keys can be used for a single User, a single Application or for a Users access to a single Application. These keys must be kept secret and PassGen provides a Key Storage facility to help in this respect.

Users can authenticate themselves to applications without the concern that passwords are intercepted using Network and Application tracing software. If Passwords are intercepted they are useless to an attacker as they will have already been used and will be useless.

If Passticket support is enabled on Security Server, Passtickets can be used in any application that uses Security Server as a replacement for the password. If a user supplies a password that does not match their Static Password Security Server will check to see if the supplied password is a valid One-Time Password.

S/Key one time passwords

Users are prompted with a challenge when they connect to a system. It looks something like this: **23 aword**

The two values are input to the S/Key algorithm and the result is computed which is six words, something like this:

LET TOY HUED TILE LARK EMIT

The Keystore

To provide secure storage PassGen provides an encrypted databased called a Keystore. A Password is required to encrypt and decrypt data in this database.

REST web application

An application provides users immediate access to the toolkit with an application that can be used to store keys and generate One-Time Passwords. This application runs as a service providing a HTTP REST interface for applications to request credential generation.

User Interface application

An application provides users immediate access to the toolkit with an application that can be used to store keys and generate One-Time Passwords. Users can try this technology easily and quickly by generating a password and then using cut-and-paste to input it into an application.

Technical Support

Product support is provided on a 24x7 basis.

The preferred method of reporting problems is by using email to support@systemwerx.com

Getting started with PassGen

Pre-requisites

A suitable level of one of the IBM Server Security products:

- IBM Security Server (RACF) V1.9 or above
- Computer Associates CA-ACF/2 6.1 or above
- Computer Associates CA-Top Secret 5.0 or above

You need some administrative rights in these systems to do the required definitions.

For the S/Key support:

Host or firewall using SKEY (RFC 1760)

To execute this product a Java runtime is required at Java 8-17.

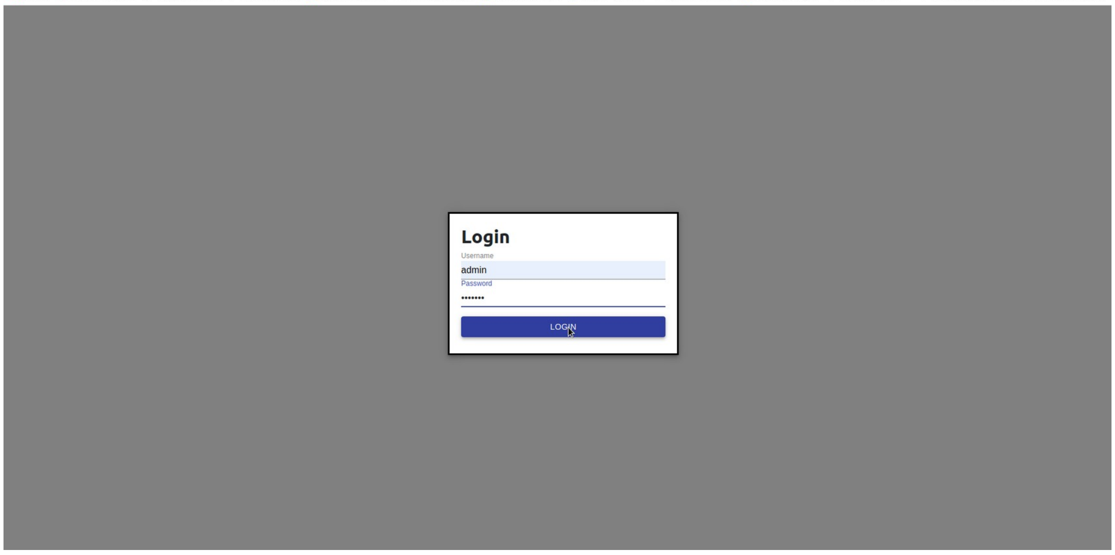
Starting PassGen REST application

The application is a standalone JAR and can be executed directly from the command line :

```
java -jar sxpassgen-webapp-7.jar
```

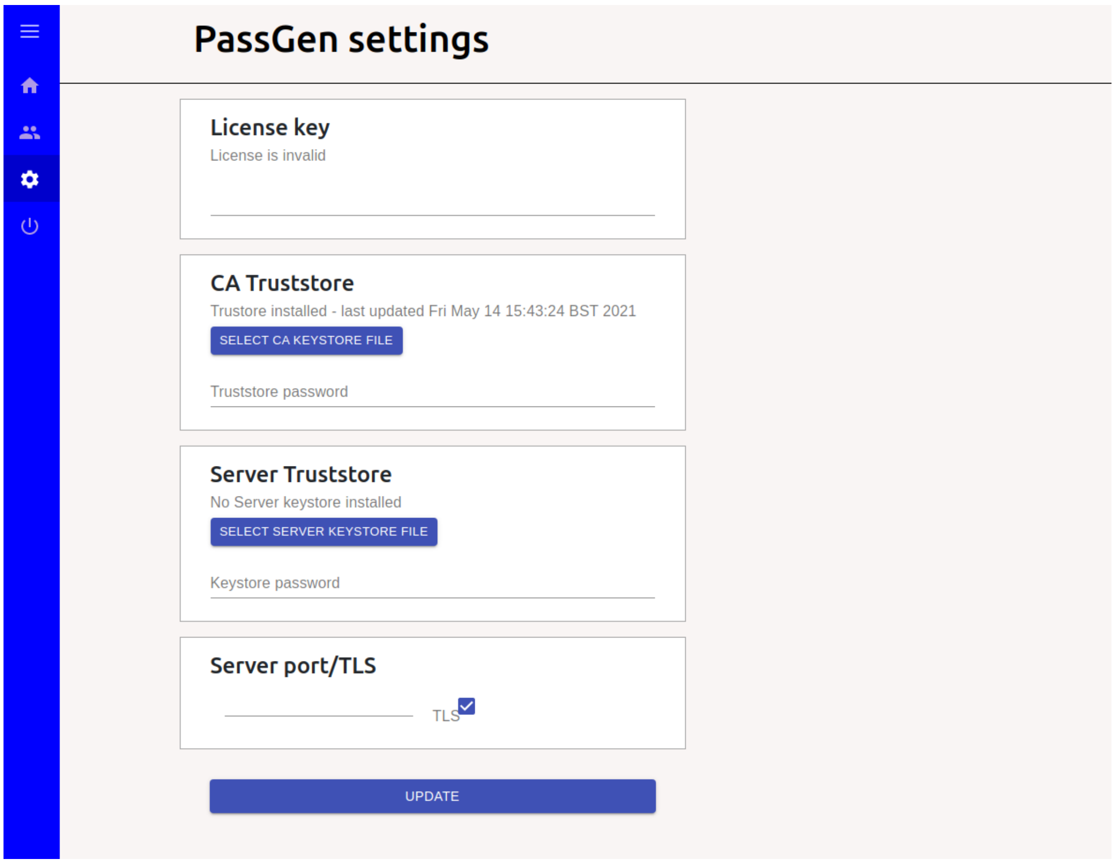
By default the application listens on port 8080 using HTTP. The application properties can be changed to use another port and HTTP/S when required.

Assuming the default configuration to illustrate the applications use – start the application and to to port 8080 on the executing host:



The default credentials are user admin password jwtpass.

To change the application settings select the configuration ⚙ icon on the side menu



PassGen settings

License key

License is invalid

CA Truststore

Trustore installed - last updated Fri May 14 15:43:24 BST 2021

SELECT CA KEYSTORE FILE

Truststore password

Server Truststore

No Server keystore installed

SELECT SERVER KEYSTORE FILE

Keystore password

Server port/TLS

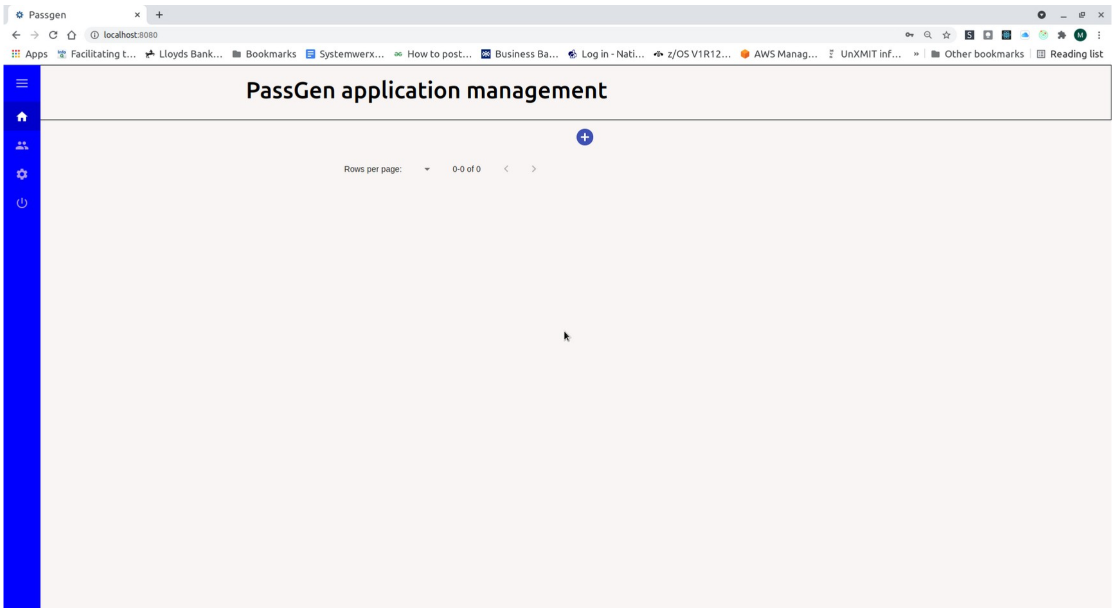
TLS ✓

UPDATE

Add the license key for the product. The application port, TLS protocol, server certificate and truststore can be changed by uploading new files. The format of these files is JRE JKS format.

