

# **APPLICATION DESIGN REPORT**

**I PRO 304**

**April 30, 2006**

## **-this is what we want to do**

- this is why we want to do it
- here are the other products
  - This is how they fail in what we want
- here is how we do what we want.

## **APPLICATION DESIGN GOALS**

Our goal is to design a way of securely recording and transporting a “patient’s” health records using information technology. Such a system is meant to replace medical information bracelets, medical information wallet cards, and the need to carry around other hardcopy medical information. Furthermore, our system was created with the following design goals:

- Create an application that allows multiple users, with the patient as the primary user.
- Allow several types of users with different levels of access to information.
- Make sure that not every user can edit information about the patient and so ensure levels of security.
- To comply with HIPPA, another goal is to have access logs indicating when users logged in and what information they changed, added or deleted.
- Have the ability to export information.
- Create an emergency page that doesn’t require a login to access vital patient information.
- Ensure that medications, vaccinations, allergies and illnesses could be added, deleted and updated.
- Have a method to ensure the authenticity of the user, e.g. making sure a doctor is a real doctor.
- It can interface into emerging electronic health record infrastructures.

These design goals are discussed in further detail throughout this document. We have explained why they are designed in such a way and also why we feel they link in a logical way. Finally, there is a user matrix to show which user can view and edit particular fields of the program.

## **HOW OUR SYSTEM IS DIFFERENT**

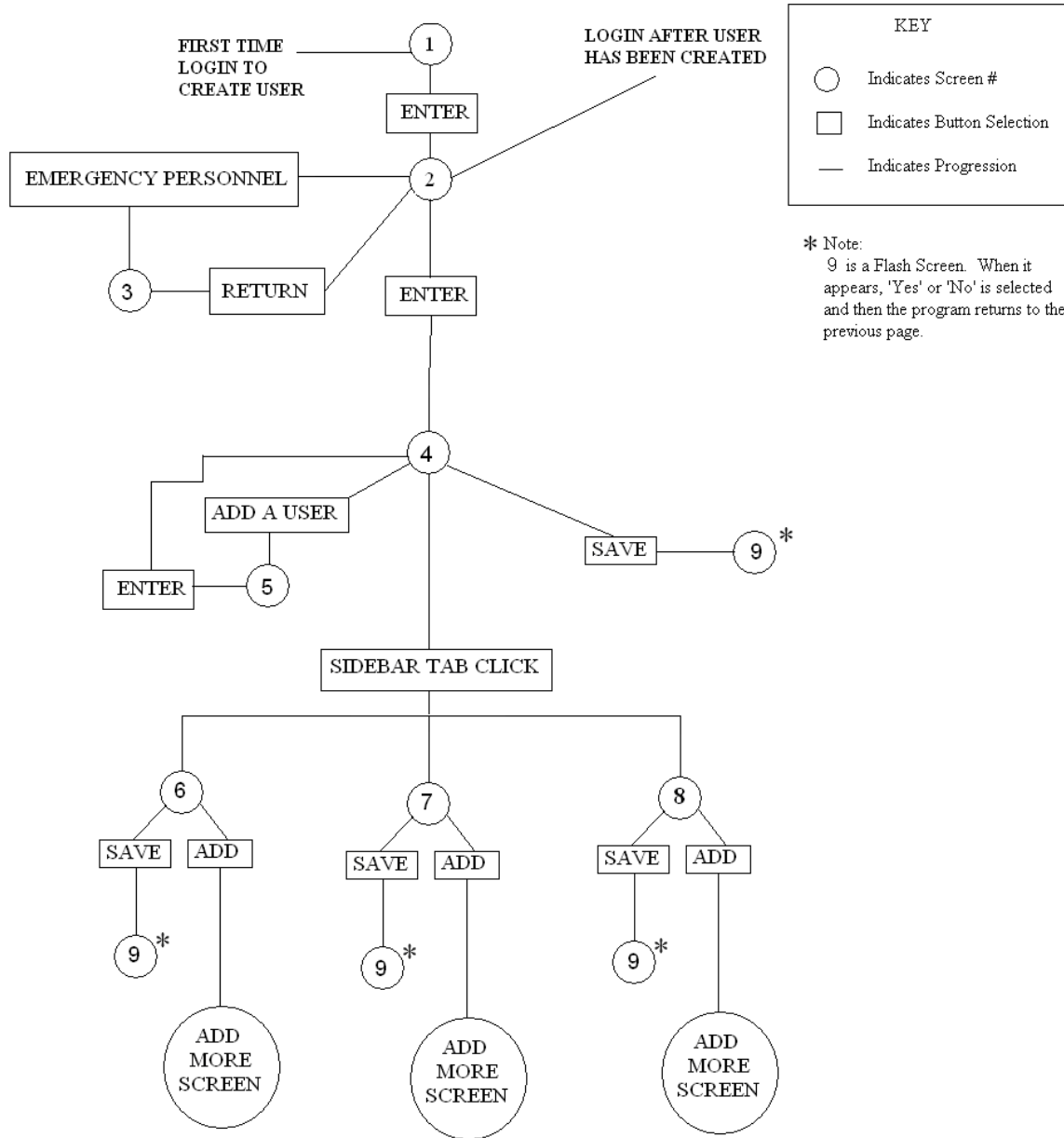
Besides supporting the patient, SafeByte also allows health-care professionals to store and sign health records. The following types of health-care professionals are currently supported:

**Table 1: User Access Level Matrix**

	<b>Patients</b>		<b>Doctors</b>		<b>Pharmacists</b>		<b>Emergency</b>
	<b>See</b>	<b>Edit</b>	<b>See</b>	<b>Edit</b>	<b>See</b>	<b>Edit</b>	<b>See</b>
<b>Personal Information</b>	<b>X</b>	<b>X</b>	<b>X</b>		<b>X</b>	<b>X</b>	<b>X</b>
<b>Allergies and Illnesses</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>
<b>Medications</b>	<b>X</b>		<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>
<b>Vaccinations</b>	<b>X</b>		<b>X</b>	<b>X</b>	<b>X</b>		

This is the User Matix. This matrix explains the different user access levels to be defined in the application. That is, it specifically exemplifies the information the different users (patients, doctors, pharmacists, and emergency) can see and the information they can edit. For example, the patient is allowed to see all the medical information, however can only edit the personal information and the allergies/ illnesses section. Similarly the emergency information can be accessed by the emergency personnel, however will not be able to edit any information. Every user will have his/her own password to log in and make the necessary changes and every change made will be logged in a separate file.

**Figure1. General Flow of the Application**



**Figure2. Welcome**

The screenshot shows a window titled "SafeByte - First Time Registration". The window has a standard Windows-style title bar with a close button on the right. The main content area is light gray and contains three labels with corresponding text input fields: "User Name :", "Password :", and "Confirm Password :". Below these fields is a "Register" button.

The welcome screen (figure 2) comes up when the user first plugs in his/her flash drive and runs the program. After a username is created, subsequent running of the program will go directly to screen 2. This screen allows one to create an account, with a username and password. The password is confirmed to make sure he/she has not accidentally typed in the wrong password. This will come up as asterisks on the screen so others sitting nearby cannot see the password. The “Register” button is clicked to go on to the next screen. A possible feature we could include is to have the password confirmation emailed to the user. Along with this, we could also include a “hint” option just in case the user forgets their password. However, both of these are still under the process of examination and approval.

**Figure3. Log in**

The image shows a software window titled "SafeByte - Log In". The window has a standard Windows-style title bar with a close button on the right. The main area of the window is light gray and contains the following elements:

- A label "User Name :" followed by a white rectangular input field.
- A label "Password :" followed by a white rectangular input field.
- A button labeled "Login" centered below the password field.
- A large, wide button labeled "Emergency Information" centered at the bottom of the window.

When the user plugs in the flash drive after the first time use, he/she will be prompted to this screen from the second time onwards. The first tab is “User Name,” where the user will be able to login in by typing in their user name created on the first screen. Then the user has to enter his/her user name and password. The “login” button allows the user to log in and proceed to the Main Personal Screen. The other button is “Emergency information.” This button allows access to the emergency information, as described in the matrix (see matrix). This is a large button so that it can’t be missed in an emergency.

**Figure4. Emergency**

**Emergency Information**

**Personal:**

**Name:**

**Date of Birth:**

**Age:**

**Gender:**

**Blood Type:**

**SSN:**

**Address:**

-----

**Medications:**

**Medication**

**Dosage:**

**Prescription:**

**Valid from**

**Notes:**

**Back**

This screen can be accessed from the first screen. In an emergency, it will be easy to access since it is just one button that needs to be clicked.

This screen will contain all the relevant emergency information. These relevant fields will appear below the name of each field as shown above for personal. It can be viewed by anyone, and that is why certain private fields are not shown. Based on the user matrix, we determined that personal, allergies and illnesses and medications were the important fields that need to be displayed on this screen. We felt that it was important to display all the important information on one screen so that it is simple to use in an emergency. None of this information can be edited however.

**Figure5. Personal Information**

The screenshot shows a web application window titled "SafeByte". The window has a menu bar with "File" and "Settings". On the left side, there is a vertical navigation menu with four items: "Personal" (which is highlighted in white), "Allergies and Illnesses", "Medications", and "Vaccinations". The main content area is titled "Personal" and contains several input fields: "Name", "DOB", "Age", "Gender" (a dropdown menu), "Blood Type", "SSN", and "Address". At the bottom right of the main content area, there are two buttons: "Save" and "Refresh".

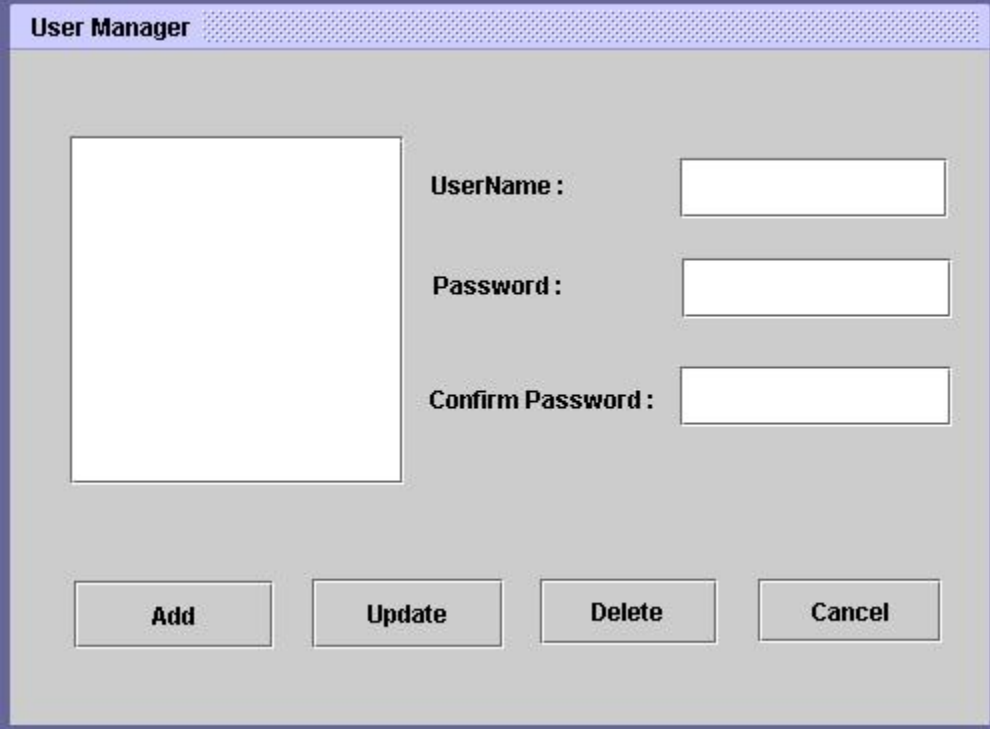
This page has two main components, the navigation column and the information column. We got the idea of having a navigation column from another competitive product (HealthFile™), and felt that it is very user friendly and will allow the user a much simpler way of accessing different screens. The navigation column includes the different pages or sub headings of the main page. This includes personal information, medications, vaccinations page...etc. The information that the designated user can change will be in white, and the information he/she does not have access to change will be grayed out. For example, if the user is a patient he/she will be allowed to change information such as his personal information, or allergy information. However, when he/she clicks on the medications he/she will not be allowed to make changes it to it.