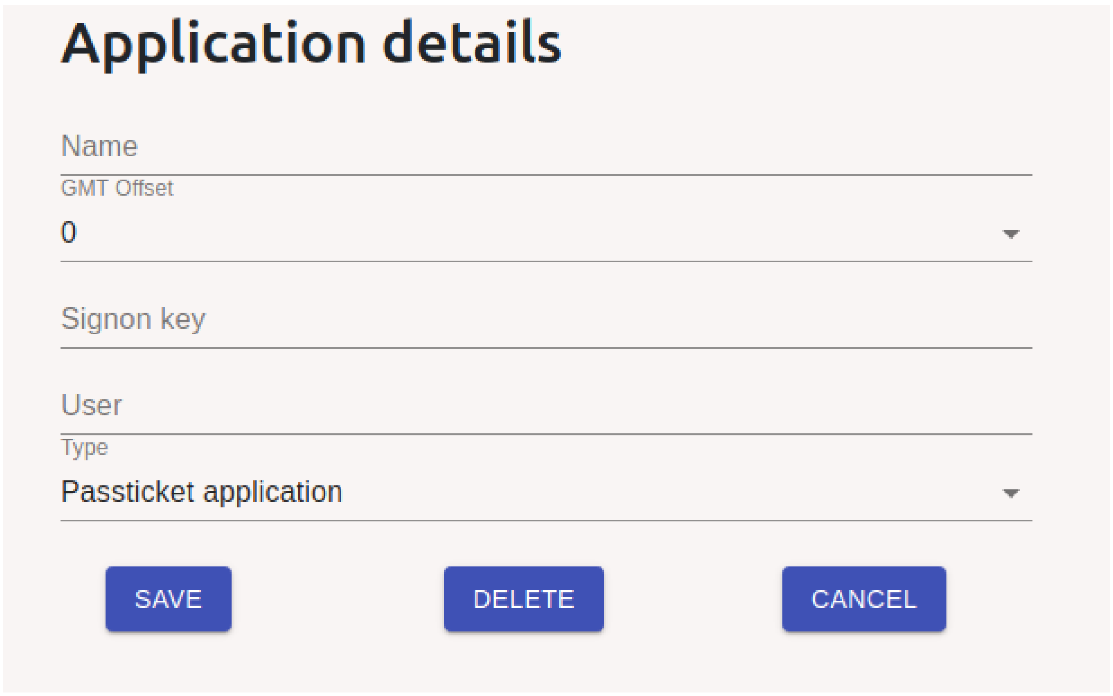
The application should be restarted after changes to settings.

Select the Home icon from the menu The user will be presented with the applications page



Select the add (plus sign) to add an application :

Applications details are presented in a pane on the right.



Application details

Name

GMT Offset: 0

Signon key

User

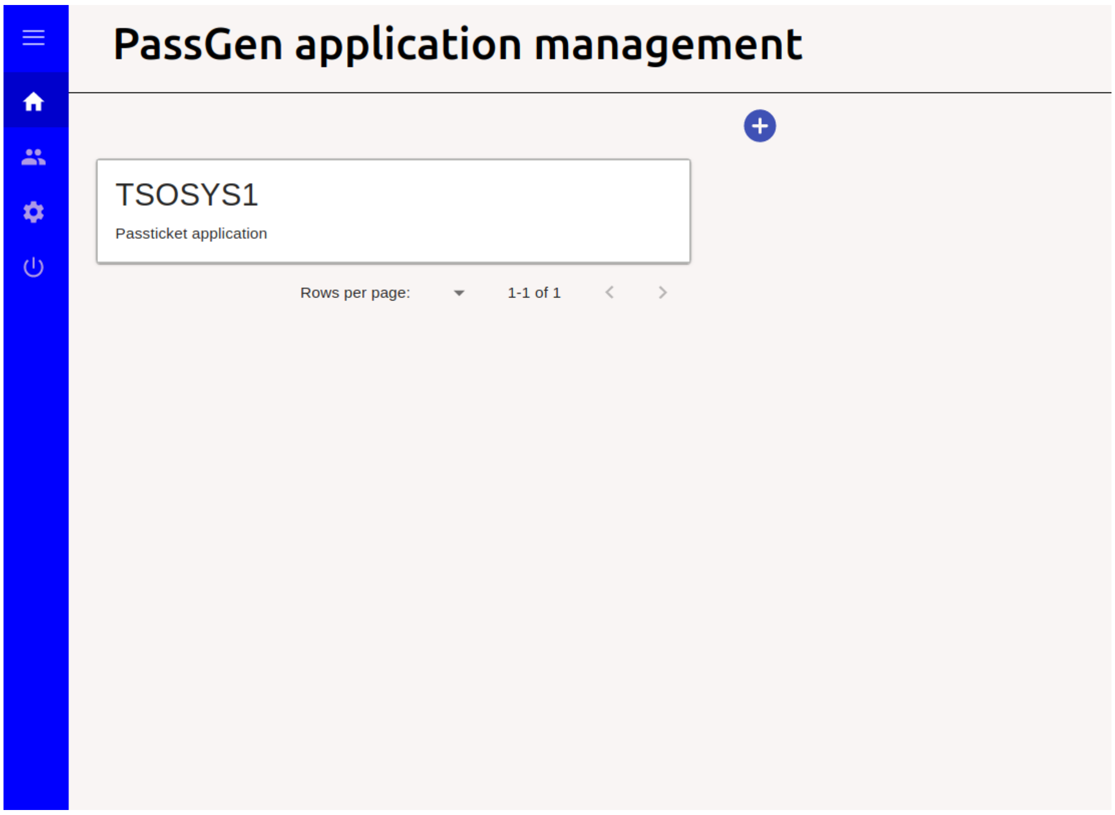
Type: Passticket application

Depending on the type of application selected the page changes to suit the requirements of the algorithm chosen :

FIELD NAME	USAGE
Name	This is the name of the application. For PassTicket Applications this must be the name of the application on the host system. For TSO the name is TSO plus the SMF ID of the system. For other applications this can be any name of your choosing
Secure Signon Key	Enter the secure signon key defined by the Security Administrator; this is a 16 character hexadecimal value.
Password	This is the password to be used with this application. If this is an SKEY application this is the key used to generate a one-time password.
User Name	This is the user account name to be used with this Application. This is optional for Password and Skey applications, but is required for PassTicket Applications.
Seed	This is the seed value for the Skey algorithm – this is supplied by your system administrator
GMT or Host Offset	This is the difference between the system time (TOD Clock) or GMT (depending on time zone) set and the time on the Windows System. To derive this value: 1. Execute the Diagnostic Option from the menu. Note the number of seconds displayed. 2. Execute the GETSEC utility on the Host (refer to the utility section)

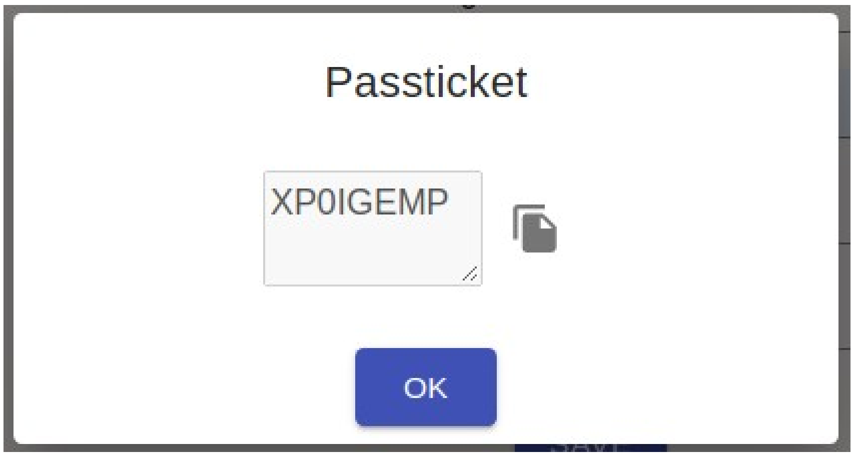
	3. Calculate the difference between the two values and divide by 3600. This is the offset between the system clocks.