The personal information section includes the name, date of birth, age, a drop down menu for gender, blood type, social security number, and address. The drop down menu for gender will decrease a chance for error and will allow for more accurate information. The "SAVE" button, will allow the patient to save any changes he/she makes.

Under "settings" which is a drop down menu at the top of the screen, there is an option to press the "User manager" button. This allows the patient to add different users and will result in them reaching the add user screen.



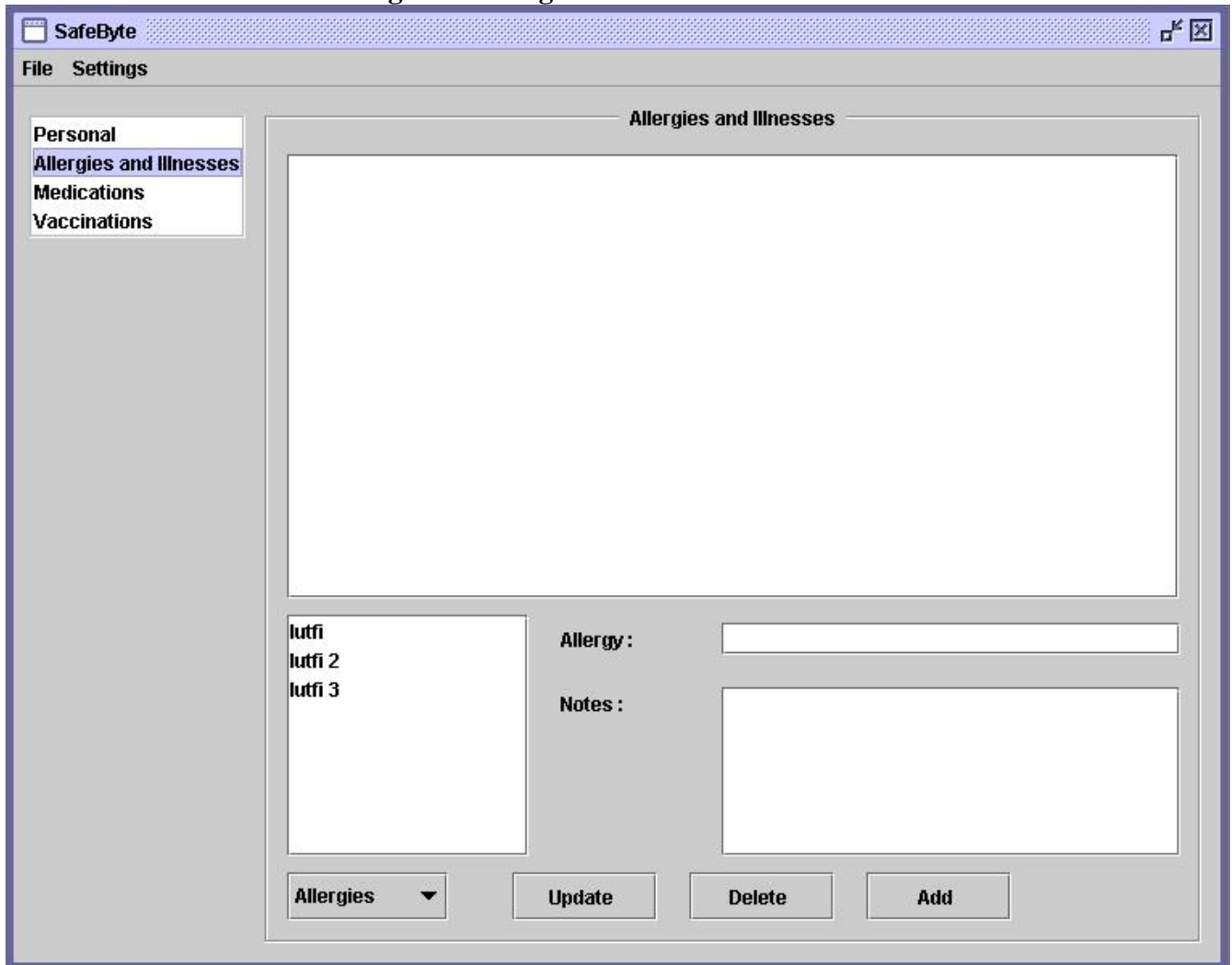
**Figure6. Add User**



The screenshot shows a window titled "User Manager" with a light blue header. The main area is grey and contains a large empty white box on the left. To the right of this box are three input fields: "UserName :", "Password :", and "Confirm Password :". Below these fields are four buttons: "Add", "Update", "Delete", and "Cancel".

The main aim of this section is to allow the patient to add other sub users, such as the doctors, pharmacists or insurance workers. Here they can enter a user name and password just like the patient did. Once they have done this, they should click the “add” button. This screen is still at a developmental stage since we are considering linking it to a website. This is to allow verification of the user’s identity, so that doctors or pharmacists are actually being added. At the moment however, different users names will appear in the column on the left, and can be selected if they wish to login. Then they can simply enter their password to gain their access rights. The “update” and “delete” button allow simple updating and deleting options for the user. Clicking “cancel” will take the user back to the personal page.

**Figure7. Allergies and Illnesses**



The allergies tab allows the patient and the doctor to manipulate the patient's allergy and illness information. In the bottom left hand corner is where a list of already added illnesses and allergies will appear. If one of these is selected, the full description will appear in the large central box. The bottom right corner of the screen is where new allergies or illnesses can be added. By clicking the button "allergies" the user can select whether to add an allergy or illness. Then they will be able to type in the name and description in the bottom right corner of the screen. To save this addition, they should click on "add". The user also has the option to delete or update current records on allergies and illnesses that are present in the list on the left. This can be done by clicking "update" or "delete". Before the changes take place, they will be prompted with the confirmation screen (9). We designed this screen and the rest in this way because we feel that it is easy to use. Also if they all look similar, then things will be simple and not confusing.

**Figure8. Medications**

	Dates of Use		Prescription Name	Dosage
	From	To		
lutfi	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
lutfi 2	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
lutfi 3	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Notes :

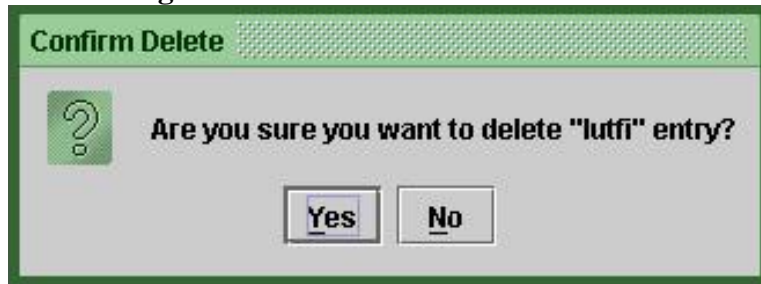
Just like the previous screen, the medications form can be accessed from the tab on the left hand side of the screen. Only the doctor and pharmacist can edit the information on this screen. They can input the dates of use for a particular drug, along with its name and dosage requirements for the patient. The final column allows them to add any other notes relevant to the medication. We felt that all these fields were necessary on a medications page. Once again, a user can click “update” to save their work, “delete” to remove items and “add” to add more drugs to the list.

**Figure9. Vaccinations**

The screenshot shows a software window titled "SafeByte" with a menu bar containing "File" and "Settings". On the left side, there is a sidebar with four menu items: "Personal", "Allergies and Illnesses", "Medications", and "Vaccinations", with "Vaccinations" selected. The main content area is titled "Vaccinations" and features a large empty text box at the top. Below this, there are three input fields labeled "Date", "Doctor", and "Vaccination Shot Given". A "Notes:" label is positioned above a larger text area. At the bottom of the form, there are three buttons: "Update", "Delete", and "Add". On the left side of the form area, there is a list of names: "lutfi", "lutfi 2", and "lutfi 3".

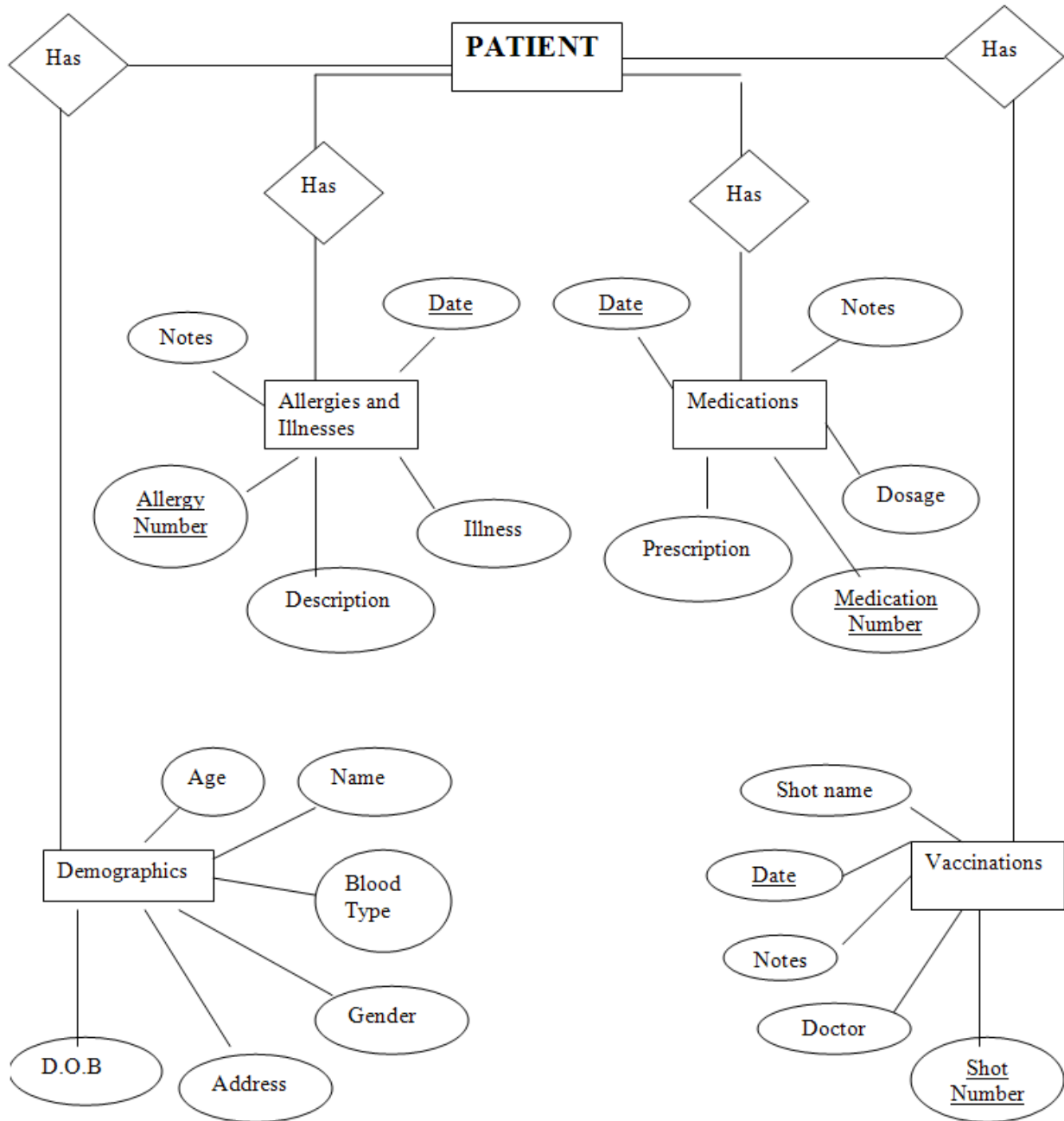
This is the final screen of the four main screens listed on the side tab. On this page, it is possible to add information regarding vaccinations taken so that the doctor can see what has already been taken and what may need to be taken. There are four fields relevant to vaccinations that we felt were necessary. First of all the date of the vaccination is vitally important, along with the name of the vaccination. It can also be important to know who gave the vaccination or where it was taken. Finally, any other additional information about the vaccinations can be added in the notes section. To add more vaccinations to the list, the user can click on “add”. To save the work, the user should click on “update” and to remove items they can click on “delete”.