Saved applications are listed – double click to generate a Passticket :


PassGen application management

TSOSYS1

Passticket application


Passticket

XP0IGEMP

Using the REST application in Docker/Podman

The application can be packaged in a Docker image file using the supplied Dockerfile :

```
podman build . --tag sxpassgenweb7
```

You can run the container using this command:

```
podman run --name sxpassgenweb -dt -p 8080:8080/tcp
sxpassgenweb7
```

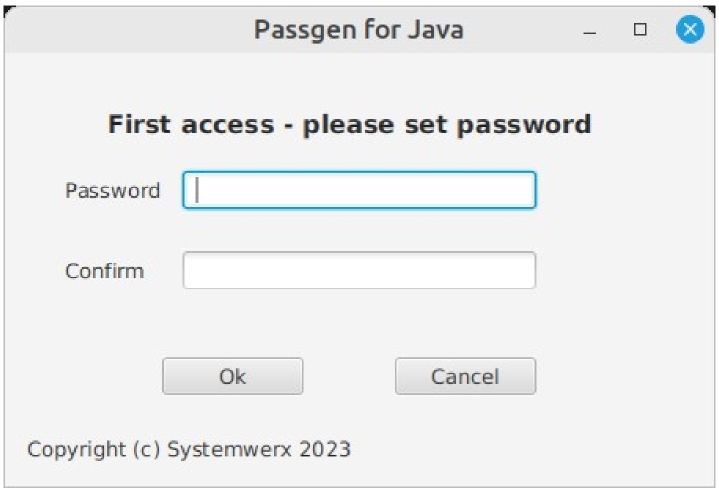
Starting PassGen user interface application

This application executes locally presenting an interface in the users native windowing system e.g. Windows, GNOME.

Under Windows : select the PassGen icon from the start bar to start the application.
Other platforms : Execute the PassGen script file or use the command :

```
java -jar PassGen.jar
```

On first execution you need to set the password to access PassGen. This password is used to encrypt the Secure Signon Keys.



Passgen for Java

First access - please set password

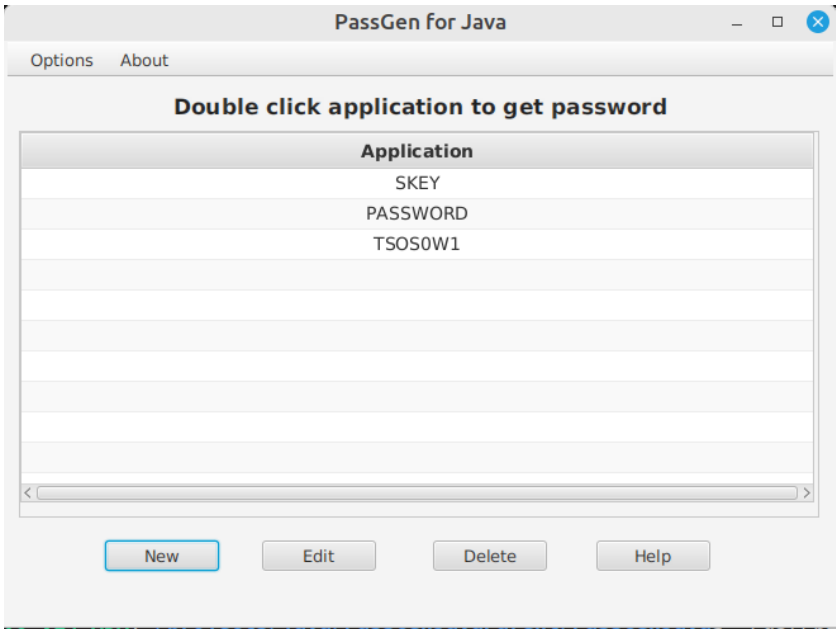
Password

Confirm

Copyright (c) Systemwerx 2023

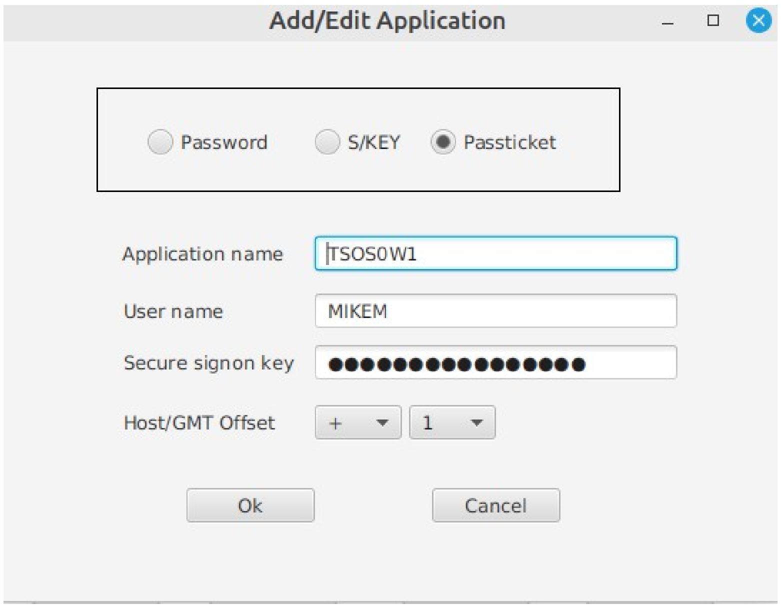
This password is used to access PassGen on subsequent logons.

When you have supplied a password you are presented with the application list:



Select New - you get a screen like this:

Add or edit a PassTicket



Password   S/KEY   [✓ Passticket]

Application name: TSOS0W1
User name: MIKEM
Secure signon key: ●●●●●●●●●●●●●●●●
Host/GMT Offset: + 1

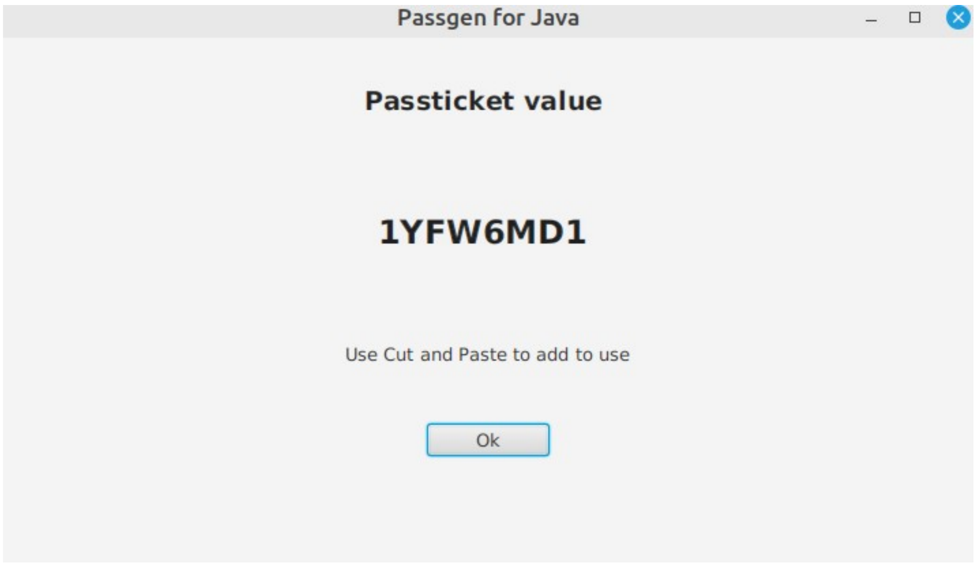
Supply the values required for each application:

FIELD NAME	USAGE
Application Name	This is the name of the application. For PassTicket Applications this must be the name of the application on the host system. For TSO the name is TSO plus the SMF ID of the system. For other applications this can be any name of your choosing
Secure Signon Key	Enter the secure signon key defined by the Security Administrator; this is a 16 character hexadecimal value.
Password	This is the password to be used with this application. If this is an SKEY application this is the key used to generate a one-time password.
User Name	This is the user account name to be used with this Application.

	This is optional for Password and Skey applications, but is required for PassTicket Applications.
Seed	This is the seed value for the Skey algorithm – this is supplied by your system administrator
GMT or Host Offset	This is the difference between the system time (TOD Clock) or GMT (depending on time zone) set and the time on the Windows System. To derive this value: 4. Execute the Diagnostic Option from the menu. Note the number of seconds displayed. 5. Execute the GETSEC utility on the Host (refer to the utility section) 6. Calculate the difference between the two values and divide by 3600. This is the offset between the system clocks.

Save the values by selecting the OK button. You are returned to the application screen.

Double-click the application to be presented with the PassTicket value.



Passgen for Java

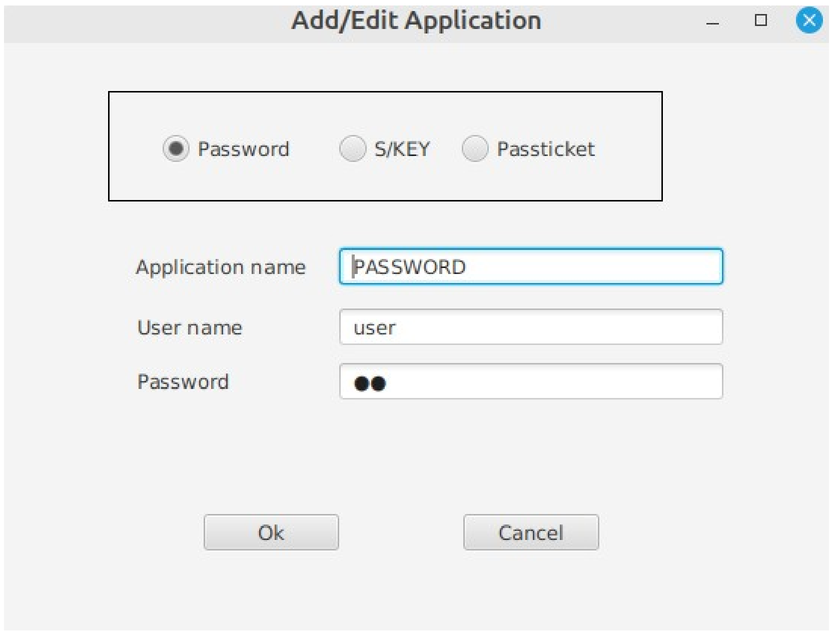
Passticket value

1YFW6MD1

Use Cut and Paste to add to use

Add or edit a Password

You can store static Passwords in PassGen. PassGen encrypts these in the PassGen Keystore for safekeeping.

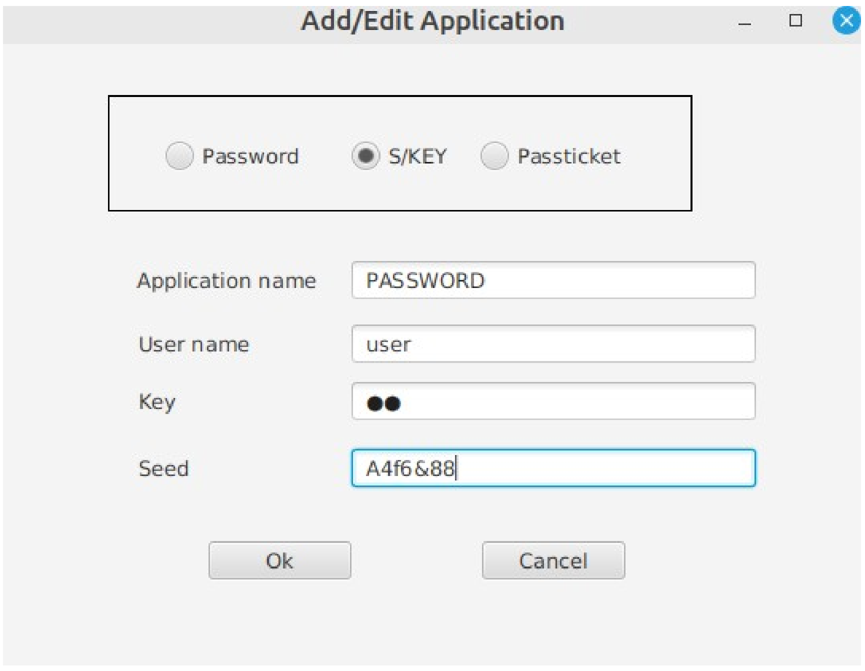


Add/Edit Application

[✓ Password] S/KEY Passticket

Application name	PASSWORD
User name	user
Password	●●

Add or edit an S/Key password


Add/Edit Application

Password [✓ S/KEY] Passticket

Application name: PASSWORD

User name: user

Key: ●●

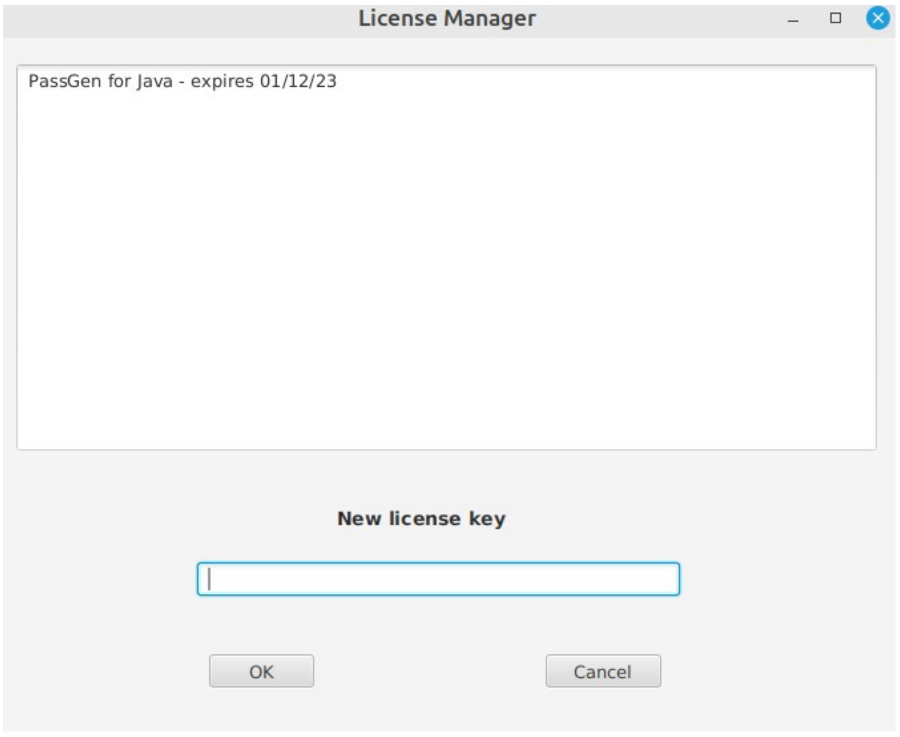
Seed: A4f6&88

License Manager

The license Manager, enables you to view and manage your license key(s). Keys are either permanent or temporary. Temporary keys are generally issued as a part of a trial and expire at a designated date.

From the Options => License Manager

To update your license key just cut and paste the key into field supplied and hit enter and if it's valid you should see an updated expires date



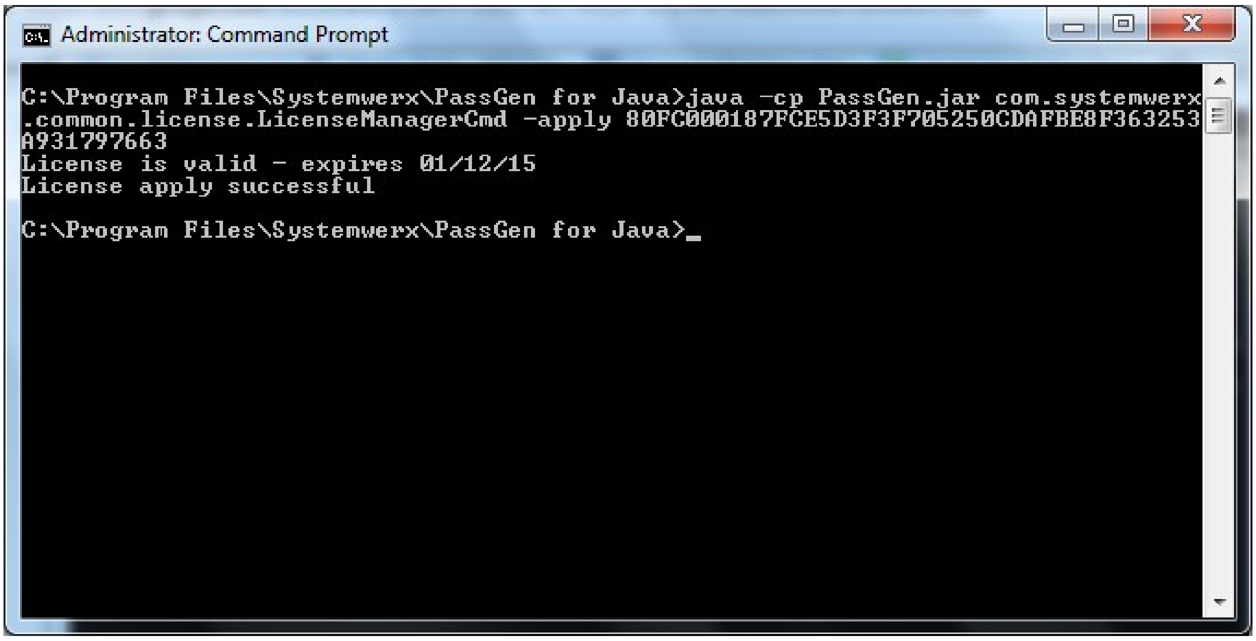
PassGen for Java - expires 01/12/23

Command line

If you do not have a display capable of a AWT display the license key can be applied in batch using the command :