**Figure10. Confirm Delete/Save Screen**



This screen will appear every time that a user wishes to either save or delete an entry. The “confirm save” screen will look identical to this, only with different text relating to save. This screen has been added to confirm that the user wants to save or delete the current work. We feel that this is important because the user may press save or delete by accident and perhaps would not have liked to save or delete the work. This just gives them the chance to confirm. By pressing “Yes” the changes should be saved and the user should return to last screen. By pressing “No” the new information should not be saved and the user should return to the previous screen. This works in similar way with deleting items.

Figure11. ER Diagram



## REFERENCES

Health File Software: <http://www.wakefieldsoft.com/healthfile/screenshots1.html>

Capmed Software: <http://www.capmed.com/phrpresentation/>

Contents of health records: [http://www.myphr.com/your\\_record/record\\_contents.asp](http://www.myphr.com/your_record/record_contents.asp)

# **SafeByte Installation & Application User Manual**

**I PRO 304**

*April 30, 2006*



## Contents

<b>Welcome and Introduction to Safebyte</b>	<b>3</b>
<b>Installing Safebyte</b>	<b>4</b>
<b>Runnung Safebyte</b>	<b>5</b>
<b>Using Safebyte's Security Tools</b>	<b>5</b>
<b>Disabling Auto-run</b>	<b>6</b>
<b>Automatic Virus Scan Prompt</b>	<b>8</b>
<b>Authenticating the SafeByte Medical Application</b>	<b>8</b>
<b>Application Walk Through with Flowchart</b>	<b>11</b>
Flowchart	12
First Time Registration	13
Login	14
Emergency	15
Personal	16
Add User	18
Allergies and Illnesses	19
Medications	20
Vaccinations	22
<b>Citations</b>	<b>24</b>

## **Welcome to SafeByte!**

SafeByte is an innovative combination of security and portability that is designed to reorganize the medical world. All medical records can be saved electronically and distributed through flash drives. SafeByte is designed with the intention of multiple users including patients, doctors, pharmacists, and EMS personnel. Each user is granted different levels of access to the medical data according to government regulation and medical standards.

### **Introduction**

Flash drives are the easiest, most cost-effective, and transportable removable storage devices on the market. Flash drives are ideal at school, in the office, and at home because of their ability to transfer a relatively large amount of data. Using USB technology, which is widely available on nearly every computer, flash drives work as a plug-and-play device, therefore being the simplest removable storage device yet. Flash drives are extremely durable. Most flash drives can be rewritten up to 100,000 times. (Flash Drive Alliance) Flash drives are the obvious choice for the field of portable health records.

There are concerns expressed about the ability to trust a flash drive that is brought into a corporate setting. Flash drives could pose a virus threat by a malicious or even an unknowing user. A user could bring an infected flash drive and connect it up to a corporate computer. The user could infect the host network by their own actions, such as executing a file or copying over files on a computer.

Another major concern of a flash drive as a virus threat would be through the auto-run capabilities of Windows when the device is connected. The potentially malicious files would execute on their own, without computer or user initialization. The auto-run feature could circumvent any virus scan activities. Understanding the auto-run options available for USB devices is critical to controlling them. Flash drives could also emulate other drive types in order to auto-run their content on a computer.

## Installing SafeByte

### Prerequisites for Installing SafeByte:

1. SafeByte security tools require that the user have a Java Runtime Environment. Before Installation begins, the user should verify that the computer has an updated version of the Java Runtime Environment. To do so, one must go to: <http://java.sun.com/downloads/> to download the latest compatible Java SDK.

### Installation Procedure:

1. After verifying the Java Runtime Environment, the first step for SafeByte's Installation is to connect the SafeByte flash drive to the computer USB Port.
2. Next, open 'My Computer' (an icon for 'My Computer' can be found on the computer desktop or under the 'Start Menu'). Once 'My Computer' is opened, locate the flash drive icon in the left hand side and double-click on it.
3. The final step to begin the installation process is to double-click on the 'Setup.bat' file located on the flash drive. The application installation window will be displayed (Figure 1, below). Follow the on-screen instructions. At the completion of the installation, the user will be required to restart the computer.

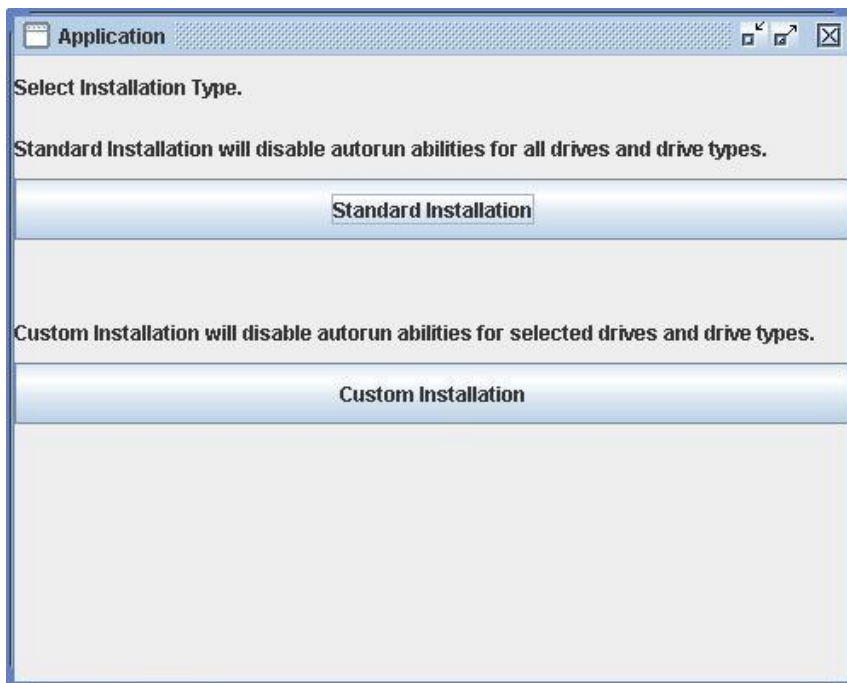


Figure 1: *Installing SafeByte*

## Running SafeByte

Running SafeByte begins by double clicking the SafeByte.bat file on the SafeByte flash drive or running SafeByte from the SafeByte folder on the program files folder after installation (Figure 2).

To run all the necessary security checks, (Auto-run disable, Virus scan prompt, and Application Authentication) SafeByte must be installed on the user's computer. Provided that SafeByte is installed properly, SafeByte automatically goes through the necessary security checks when the flash drive is inserted. The program will check for the correct USB flash drive by identifying the contents. The program will also recommend that the user perform a virus scan on the drive using third party software. Finally, the program will check to see if the contents have been tampered with by verifying the SafeByte medical application's digital fingerprint.

Running SafeByte without installation will allow the SafeByte Medical Application run without going through the recommended security checks.

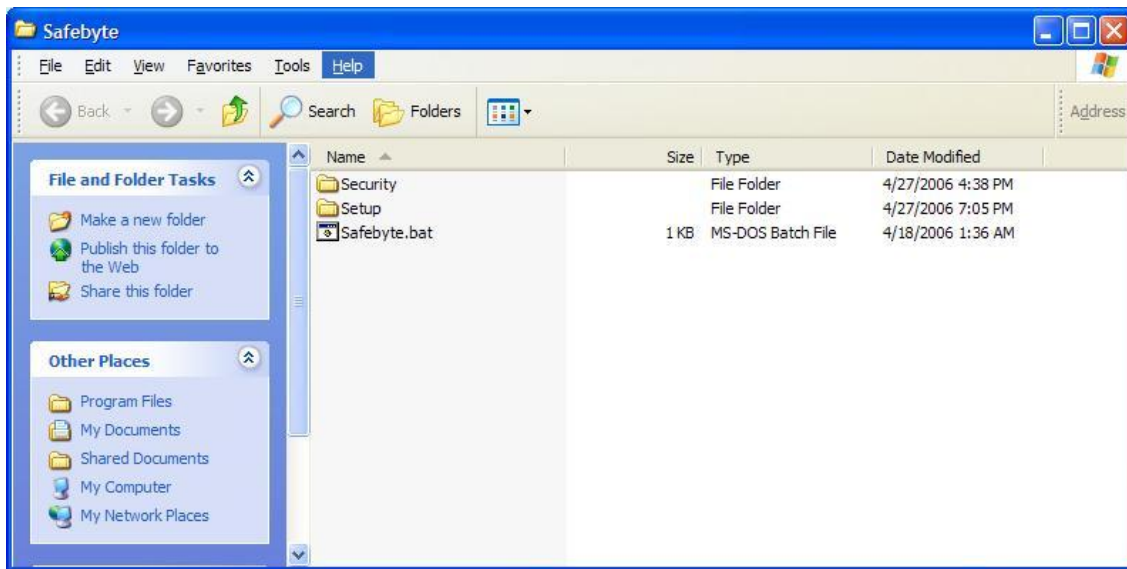


Figure2: *Running SafeByte*

## Using SafeByte's Security Tools

Installing SafeByte can ensure that the user's computer will not be attacked by the SafeByte USB flash drive, SafeByte's security features are:

1. Disabling the auto-run feature in Windows. (PC911): Disabling the auto-run feature not only affects USB flash drives, but also DVDs, CDs, floppy disks, and various other information media.
2. Virus Prevention: SafeByte provides preventive measures against onboard triggered flash drive attacks, by automatic virus scan requests on inserted USB drives.

3. **SafeByte Authentication:** SafeByte is able to authenticate the SafeByte application itself by collecting and comparing the application’s digital “fingerprint” with its authentic fingerprint.

These are all automatically loaded when a USB flash drive is inserted into the computer. Instructions on these tools, as well as how to change their settings are given below:

### **Disabling Auto Run:**

SafeByte allows the user to disable the auto-run feature during installation. Once the setup.bat file is run, a window pops up offering two options for disabling auto run:

1. The standard installation, which disables auto-run for all drive types & letters
2. The custom installation option, which disables auto-run for select drives. The choice of drives can be selected from the choice of checkboxes on the screen.
3. Click the ok button after making the proper selection.

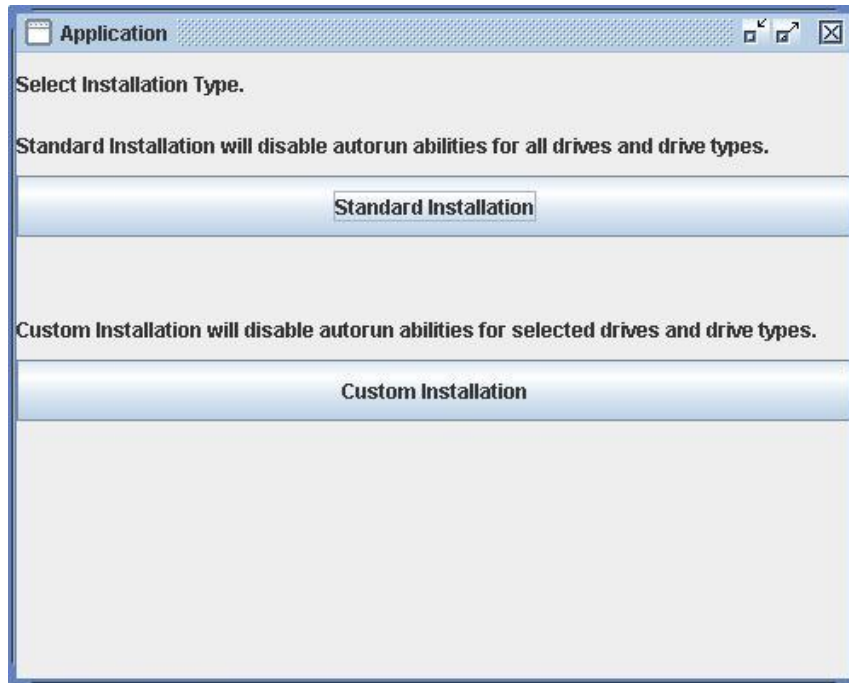


Figure 3: *SafeByte Setup Window*

Maintaining other parts of this software’s functionality while re-enabling auto-run can be done by choosing devices or drive letters to allow auto-run capabilities in the ‘Custom Installation’ option (Figure 4). It is suggested that the user select the default setting, where all auto-run is disabled, to prevent the possibility of the flash drive emulating a device which still has auto-run enabled.

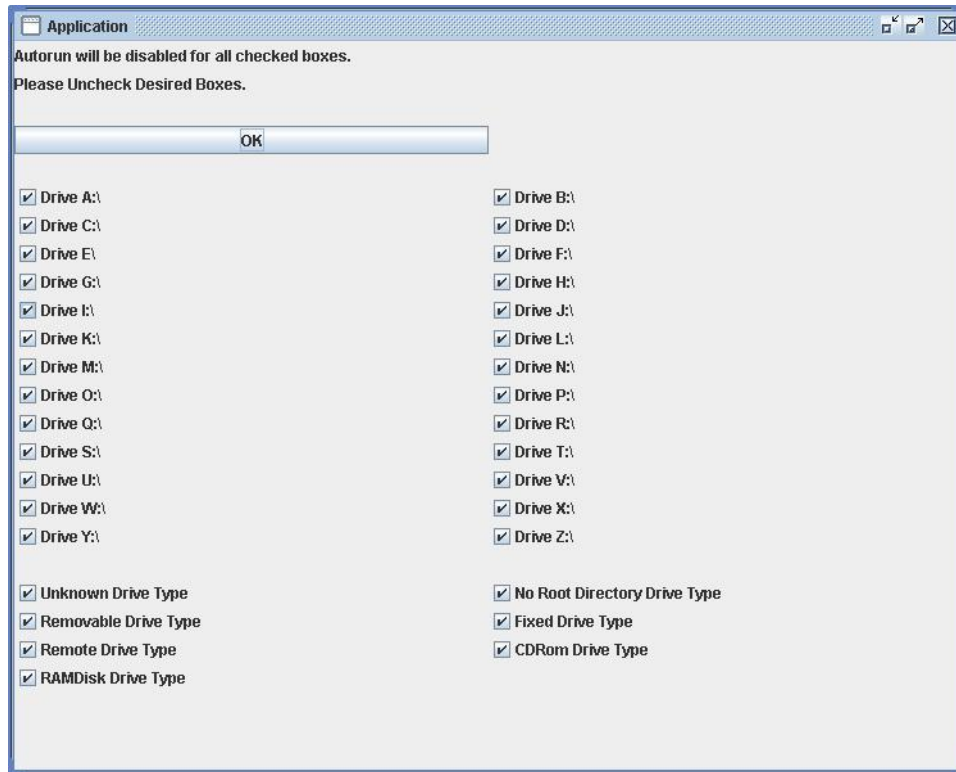


Figure 4: *Auto-run Disable window*

Once the proper selections are made, the user will be asked to restart his/her computer to have the changes work. (Figure 5)



Figure 5: *Window showing completed installation*

After installation, the folder where SafeByte was installed on the user's computer is displayed

### **Automatic Virus Scan Prompt:**

Installing SafeByte ensures that the Automatic Virus Prompt and the Auto-run disable features are enabled. Once the user inserts the SafeByte flash drive after installation, the Automatic Virus Scan Prompt window is displayed. (The auto-run disable feature is also in operation after installation)

It is highly suggested that the user should use his virus scan application to scan the USB flash drive mentioned in the virus scan prompt. Once the virus scan client has finished checking the flash drive, click the ok button.

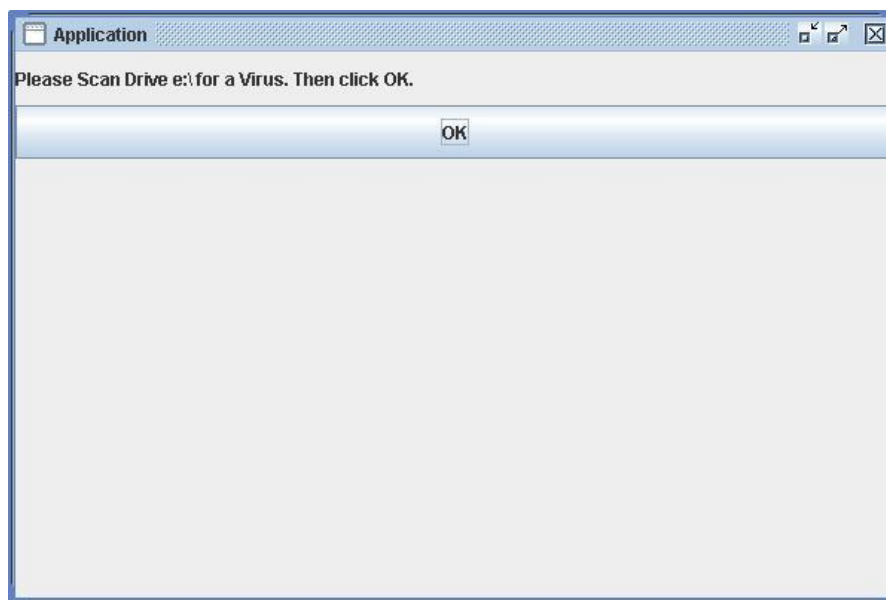


Figure 6: *Virus Scan Prompt*

### **Authenticating the SafeByte Medical Application:**

The authentication utility is run after clicking the "ok" button in the Virus-Check window. This gives the user the option of verifying the application file's digital fingerprint with the authentic SafeByte fingerprint.

1. Once the Virus Scan is completed, a window pops up with the option of verifying the application. The user can check yes to run the authentication, or no to skip this step and quit SafeByte Security. (Figure 7)

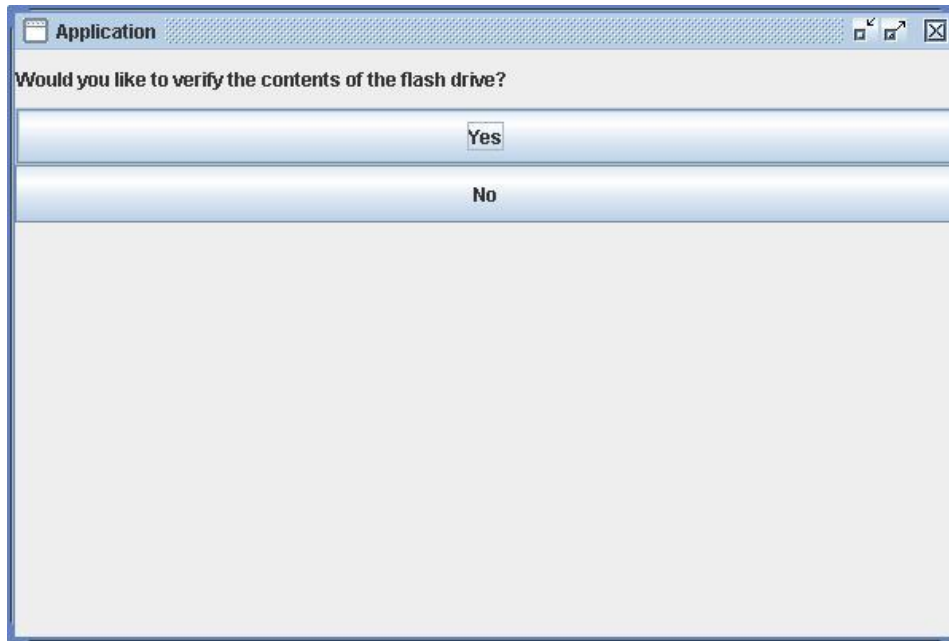


Figure 7: *SafeByte Authentication Window*

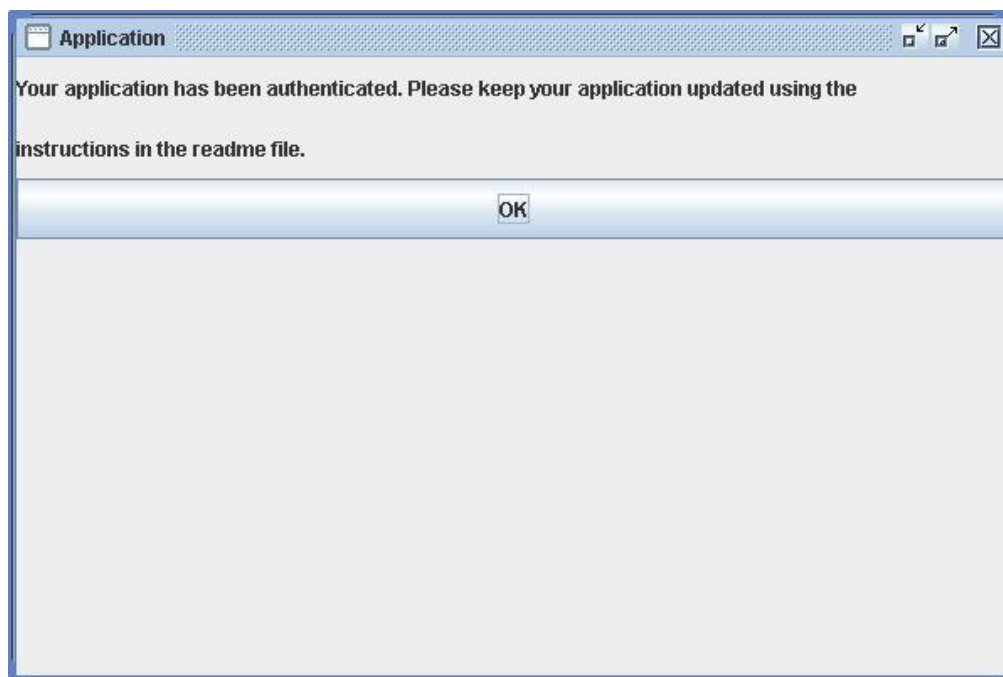


Figure 8: *SafeByte Verification Results*

2. Once the verification results are shown on the screen, the user will be given the choice to verify the application using a signature online. The user should click either the yes or no button. (Figure 9)