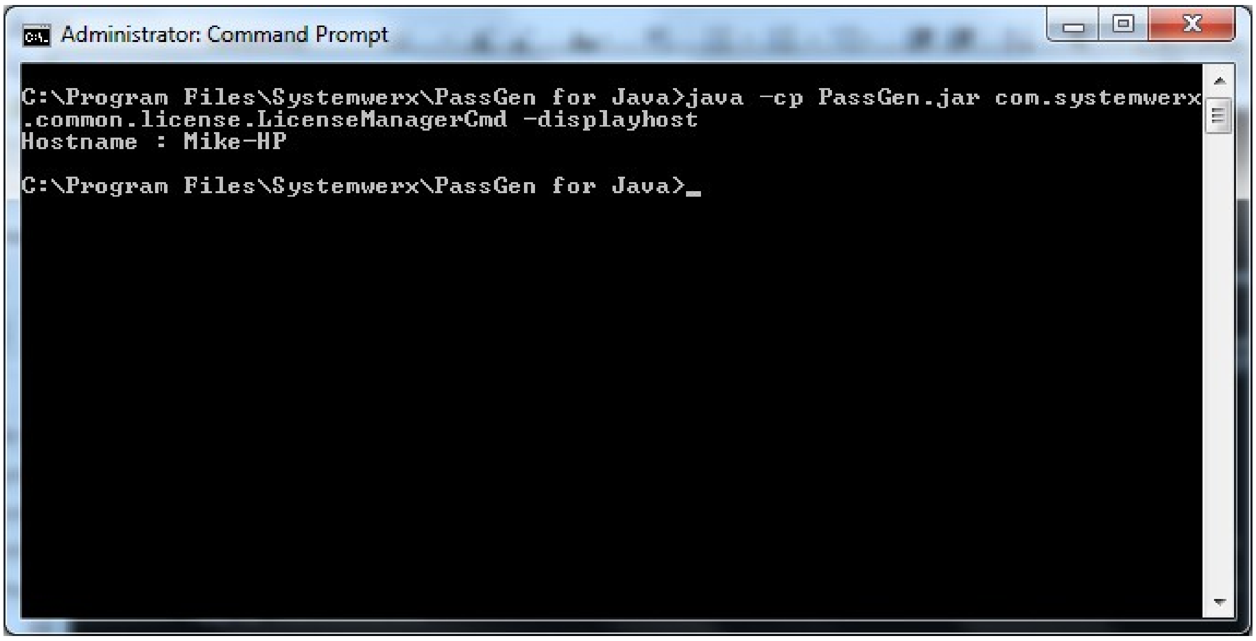
```
java -cp PassGen.jar com.systemwerx.common.license.LicenseManagerCmd –apply {license}
```

This will generate the license file in the current working directory :


Administrator: Command Prompt

```
C:\Program Files\Systemwerx\PassGen for Java>java -cp PassGen.jar com.systemwerx
.common.license.LicenseManagerCmd -apply 80FC000187FCE5D3F3F705250CDAFBE8F363253
A931797663
License is valid - expires 01/12/15
License apply successful

C:\Program Files\Systemwerx\PassGen for Java>_
```

If you wish to see the hostname of the system use the –displayhost command. This can be useful for diagnosing license issues.


Administrator: Command Prompt

```
C:\Program Files\Systemwerx\PassGen for Java>java -cp PassGen.jar com.systemwerx
.common.license.LicenseManagerCmd -displayhost
Hostname : Mike-HP

C:\Program Files\Systemwerx\PassGen for Java>_
```

Classpath stored license

A license file can also be stored in the classpath. If the sxlicense property is not set and PassGen finds no license file (license.dat) in the current working directory.

Programming Information

PassGen supplies classes to use its functionality:

Supplied Class	Function
PassTicketPure Bean	Generates Passtickets
sKeyBean	Generates S/Key passwords
KeyStoreBean	Provides access to the Keystor
KeyStoreEntry	Encapsulates an entry from the Keystore

The Java code in this product is supplied in one JAR file PassGenJava.jar, add this file to your CLASSPATH before using the Classes and Methods described below.

Access to license key file

Programs require access to the license key file to verify that the current license is valid. By default PassGen will look in the current working directory.

If you wish to supply another location this can be set in the JVM property **sxlicense**.

For example :

```
java -Dsxlicense=..\license.dat passTicketDemo1
```

Command Line

The command line interface allows users in scripting systems to access PassGen Functionality to :

- Generate PassTickets, S/Keys or get stored passwords
- Maintain application parameters
- Save password into a password file

Sample batch Scripts are located in the Batch folder within the Samples folder of the application installation.

Calling any command is using the Java runtime like this :

```
java -classpath PassGenJava.jar com.systemwerx.PassGen.SxPassGenCmd (command)
```

Where (command) is the command line command.

Generate a PassTicket

```
/GENERATEPTKT /APPLICATION:{applicationname}
              /TRANSTABLE:{translate table}
              /USERID:{userid}
              /SESSKEY:{session key}
              /KEYSTORE:{keystore file name}
              /KEYSTPWD:{password file name}
              /KEYSTPWDF:{ keystore password file}
              /OFFSET:{GMT or Host Offset}
              /RUNCOMMAND:{ Command to run once complete}
              /SETENVVARIABLES
```

If a Keystore is used the parameters are retrieved from there first and any values specified on the command line override them.

If Runcommand is used then once PassGen has successfully retrieved a passticket the Runcommand will be executed. Variables can be used to pass the passticket and username to the executed program. $passticket for the passticket and $userid for the username.

If SetEnvVariables is used then PassGen will set environment variables SXUSERID to the username and SXPASSTICKET to the passticket. These will only be accessible by applications started from the Runcommad parameter.

Generate an S/Key Password

```
/GENERATESKEY /APPLICATION:{applicationname}
                  /SEED:{seed}
                  /PASSWORD:{password}
                  /SEQUENCE:{sequence}
                  /KEYSTORE:{keystore file name}
                 /KEYSTPWD:{password file name}
                 /KEYSTPWDF:{ keystore password file}
                 /RUNCOMMAND:{ Command to run once complete}
                 /SETENVVARIABLES
```

If a Keystore is used the parameters are retrieved from there first and any values specified on the command line override them.

If Runcommad is used then once PassGen has successfully retrieved an S/Key the Runcommand will be executed. Variables can be used to pass the S/Key and username to the executed program. $skey for the S/Key and $userid for the username.

If SetEnvVariables is used then PassGen will set environment variables SXUSERID to the username and SXSKEY to the S/Key. These will only be accessible by applications started from the Runcommad parameter.

Get a Password

```
/GETPASSWORD /APPLICATION:{applicationname}
              /KEYSTORE:{keystore file name}
             /KEYSTPWD:{password file name}
             /KEYSTPWDF:{ keystore password file}
             /RUNCOMMAND:{ Command to run once complete}
             /SETENVVARIABLES
```

If a Keystore is used the parameters are retrieved from there first and any values specified on the command line override them.

If Runcommad is used then once PassGen has successfully retrieved a password the Runcommand will be executed. Variables can be used to pass the password and username to the executed program. $password for the password and $userid for the username.

If SetEnvVariables is used then PassGen will set environment variables SXUSERID to the username and SXPASSWORD to the password. These will only be accessible by applications started from the Runcommand parameter.

Save a Keystore Password to a password file

Keystore Passwords can be stored in an encrypted format for use by the Command Line or Programming interfaces. This allows passwords to be provided in environments where users cannot be prompted. The file must be protected against unauthorized access.

```
/SAVEPASSWORD /KEYSTORE:{keystore file name}
             /KEYSTPWDF:{password file name}
             /KEYSTWD:{password}
```

Create Application in Keystore

This adds an application to the Keystore.

```
/CREATEAPPLICATION /APPLICATION:{applicationname}
                /TYPE:{Application Type = SKEY or PTKT or PWD}
                /KEYSTORE:{keystore file name} KEYSTPWD:
{password}
                /KEYSTPWDF:{password file name}

                PassTicket Values:
                /SESSKEY:{session key}
                /OFFSET:{GMT or Host Offset}
                /USERID:{userid}

                S/Key Values:
                /SKEYPWD {S/Key Password}
                /SEQUENCE:{sequence}
                /SEED:{seed}

                Password Values:
                /PASSWORD:{password}
                /USERID:{userid}
```

Delete Application in Keystore

This deletes an application in the Keystore.

```
/DELETEAPPLICATION /APPLICATION:{applicationname}
                /KEYSTPWDF:{password file name}
                /KEYSTORE:{keystore file name} KEYSTPWD:
{password}
```

Display Applications in Keystore

This displays an application in the Keystore.

```
/DISPLAYAPPLICATIONS
                /KEYSTPWDF:{password file name}
                /KEYSTORE:{keystore file name} KEYSTPWD:
{password}
```

Modify Application in Keystore

This modifies an application in the Keystore.

```
/MODIFYAPPLICATION /APPLICATION:{applicationname}
                /KEYSTORE:{keystore file name} KEYSTPWD:
{password}
                /KEYSTPWDF:{password file name}

                PassTicket Values:
                /SESSKEY:{session key}
                /OFFSET:{GMT or Host Offset}
                /USERID:{userid}

                S/Key Values:
                /SKEYPWD {S/Key Password}
                /SEQUENCE:{sequence}
                /SEED:{seed}

                Password Values:
                /PASSWORD:{password}
                /USERID:{userid}
```

Parameters

PARAMETER	FUNCTION
APPLICATION	Set the Application name the PassTicket is to be generated for or Keystore update is to be made for

TYPE	Application Type :
	
	PTKT = Passticket
	S/KEY = S/KEY
	PWD = Password

USERID	Set the Userid
SESSKEY	Set the Session Key for this Application
OFFSET	Set the Offset between the Native time on the client computer and the GMT time or TOD clock on the RACF Host Generally this parameter is zero unless the TOD clock is not using GMT. Most systems use GMT in their TOD clock even in regions that are not in the GMT Time Zone. Use of the operator D T command can be used to confirm this in addition to the techniques documented in the manual
TRANSTABLE	Set the name of a ASCII-EBCDIC translation table. By default a US American Table is used
KEYSTPWDF	A Password file name that is used for Keystore access
KEYSTORE	The file name of the keystore
KEYSTPWD	Key store password
PASSWORD	S/Key or application Password
SEED	S/Key Seed
SEQUENCE	S/Key Sequence
RUNCOMMAND	Set command to run once a password, Skey or $passticket has been successfully generated. Variables can be used to pass the password and username to the executed program. $password for the password, $userid for the username, $skey for the S/Key and $passticket

SETENVVARIABLES	If SetEnvVariables is used then PassGen will set environment variables SXUSERID to the username, SXPASSWORD to the password, SXSKEY to the S/Key and SXPASSTICKET to the passticket. These will only be accessible by applications started from the Runcommad parameter.

Figure 1: Command Line Parameters

Sample Commands

Generate a PassTicket using Keystore data :

```
/generateptkt /application:tsosys1 /keystore:keystore
/keystpwdf:pswd.pwd
```

Create an application in the Keystore

```
/createapplication /application:tsosys1 /keystore:keystore
/keystpwdf:pswd.pwd /type:ptkt /userid:joe
/sesskey:0102030405060708 /offset:0
```

Generate a PassTicket using Keystore data and run command:

```
/generateptkt /application:tsosys1 /keystore:keystore
/keystpwd:test "/runcommand:echo $userid $passticket "
```

Get a password using Keystore data and run command:

```
/getpassword /application:FTPpassword /keystore:keystore
/keystpwd:test "/runcommand:echo $userid $password "
```

Get a password using Keystore with spaces in the Path

```
/getpassword /application:PsftpLocal "/KEYSTORE:C:\Users\
Chris\AppData\Roaming\Systemwerx PassGen\KeyStore"
/KEYSTPWD:test
```

Using Passgen from your program

You can imbed Passgen easily using the main Passgen classes

- PassTicketPureBean – Pure java Passticket generator
- Keystore – Keystore management and access The Keystore is an optional Key storage facility
- SkeyBean – generates S/Key passwords

passTicketPureBean class

This class generates Passtickets.

METHOD	FUNCTION
public void **setApplication**(java.lang.String In)	Set the application name the Passticket is generated for. The field is a maximum of 8 characters. This field is used by loadKeystoreParms() to determine which key entry to load.
public void **setDebugTime**(int DT)	This allows the time value being used to be overridden. Useful for debugging purposes.
public void **setUserID**(java.lang.String In)	Set the User name. This field is a maximum of 8 characters.
public void **setPasswordFile**(java.lang.String In)	Set the Password file name. This file contains an encrypted KeyStore password for use in applications where users do not wish to hard code passwords This can be used in conjunction with the loadKeystoreParms() method
public void **setKeystore**(java.lang.String In)	Set the Keystore file name. This is used in conjunction with the loadKeystoreParms() method.
public void **setSessionKey**(java.lang.String In) throws com.Systemwerx.PassGen.PassTicketExceptio n	Set the Session Key. This is specified as 16 Hexadecimal characters representing the 8 byte PTKTDATA profile in RACF. This value must be the same as coded in the User or Applications PTKTDATA profile.
public void **setGMTOffset**(int In)	Set the Offset between hosts. This field is rarely used. If you have a TOD clock not using GMT you can compensate by adjusting the time generated on this end of the connection. This value is added or subtracted (if a minus value) from the current time before generating the PassTicket. Values range from -12 through +12.
public void **setTranslateTable**(java.lang.String In)	Specify and ASCII-EBCDIC translate table from a file.
public void **setTranslateArray**(java.lang.String In) throws java.lang.Exception	Pass an ASCII-EBCDIC translate table from an array. The array is specified in Hexadecimal format. The table is 256 characters long and the function expects 512 Hexadecimal characters to be provided.

public boolean **loadKeystoreParms**() throws com.Systemwerx.PassGen.PassTicketException	Load parameters from a Keystore. This method opens the keystore specified by setKeystore() using the password in setPasswordFile(). The entry in the keystore specified in setApplication() is retrieved and parameters loaded into the bean.
public java.lang.String **getPassTicket**() throws com.Systemwerx.PassGen.PassTicketException, com.Systemwerx.Utils.e3UtilException	Generate a Passticket.

Sample code:

```
import com.Systemwerx.PassGen.*;
String Result;
passTicketPureBean pw=new passTicketPureBean();
pw.SetUserID("USERA");
pw.SetGMTOffset(0);
pw.SetTranslateTable("trans.txt");
pw.SetSessionKey("E001193519561977");
pw.SetApplication("TSOSYS1");
try
{

Result = pw.GetPassTicket();
System.out.println("Passticket is "+Result);
}
catch ( Exception e)
{
Class c = e.getClass();
String s = c.getName();
System.out.println("Exception >>>"+s+" "+e.getMessage());
}
```

keyStore class

The “keyStore” class provides the access to PassGen’s encrypted Key database. Use of the PassGen Keystore is not mandatory and users can elect to use their own storage method in place of this one:

METHOD	FUNCTION
public boolean **create**(int Type, java.lang.String FileName, java.lang.String Password) throws java.lang.Exception	Create a keystore. Parms : Type = Type of Password (External_Keyfile = Password file Key_Supplied = Password supplied; Internal_Key = Reserved for future use) FileName = Filename of KeyStore Password = Password or name of Password file, depending on Type
public boolean **open**(int Type, java.lang.String FileName, java.lang.String Password) throws java.lang.Exception	Open a keystore. Parms : Type = Type of Password (External_Keyfile = Password file Key_Supplied = Password supplied; Internal_Key = Reserved for future use) FileName = Filename of KeyStore Password = Password or name of Password file, depending on Type
public keyStoreEntry **getKey**(java.lang.String Value) throws java.lang.Exception	Get a keyStoreEntry from the keystore.
public boolean **replaceKey**(keyStoreEntry IEntry) throws java.lang.Exception	Replace a keyStoreEntry in the keystore.
public keyStoreEntry **getFirstKey**() throws java.lang.Exception	Get first Key entry in KeyStore
public keyStoreEntry **getNextKey**() throws java.lang.Exception	Get next entry in KeyStore. Use this with getFirstKey().
public boolean **deleteKey**(keyStoreEntry IEntry) throws java.lang.Exception	Delete entry in KeyStore.
public boolean **deleteKey**(java.lang.String App) throws java.lang.Exception	Delete entry in KeyStore.
public boolean **close**() throws java.lang.Exception	Close KeyStore.
public boolean **changeEncryptionKey**(java.lang.String IKey)	Change key of currently open keystore. IKey is new key

When opening the Keystore there are three methods for supplying the Key:

VALUE	USAGE
External_Keyfile	Key is supplied in a file – parameter is the file name
Key_Supplied	Key is supplied directly in parm
Internal_Key	Not supported in this release

Sample Code:

```
Keystore kb = new keyStoreBean();
kb.open(kb.Key_Supplied,"keystore","password");
if ( ( ke = kb.getKey("TSOSYS1")) == null)
{
      System.out.println("Cannot find application name");
}

passTicketPureBean pw=new passTicketpureBean();
pw.SetUserID(ke.getUserID());
pw.SetGMTOffset(ke.getGMTOffset());
pw.SetTranslateTable("trans.txt");
pw.SetSessionKey(ke.getSessionKey()");
pw.SetApplication("TSOSYS1");
kb.close();

try
{
  Result = pw.GetPassTicket();
  System.out.println("Passticket is "+Result);
}
catch ( Exception e)
{
  Class c = e.getClass();
  String s = c.getName();
  System.out.println("Exception >>>"+s+"  "+e.getMessage());
}
```

keyStoreEntry class

The “keyStoreEntry” class is used to get data to and from the Keystore. KeyStoreEntry instances are provided from the Keystore when get methods are used and passed to the Keystore when add/replace/delete methods are used.