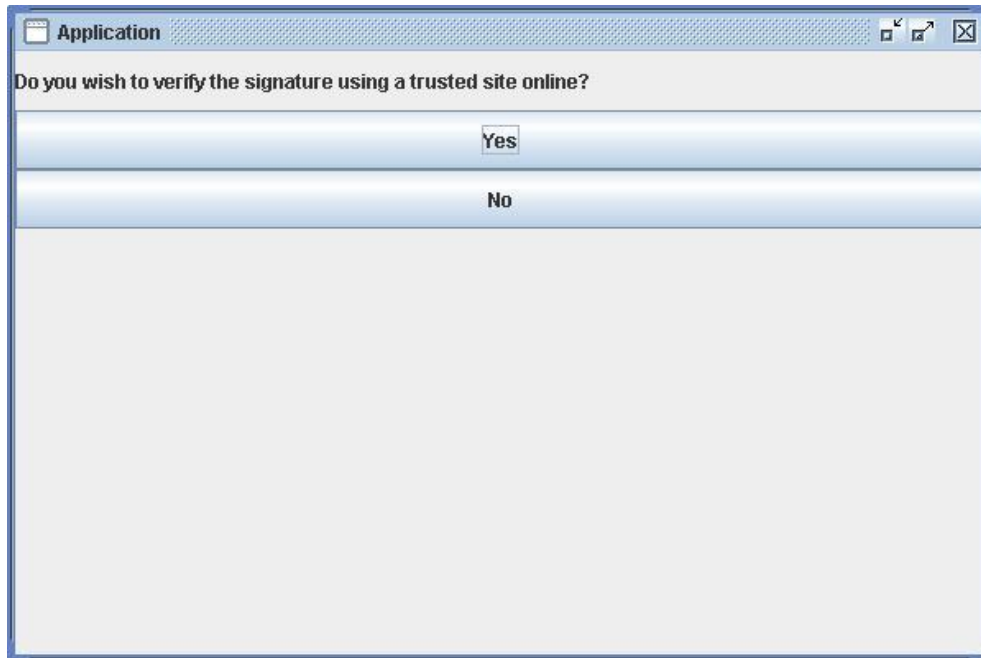


Figure 9: *Online Authentication Option*

3. After the online verification, the security tools exit, and the SafeByte Medical Application starts up (Figure 10)

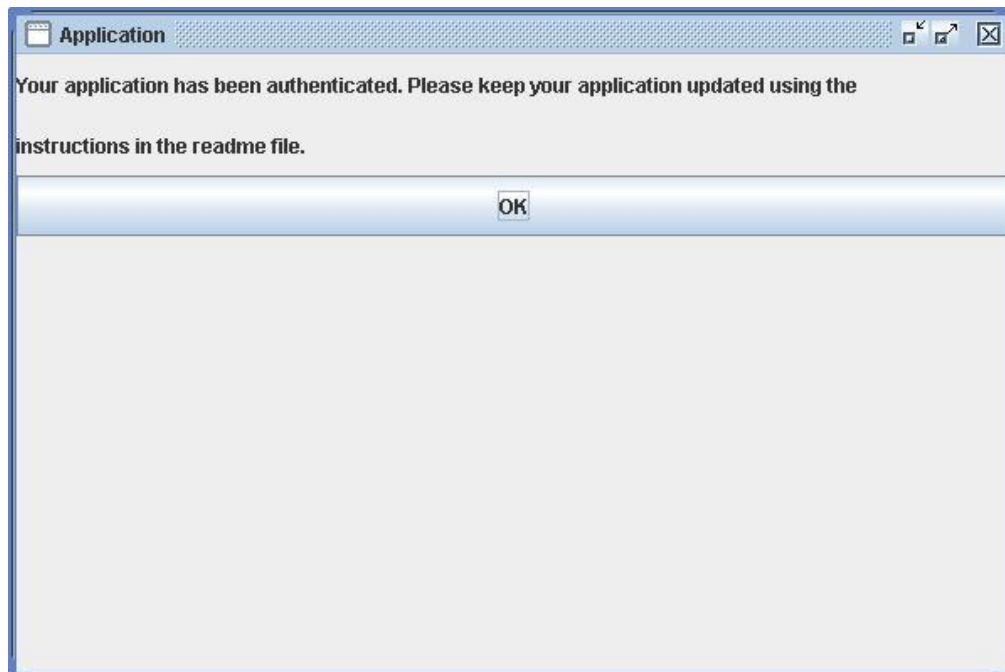


Figure 11: *Final SafeByte Security Window*

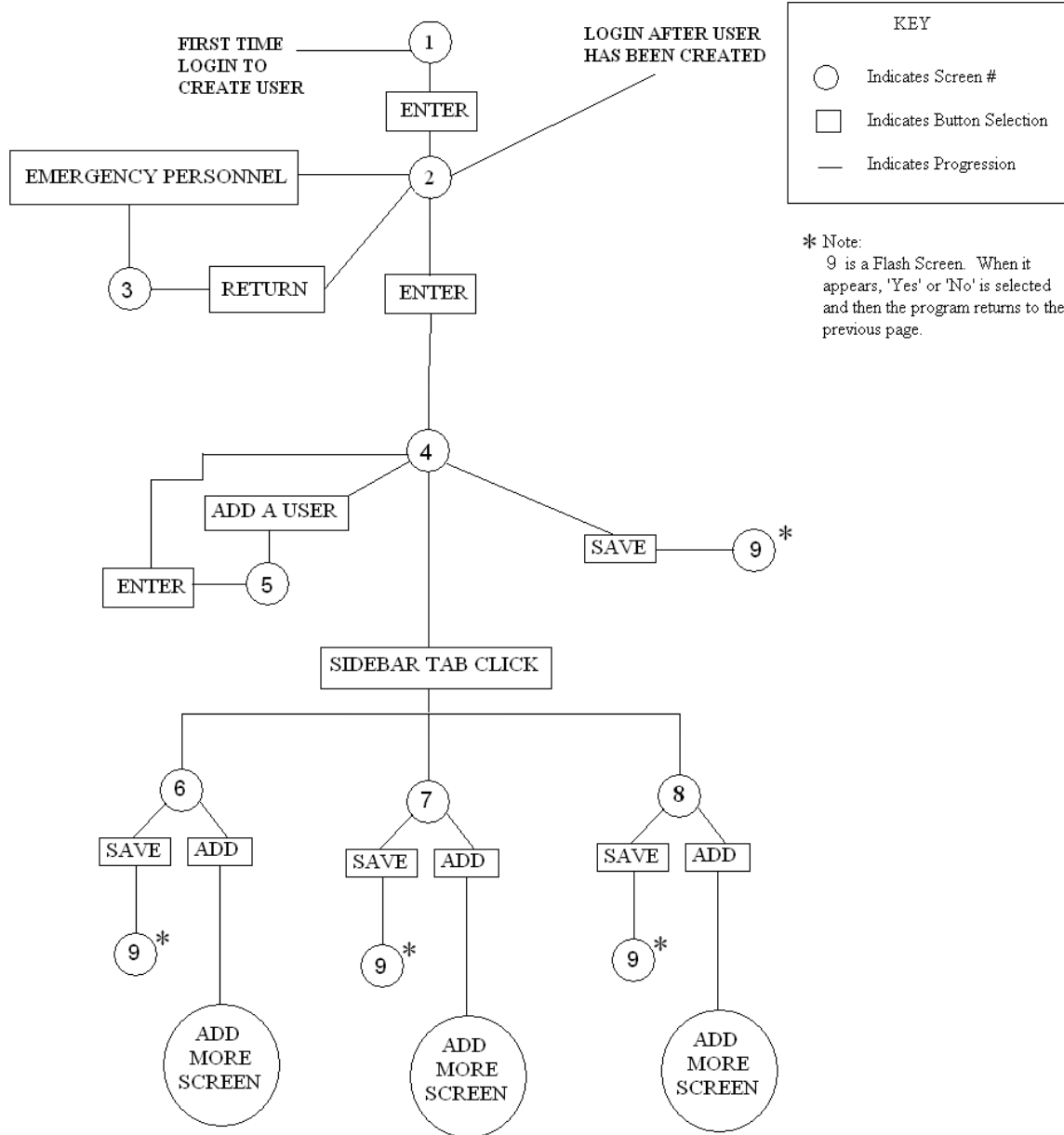
Once all the SafeByte security procedures described above are completed, the program will allow the user to sign into their account and have access to the medical information. The other option to see data is the Emergency Personnel ability, which would allow for some data to be seen in case of an emergency.

## **Application Walk Through**

### Screen Order

1. Welcome
2. Login/Emergency Screen
3. Emergency
4. Personal
5. Add User
6. Allergies and Illness
7. Medications
8. Vaccinations
9. Confirm Delete/Save

# FLOWCHART



**Figure1. General Flow of the Application**

The general flow of the application is shown in **Error! Reference source not found..** In this flow chart, each circle represents a dialog window that appears on the screen, and each rectangle indicates a user action/decision. The user is first asked to authenticate him/herself via a password prompt. Subsequently, s/he will be able to read, update, and insert data based on his/her access level. For example, the data accessible to a patient is

not necessarily the same as that which is accessible to a doctor. After each update, the user will be given the option of saving the changes.