METHOD	FUNCTION
public java.lang.String **getSeed**()	Get the S/Key seed
public java.lang.String **getSessionKey**()	Get Passticket Session Key
public java.lang.String **getUserID**()	Get the UserID.
public int **getGMTOffset**()	Replace a keyStoreEntry in the keystore.
public java.lang.String **getApplicationName**()	Get first Key entry in KeyStore
public void **setPasswordType**(int i) throws com.Systemwerx.PassGen.keyStoreEntryValueInvalid	Set the password type. This can be one of the following values : **passwordType_PassTicket** **passwordType_Password** **passwordType_SKEY**
public void **setSessionKey**(java.lang.String k) throws com.Systemwerx.PassGen.keyStoreEntryValueInvalid	Set the Passticket session key.
public void **setSeed**(java.lang.String k) throws com.Systemwerx.PassGen.keyStoreEntryValueInvalid	Set the S/Key seed
public void **setUserID**(java.lang.String u) throws com.Systemwerx.PassGen.keyStoreEntryValueInvalid	Set the UserID
public void **setApplicationName**(java.lang.String a) throws com.Systemwerx.PassGen.keyStoreEntryValueInvalid	Set the application name.

sKeyBean Class

The “sKeyBean” class is used to generate S/Key passwords :

METHOD	FUNCTION
public void **setSeed**(java.lang.String In)	Set the S/Key seed
public void **setHashAlgorithm**(int In) throws com.Systemwerx.PassGen.sKeyException	Set the Hashing Algorthm used. The values of MD4 or MD5 are valid.
public void **setPassword**(java.lang.String In)	Set the password
public java.lang.String **generatePassword**(int Seq) throws com.Systemwerx.PassGen.sKeyException	Generate the S/Key Password. Seq is the S/Key sequence number.
public boolean **loadKeystoreParms**() throws com.Systemwerx.PassGen.PassTicketException	Load parameters from a Keystore. This method opens the keystore specified by setKeystore() using the password in setPasswordFile(). The entry in the keystore specified in setApplication() is retrieved and parameters loaded into the bean.
public void **setApplication**(java.lang.String In)	Set the application name the Passticket is generated for. The field is a maximum of 8 characters. This field is used by loadKeystoreParms() to determine which key entry to load.
public void **setPasswordFile**(java.lang.String In)	Set the Password file name. This file contains an encrypted KeyStore password for use in applications where users do not wish to hard code passwords This can be used in conjunction with the loadKeystoreParms() method
public void **setKeystore**(java.lang.String In)	Set the Keystore file name. This is used in conjunction with the loadKeystoreParms() method.

Sample Code

```
     public static void main(String args[])
     {
       try
        {
       sKeyBean s = new sKeyBean();
       s.setSeed("1111");
       s.setPassword("2222");
       s.setHashAlgorithm(s.MD5);
       System.out.println(" MD5 Password is
"+s.generatePassword(23));
       s.setHashAlgorithm(s.MD4);
       System.out.println(" MD4 Password is
"+s.generatePassword(23));
       }
      catch ( Exception e)
       {
        Class c = e.getClass();
        String s = c.getName();
        System.out.println("Exception >>>"+s+"
"+e.getMessage());
       }
       return;
    }
```

Accessing the function from Java Server Pages and Servlets

Loading the software

The Java code is supplied in a single jar file - PassGenJava.jar.

This file must be accessible to the JSPs and Servlets of you application. This is done either by:

- Packaging PassGenJava.jar with your application (loaded using a Application scoped classloader)
- Placing PassGenJava.jar in a shared system area such as $TOMCAT_HOME\shared\lib under Tomcat.

Copy your JSPs into an application folder. Alternatively you can use the samples we provide in the Samples folder - files PassTicketPure.jsp and PassTicketPureGen.jsp

Users may find that saving their Keystore password into a file is useful and allows the values for PassGen to be stored in one place in a secure manner.

Sample code fragment:

```
  <!--
 Copyright (c) 2003 Systemwerx Ltd.

 All rights reserved.

 This is a sample JSP to generate a passticket
-->

<%@ page import = "com.Systemwerx.PassGen.*" %>

<jsp:useBean id="pw" class="com.Systemwerx.PassGen.passTicketPureBean" scope="session"/>


<html>
<head><title>Generated PassTicket</title></head>
<body bgcolor="white">
<font size=4>

<%
 try
 {
   String Value;
   int Error = 0;
   Value = request.getParameter("UserID");
   if ( Value.length() > 8 || Value.length() == 0 )
      {
        out.println("Please enter a valid userid<BR>");
        Error = 1;
      }

   pw.SetUserID(Value);
   Value = request.getParameter("AppName");

   if ( Value.length() > 8 || Value.length() == 0 )
      {
        out.println("Please enter a valid application name<BR>");
        Error = 1;
      }

   pw.SetApplication(Value);

   Value = request.getParameter("SessionKey");

   if ( Value.length() != 16 )
      {
        out.println("Please enter a valid Session Key<BR>");
        Error = 1;
      }

   pw.SetSessionKey(Value);

// Set the following two values

   pw.SetTranslateTable("e:\\projects\\e3PassTicket\\trans.txt");
   pw.SetGMTOffset(0);

   if ( Error == 0)
      {
       out.println("<BR><BR><H2>PassTicket value is "+pw.GetPassTicket());
       out.println("</H2>");
      }
   else
      {
       out.println("PassTicket generation failed - please select the browser back key and try again");
      }



  }
  catch (Exception e)
  {
      Class c = e.getClass();
      String s = c.getName();
      out.println("Exception >>>"+s+"  "+e.getMessage());
}
 %>



</font>
</body>
</html>
```

Using the REST API

The PassGen application can be accessed using REST.

The Passgen web application provides a REST API that is secured using OAUTH2 JWT. The application provides an authetication server to generate and renew JWT tokens required to access the service.

Requesting a token

A token request is made by making a GET request against the OAUTH token endpoint :

```
curl sxclientid:sxclientsecret@localhost:8080/oauth/token -d
grant_type=password -d username=(username) -d
password=(password)
```

The token returned is then provided in any REST call with the HTTP Authorization header in the format :

    Authorization:Bearer (token)

Requesting a Passticket

Once you have a token you can request a passticket using the URL :

```
curl
sxclientid:sxclientsecret@localhost:8080/application/credential/TSO
SYS1  -H "Authorization: Bearer (Token value)"
```

The returned data will look like this :

```
{

"message":"Credential generated",

"data":"{ "credential":"6OIFZ5UC", "type":"P"}","success":true

}
```

Saving your password in a password file

A password file contains a KeyStore password in an encrypted format.

In environments where you do not want to supply a password hard-coded in a program you can supply a password file. The methods of passTicketPureBean and sKeyBean:

- setPasswordFile()
- loadKeystoreParms()

Sample code Fragment:

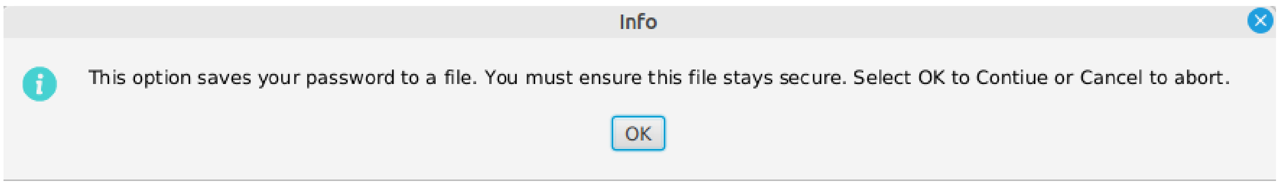
```
pw.setApplication("TSOSYS1");

pw.setPasswordFile("pswd");

pw.setKeystore("keystore");

pw.loadKeystoreParms();

Result = pw.getPassTicket();
```

This will load the data for the application **TSOSYS1** from the keystore **pswd** and generate a PassTicket

Controlling read access to this file controls access to this password so you must ensure this file is appropriately secure.

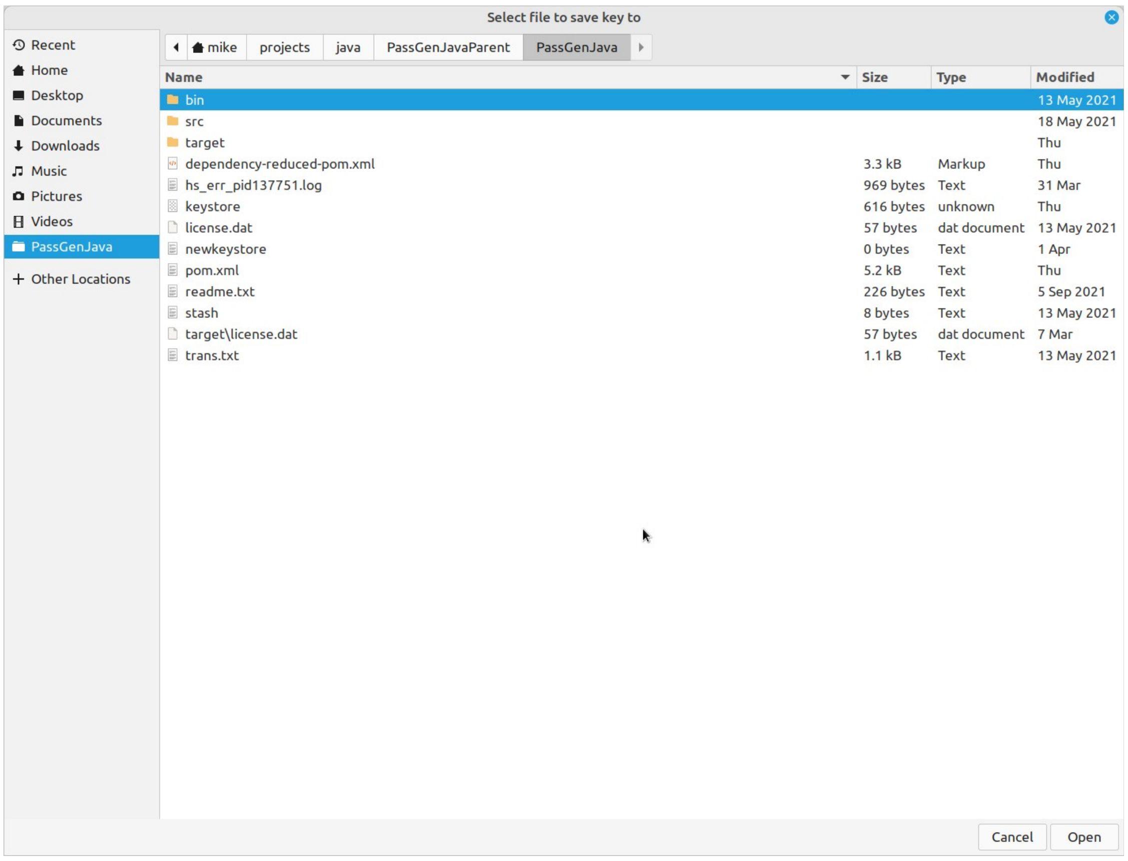
To create a password file select the “Store Passgen Password” option from the Options menu.

You are then given an option to continue select “OK”.



This option saves your password to a file. You must ensure this file stays secure. Select OK to Contiue or Cancel to abort.

You then select the file to save the password in:



Name	Size	Type	Modified
bin			13 May 2021
src			18 May 2021
target			Thu
dependency-reduced-pom.xml	3.3 kB	Markup	Thu
hs_err_pid137751.log	969 bytes	Text	31 Mar
keystore	616 bytes	unknown	Thu
license.dat	57 bytes	dat document	13 May 2021
newkeystore	0 bytes	Text	1 Apr
pom.xml	5.2 kB	Text	Thu
readme.txt	226 bytes	Text	5 Sep 2021
stash	8 bytes	Text	13 May 2021
target\license.dat	57 bytes	dat document	7 Mar
trans.txt	1.1 kB	Text	13 May 2021

Appendix

ASCII-EBCDIC Translation

The algorithm needs a translation table to translate characters to EBCDIC for PassTicket processing. For most normal requirements the standard Table “trans.txt” provides an adequate translation. Specify the translation table by:

- SetTranslateTable on the e3PassTicket class
- The parameter on the e3GenPassTicket function
- The parameter on the e3VBGenPassTicket function

If the supplied table is not adequate then the user can code an alternative, each row of the table represents a range of ASCII Hex characters 00-0F. The first row being bytes 00-0F, second row being 10-1F.

In the appropriate position the EBCDIC equivalent is coded:

```
#
# ASCII-to-EBCDIC table
# 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
#
  00 01 02 03 37 2D 2E 2F 16 05 25 0B 0C 0D 0E 0F     ; 00 ;
  10 11 12 13 3C 3D 32 26 18 19 3F 27 1C 1D 1E 1F     ; 10 ;
  40 5A 7F 7B 4A 6C 50 7D 4D 5D 5C 4E 6B 60 4B 61     ; 20 ;
  F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 7A 5E 4C 7E 6E 6F     ; 30 ;
  7C C1 C2 C3 C4 C5 C6 C7 C8 C9 D1 D2 D3 D4 D5 D6     ; 40 ;
  D7 D8 D9 E2 E3 E4 E5 E6 E7 E8 E9 B1 E0 BB BA 6D     ; 50 ;
```

The start of the third row (rows starting with # are comments and ignored) is ASCII Hex 20, which is a space, and its equivalent Hex character is 40, which is Blank in EBCDIC. Any characters after the 0F position in each row is treated as comments and ignored.

Determining the PTKTDATA profile name

This can sometimes be confusing. In most environments this is the VTAM Application name, CICSID or there is a user definable value.

In environments such as TSO and Batch this is not always clear.

In the case of TSO the name is built using the SMFID of the system, as a prefix to the word TSO, for example a system called SA12 would have a TSO PTKTDATA name of:

TSOSA12

In the case of BATCH the name used is equal to the SMFID of the system prefixed with MVS. On our system with an SMFID of SA12 the PTKTDATA profile name used for batch authentication is:

MVSSA12

In the case of the IMS Bridge and CICS Bridge, you should use an application name as if you were creating a passticket for a BATCH job, on which the target queue manager runs (Mqseries), as above.

IBM Security Server (RACF) definitions

To enable Passticket support you activate the PTKTDATA class in Security Server and add a RACF PTKTDATA profile to store a Secure Signon Key:

```
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
```

Activates the PTKTDATA class and:

```
RDEFINE PTKTDATA TSOSA12
SSIGNON(KEYMASKED(E001193519561977))
```

Defines a PTKTDATA profile name with a Secure Signon Key, the application defined here TSOSA12 is TSO on a system with an SMFID of SA12 then issue :

```
SETROPTS RACLIST(PTKTDATA) REFRESH
```

Refreshes the profile – please refer to Security Server (RACF) Security Administrators Guide for more information

Computer Associates CA-ACF/2 definitions

To enable Passticket support you store profiles in the ACF/2 PTKTDATA profile in the Infostorage database. Store a Secure Signon Key by issuing a command that looks like this:

```
SET PROFILE(PTKTDATA) DIVISION(SSIGNON)
INSERT TSOSA12 SSKEY(E001193519561977)
```

This defines a PTKTDATA profile name with a Secure Signon Key, the application defined here TSOSA12 is TSO on a system with an SMFID of SA12 then issue:

```
F ACF2,REBUILD(PTK),CLASS(P)
```

Rebuilds the records in storage. Please refer to the Administration Guide for more details.

Computer Associates CA-Top Secret definitions

To define a Passticket enabled application in CA-Top Secret define the Secure Signon key using a command similar to this:

```
TSS ADD(NDT) PSTKAPPL(TSOSA12) SESSKEY(E001193519561977)
```

This defines a PTKTDATA profile name with a Secure Signon Key, the application defined here TSOSA12 is TSO on a system with an SMFID of SA12.

Utilities

Two utilities are provided: GETSEC and E3PASST.

GETSEC

Getsec is a REXX program that displays the time represented as the number of seconds since 1st Jan 1970. GETSEC is supplied in the file getsec.txt in the Passgen directory.

Upload to a PDS on the host and execute. There is a display that looks like this:

```
1077057109 seconds since 01 January 1970 GMT
```

E3PASST

This utility uses the RACF callable Service to generate a PassTicket. It is used to diagnose problems in Host definitions.

The utility is supplied in the file e3passt.xmt in the PassGen directory and is in TSO TRANSMIT format.

Sample JCL:

```
//USERA JOB 111,'PASST',MSGCLASS=X,NOTIFY=&SYSUID
//* ----------------------------------------------------------
//S1        EXEC PGM=E3PASST,PARM='USERA   TSOSYS1 '
//*                              0123456789012345
//STEPLIB   DD DSN=USERA.LOADLIB,DISP=SHR
```

The parm field is 16 chars – first 8 is the userid, second 8 is the application name.

The program issues a WTO that looks like this:

```
e3PASST PassTicket = KAFRXPQK
```

Note: E3PASST is an authorised program AC(1). It needs to be in an APF library to run successfully. Otherwise you will get a S047 abend.

Problem Determination Procedure

Having problems making things work, here is a sample process to follow to isolate issues:

1. Run the getsec REXX program and the Windows diagnostic option and compare the number of seconds. There values should be within 600 of each other. If they are not then calculate the difference and divide by 3600 to get the offset between the systems, set this value on the PassGen application definition.

   This eliminates any clock issues

2. Run the E3PASST program to generate a PassTicket for your application.

   This eliminates if profiles are correctly installed (but not key value issues)

3. Log on to the application using the PassTicket generated by E3PASST

   This eliminates any issues with being re-routed to another application

4. Use the PassGen Dialog under Windows to generate a PassTicket and use this to log on to the application manually.

The eliminates PassGen software and that the Signon Key value is as expected

5. Attempt access from User Application