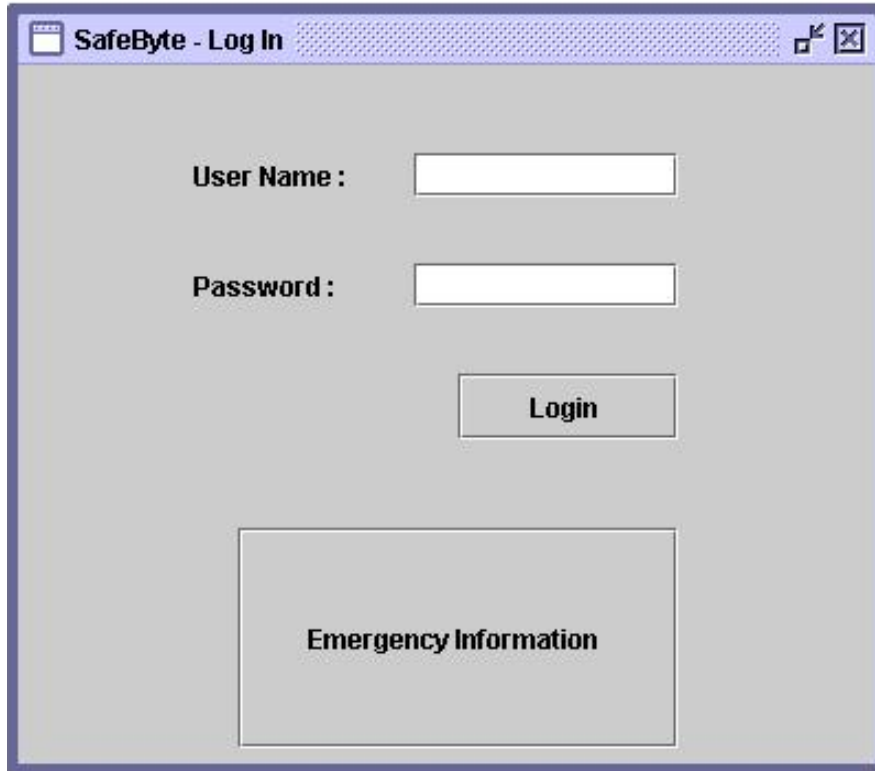


The image shows a software window titled "SafeByte - First Time Registration". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. The main content area is light gray and contains three text input fields. The first field is labeled "User Name :", the second "Password :", and the third "Confirm Password :". Below these fields is a "Register" button.

**Figure2. Welcome**

Figure 2 shows the first screen of the application that will appear on the very first time of operation once the program is installed. The user will need to create an account in order to proceed. This account created will serve as the primary account for the application. In order to do this, the user should enter a username in the first box shown. Then a password should be entered in the second text box. When this is typed, \*\*\*\* will appear because the password will not be displayed on the screen. It is recommended to have at least 6 characters as a password. It is also recommended to have numbers included in the password. Once it is typed for the first time, it should be re-typed in the “confirm password” box. This eliminates the chance of spelling errors. Once the user is satisfied with the name and password, they should click “register” to create the patient account.

After the user is created for the first time, it will not be possible to go back to this screen. Subsequent execution of the application will lead directly to Figure 3.



**Figure3. Log in**

After creating an account for the first time, Figure3 will appear on the screen. This is the main login screen. The username and password just created in figure 2 should be entered in the appropriate boxes on this screen in order to login to the application.

If this is not the first time the application is run, then this is the primary screen that will appear after executing the application. At this point the user has two options.

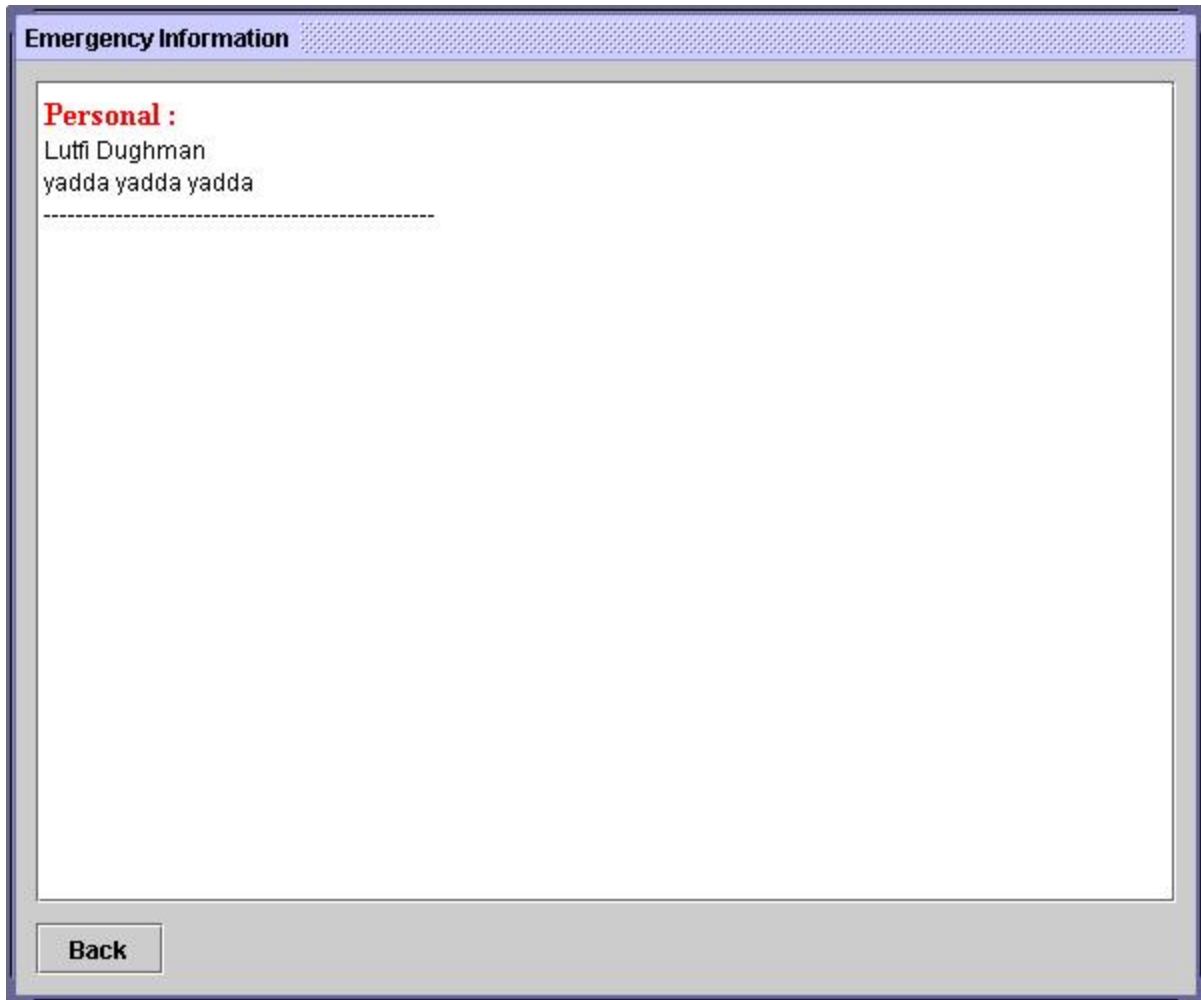
1. Enter a user name and password, then click login. This will take the user to the personal screen (Figure 5). The user name and password could be either the set created in Figure 2 or it could be the user name and password of another user such as a doctor or pharmacist.
2. Click emergency information. This takes the user to the emergency screen (Figure 4).

Patient login:

Enter the user name and password created for the first time in Figure 2. Then click “login”.

Doctor or pharmacist login:

Enter the user name and password created in Figure 6. This will be explained in detail on page 8. Once the name and password for the doctor or pharmacist is entered, click “login” to access the personal page with specific privileges.



**Figure4. Emergency**

The emergency information page will display information that is necessary in an emergency. This will include personal information, allergies and illnesses and medications. A login is not required for this page, and hence anyone can assess it at anytime.

Nothing on this screen can be deleted, added or updated. It is in READ ONLY format and so it just displays an output. If the relevant information is more than what will fit on a single page, then a scroll bar will appear to go through the page.

The only action that can be performed on this page is the “back” option. If this button is clicked, the user will be taken back to the previous login screen (Figure 3).

The screenshot shows a web application window titled "SafeByte" with a menu bar containing "File" and "Settings". The main content area is titled "Personal" and contains a sidebar on the left with the following menu items: "Personal", "Allergies and Illnesses", "Medications", and "Vaccinations". The "Personal" section is active. The form fields are as follows:

- Name:
- DOB:
- Age:
- Gender:
- Blood Type:
- SSN:
- Address:

At the bottom right of the form area, there are two buttons: "Save" and "Refresh".

**Figure5. Personal**

Figure 5 shows the personal page which is first screen that appears once a user has logged in from Figure 3.

This page has several functions:

1. Patient demographic information.

- The user will be able to add their demographic information on this screen. It is possible to add name, DOB, Age, Blood type, SSN and Address by clicking on the text box next to each label and then typing in the relevant information.
- Once the user has added this information, they can click save, so that it is stored and will appear every time this screen is opened. The “save” button allows this function.
- Information can be edited by simply selecting it and then adding or deleting necessary data. This can be saved also by clicking on the “save” button.
- In order to select gender, the user must chose male or female clicking on the drop down menu next to the label “gender”. This is saved with the other data by clicking “save” once again.

2. Ability to navigate to other screens in the program such as allergies and illness, medications and vaccinations. The ability to access the four main screens in the application appear in the tab in the top left corner of the screen. By clicking on any of these options, the user will be taken to the relevant page of the application. This tab will appear on all four of the screens so that the user can easily move between screens by simply clicking on that name in the tab menu.
3. Has a menu bar which allows a user to exit the program or access another user manager screen (Figure 6). The personal page has a menu bar on the top of the screen. By clicking on “file” the user can then select “exit” to end the program without saving changes. Alternatively, under the “settings” menu the user can access the “user manager” screen (Figure 6) which is explained in detail on the next page.
4. Allows different levels of access by blocking certain demographic information. The information stored on the personal page can be viewed by all users that login into the system. However, only a patient or pharmacist can edit/delete or add this data. Therefore, when a doctor logs into the program, these fields will be seen, but blocked so that they cannot be edited. The doctor will be unable to click on any of these fields to enter information. If a patient or pharmacist is logged in, then they will not be blocked. In this case, the user can click on the text box to make changes.
5. Allows different levels of access by blocking who can open allergies and illness, medications and vaccinations. The tab on the left hand side will also have different access levels for the different types of users that log onto the system. If they are available to access, then they will appear in white so that the user can click on it to reach that screen. If they user does not have access then they will be greyed out.



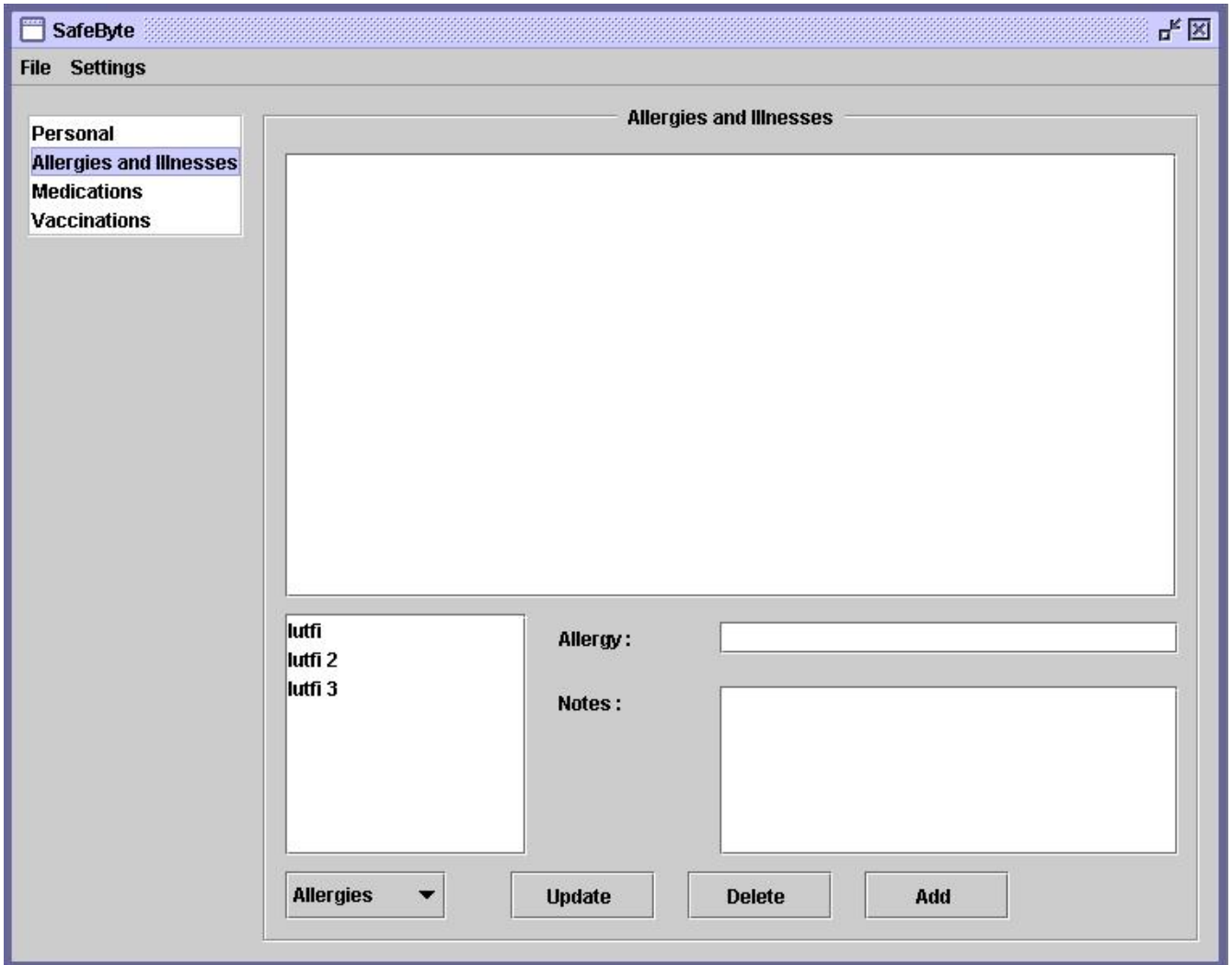
**User Manager**

**UserName :**

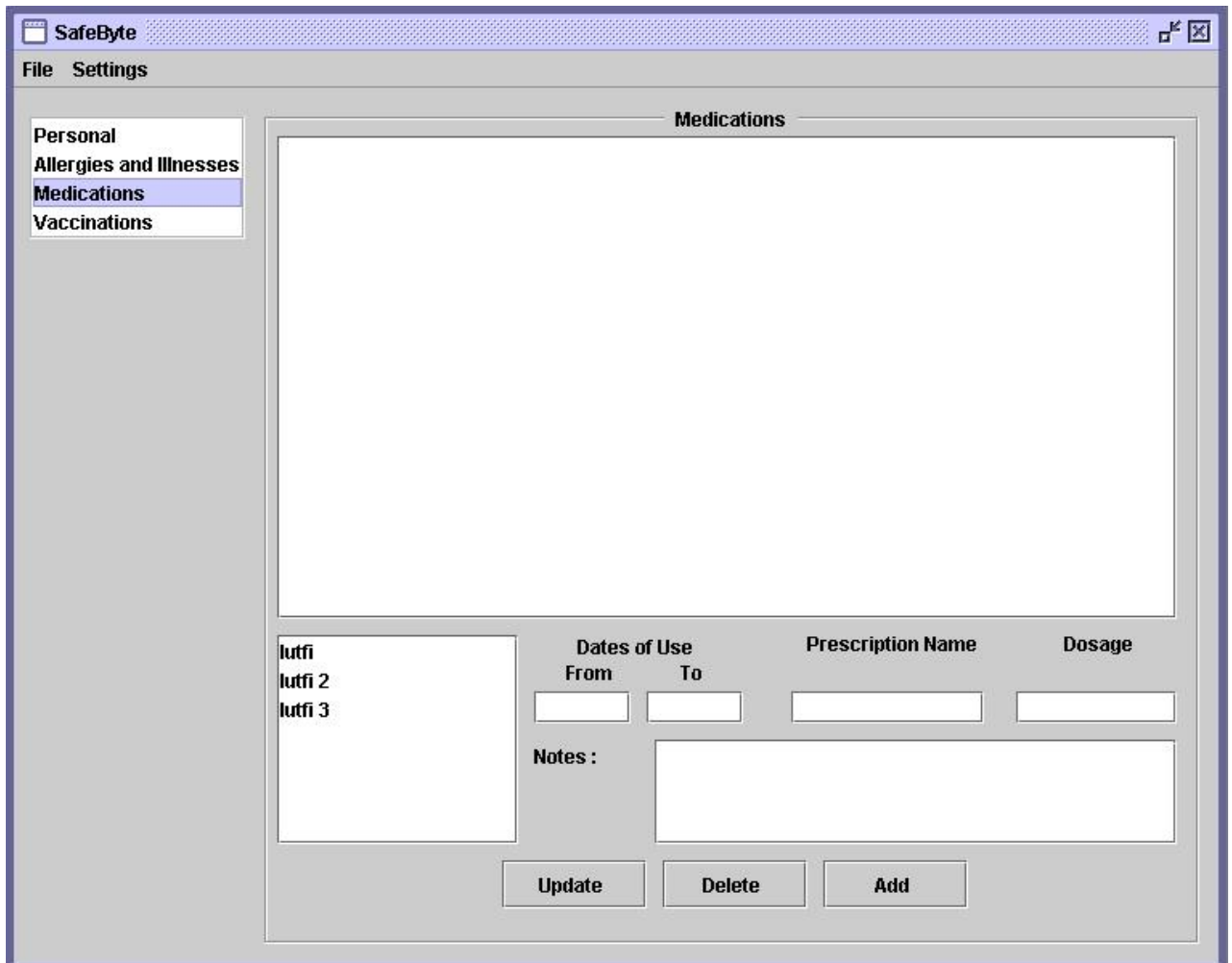
**Password :**

**Confirm Password :**

**Figure6. Add User**



**Figure7. Allergies and Illnesses**



**Figure8. Medications**

The large text box in the center of the screen will display all the important information that is relevant to one particular type of medication that the patient is taking. The bottom left hand corner will display the different medication names.

This screen can be viewable by all users of the application. However, patients will not have access to edit/add/delete any of the data. In order for these to be changed, either a doctor or a pharmacist has to log in.

Adding a medication:

The doctor or pharmacist can add a medication by typing in text into the boxes in the lower right corner of the screen. The dates of use should be entered, followed by the prescription name, dosage and any notes. Once all the information is complete, the data can be saved by clicking “add”. Once this is complete, the medication name will appear in the box shown in the lower left corner of the screen.

Viewing a medication:

To view details about already existing medications, the user should click on the appropriate drug displayed in the bottom left text box. Once this drug name is clicked, all the information regarding it will appear in the large central text box. To view another one, a user can simply click on another name, and the information will appear.

Deleting a medication:

To delete a medication, the user must first select it from the list box in the bottom left corner. Once it is selected, the user should click “delete”. This will pop up a confirm delete screen which allows the user to confirm if they really want to delete the record. If “yes” is selected, the name of the drug and all its information will be deleted from the list.

Updating a medication:

To update a record, the user should select it, change any information on the screen and then press “update”.

The screenshot shows a software window titled "SafeByte" with a menu bar containing "File" and "Settings". On the left, a vertical menu lists "Personal", "Allergies and Illnesses", "Medications", and "Vaccinations", with "Vaccinations" highlighted. The main area is titled "Vaccinations" and contains a large empty rectangular box. Below this box is a form with the following elements:

	Date	Doctor	Vaccination Shot Given
lutfi	<input type="text"/>	<input type="text"/>	<input type="text"/>
lutfi 2			
lutfi 3			

Below the table is a "Notes :" label followed by a large text area. At the bottom of the form are three buttons: "Update", "Delete", and "Add".

Figure9. Vaccinations



**Figure10. Confirm Delete/Save Screen**

This screen shown in Figure 10 simply allows the user to confirm whether or not they would like to delete the item they selected. If they clicked delete on any of the previous screens of the application by accident, then they can click “no” to return without deleting the information. If however, they intend to delete the information, this can be achieved by clicking “yes” on this screen. This screen allows a second chance to confirm an action. A similar screen will appear with a yes/no option for saving records. Directions for using the confirm save screen are the same as the confirm delete screen that is shown above.

## Citations

USB Flash Drive Alliance:

[http://www.usbflashdrive.org/usbfd\\_faq.html](http://www.usbflashdrive.org/usbfd_faq.html)

PC911:

<http://www.pcnineoneone.com/howto/auto-run2.html>

Technical Manual  
and  
Debugging Utility  
For  
**SafeByte**



Last Updated Thursday, April 29, 2006

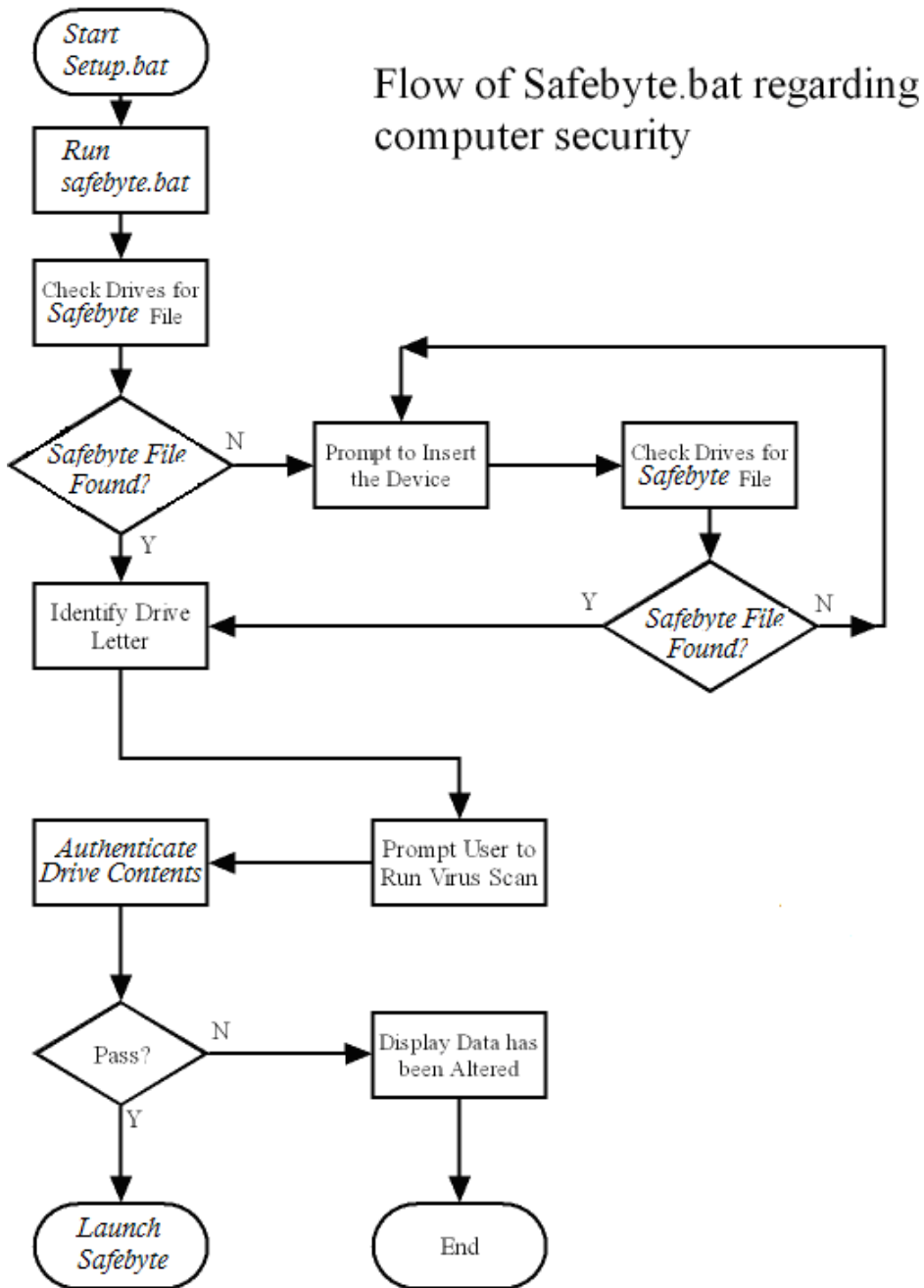


The goal of this is to establish “trust” in the SafeByte flash device. (Trust means that a user can be reasonably confident in plugging in the device and then running the SafeByte application located on the device.) This involves, in order:

1. Ensuring that plugging the flash returns the right operation
2. Ensuring that the content of the flash drive are authentic

Once you have accomplished these two things, the rest of the security is up to the design and implementation of the SafeByte application.

## Safebyte Computer Security



**Figure. 1** Flowchart showing steps in security operations

## Technical Manual

### SafeByte has two levels of Computer Security

listed below:

#### 1. **Physical Interface Security (PIS)**

Dealing with Preventing Auto-run Attacks and preventing User Triggered Attacks.

**Designed by:** Steven Banaska  
[banaste@iit.edu](mailto:banaste@iit.edu)

Michael Brenyo  
[brenmic@iit.edu](mailto:brenmic@iit.edu)

#### 2. **Authentication of Drive Contents (ADC)**

Implementing an Authentication utility to give a simple but effective way of verifying the integrity of the SafeByte application file

**Designed by:** Usman Abubakar  
[abubusm@iit.edu](mailto:abubusm@iit.edu)

Ikechi Emelogu  
[emelike@iit.edu](mailto:emelike@iit.edu)

## Physical Interface Security

AutoRun or AutoPlay is the ability of many modern-day computer Operating Systems (OS) to automatically take some action upon inserting of removable media such as CD-ROM, DVD-ROM, or in this case a USB flash drive. AutoRun is intended for convenience and, it automatically prompts an install option when the removable media is inserted.

“The auto-run capabilities [of Windows] are restricted to CD-ROM drives and fixed disk drives. If you need to make a USB storage device perform auto-run, the device must not be marked as a removable media device and the device must contain an *autorun.inf* file and a startup application.” (Microsoft)

Windows can be configured to disable auto-run and the ideal method of doing so is by editing Windows Registry settings. The change remains until the registry is rewritten, but is the best way to ensure safety against the AutoRun feature. Editing the registry will guarantee the maximum security from a flash drive attack. The registry would be changed so that auto-run will be disabled for all devices, because flash drives can emulate other devices. Though the device will not automatically run software itself, the potential for the user to unknowingly spread a virus still exists. The flash drive should ideally be scanned for a virus using an antivirus program to be certain of the computer’s safety and the content of the drive must be determined to be authentic.

The installation process of disabling auto-run (Reg\_Edit.java) is written such that it will back up the HKEY CURRENT USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer portion of the registry, in case the old settings need to be reverted to. The backup is done through the Regexport() method. The program will create a “*AutorunDisable.reg*” file which will be imported into the registry and change the necessary values, through the Createreg() method. The values of “*AutorunDisable.reg*” are determined by the user’s installation selections: Standard Installation will disable auto-run for all drive letters and drive types; Custom Installation will disable auto-run for selected drive letters and drive types. The user must then log off and back onto Windows or restart the computer in order for the registry settings to take effect.

The process would be done when the software is installed and only at this time, to avoid repeatedly writing the sensitive registry. The program allows the User to select the level of protection from auto-running devices and also if particular drive letters should have their auto-run abilities will be disabled. The default setting disables auto-run for all information media such as USB flash drives, DVDs, CDs, and floppy disks. There is a file, “*regrevert.bat*” that can be used to revert back to the registry settings that were set before SafeByte was installed, though the action is not recommended.

Safebyte uses USB\_Check.java to identify the correct device and recommend a virus scan. The ListenUSB() method is called in the main method and it calls another checkDrives() method. checkDrives() takes a File System View of the computer and loops through each drive letter to see if it is a drive, if

the drive exists, it is tested to see if the application file exists. If the application can not be found, the user is asked to insert the correct device.

Once the file has been found, the drive letter of the drive is used to recommend a virus scan of the storage device, as shown in Figure. 1. The user has the option to verify the integrity of the contents of the drive using an authentication utility, **SafeByteV** which is described in the Authentication of Drive Contents section.

## Class Description

### Reg\_Edit Class Description:

Reg\_edit is used during installation to set the registry to disable auto-run based on the user selection.

### Reg\_Edit Functions:

- **Regimport ()** – AutorunDisable.reg is imported into HKEY CURRENT USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer section of the registry.
- **Regexport ()** – The contents of HKEY CURRENT USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer are saved in Originalsettings.reg.
- **Regrevert ()** – Originalsettings.reg is imported into HKEY CURRENT USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer section of the registry.
- **Createreg ()** – AutorunDisable.reg is created based on variables that the user selects.

### USB\_Check Class Description:

USB\_Check is used to identify the correct drive with the Safebyte application and recommend a virus scan. USB\_Check will call the Verify functionality described in the Authentication of Drive Contents section.

### USB\_Check Functions:

- **ListenUSB ()** – Displays the GUI to the user based on the results of the methods that are called. Uses checkDrives() to find the correct drive letter of the flash drive. Recommends the virus scan of the drive letter to the user. Calls the Verify functionality if the user chooses to authenticate the drive contents.
- **checkDrives ()** – Creates a File System View of the computer. Loops through a-z drive letter names and checks if they exist. If the drive exists, it is tested to see if the application file is on the drive. The drive letter is found here and the method returns an integer “devFound” that is set to ‘1’ if the device is located.

- **CreateBat ()** – Creates “Run.bat” that is used to run the application of the flash drive. “Run.bat” is written using the drive letter because the drive will not always be designated the same letter.

## Authentication of Drive Contents

The authentication utility was designed to give a simple but effective way of verifying the integrity of the SafeByte application. This is a precautionary measure, whose goal is to thwart the replacement of the SafeByte application with malicious code with the same filename. Once the integrity of the SafeByte application has been verified, the user can trust that it will perform as its implementers intended. Authentication utility prompts right after the disabling autorun feature in the physical interface security.

The verification utility, **SafeByteV**, works by computing digital signatures of specified files, and comparing these signatures with pre-computed ones (use Figure.1 as a reference). If the signatures match, then the files’ authenticity has been verified.

The authentication scheme is meant to be elective. The user has the option of choosing to authenticate SafeByte at various levels of trust, each of which require various levels of effort/computational resources:

**Level 0** – No trust – The user does not invoke SafeByteV.

**Level 1** – Low trust – The user invokes SafeByteV using signatures stored locally on the flash drive.

**Level 2 (Recommended)** – Moderate trust – The user invokes SafeByteV using signatures downloaded from a trusted source. Level 2 is meant to counteract the case where an attacker would replace both the SafeByte application as well as the locally stored flash drive. To invoke level 2, the user must have an Internet connection.

**Level 3** – High trust – The user can download the SafeByte software from a trusted source.

Level 3, not supported by SafeByteV, requires the user to download a copy of the SafeByte application from a trusted source. The user may do this if s/he does not trust verification by digital signatures.

**SafeByteV** was designed to use the MD5 algorithm as the default tool for verification. MD5 is an open algorithm and is widely available for public use. The specific code used was located at the URL

<http://www.freevbcode.com/ShowCode.asp?ID=741> and created by Robert Hubley, MIT Laboratory for Computer Science and RSA Data Security, Inc.

There are three **class files**:

1. the main procedure file, (verify.java),
2. the main cryptographic function (MD5.java)

### 3. the cryptographic function interface (messDigest.java)

The file verification functions, as well as the main() procedure of this application is located in the verify.java file. The implementation of the cryptographic algorithm (in this case md5) is done in md5.java file and the messDigest.java holds the interface needed for any cryptographic function to be used with the application. The MD5.java file matches this interface.

For the purpose of file verification, there will be no need for the user to change any variables within the MD5.java file. **Note:** All the coding for the SafeByteV authentication utility was done in java under a J2EE 1.4 SDK (Includes Sun Java System Application Server PE 8.2) runtime environment which you can download anytime at <http://java.sun.com/download/>

## Class Description

### Verify Class Description:

The variables in the verify.java file define the file verification specifics, like the location of the file to be verified, the online url for the online verification option, etc, and these variables are:

1. **SigURLString:**  
Contain the URL for the online file verification option. **N:** The authentic signature file must have an identifier before the signature. This identifier is set via the `setStringPointer(String a)` function, described below. (default value: "http://www.geocities.com/ike9484/MD5sig1.html")
2. **AppFileString:**  
Contains the string for the name of the application file. This is set with the `setAppFileString(String a)` function in the main procedure.(default: "app.java")
3. **SigFileString:**  
Contains the string for the location of the authentic signature file string. This is set with the `setSigFileString(String a)` function in the main procedure.(default value: "sig.dat")
4. **StringPointer:**  
Defines what identifier will be used to locate the signature for the online option. Since HTML files contain other information, one needs an identifier to know where the signature string starts. Only useful for online signature verification. Can be set with the `setStringPointer(String a)` function in the main procedure. (default value: "Here")

### Other Verify Functions:

Apart from those used to set the verify class variables, the other functions in the verify class are:

- **FileToString()** – Converts the contents of the app.java file to a string. Uses both string and stringbuffer objects.
- **FileSigToString()** – Converts the contents of the signature file to a string.
- **verifySig()** – Takes the stored values from the two above functions, and compares them. Returns a true if they match, returns a false value if they don't.
- **inputVal()** – Used to take an input value from the user. Needed to confirm if the user would like to compare the application's digital signature to a certified value. Asks for a y or n, and returns a true or false value.
- **URLval()** – Used to collect the URL value contained in the SigURLString variable
- **SignatureCheckMessage()** – Function used to verify signatures, and display accompanying output. Uses the verifySig() function.
- **setSigURLString(String a)** – Sets the SigURLString variable.
- **setSigURLString(String a)** – Sets the SigURLString variable.
- **setSigFileString(String a)** - Sets the SigFileString variable.
- **Main()** – Runs the main procedure of the program.

The main function must have a “throws IOException” label at the start.

### **MD5 Class Description:**

The MD5 class contains variables needed to implement its message digest conversion. It would be wise not to change anything in the class. Any changes to the definition can potentially make the class produce signatures that do not match the industry standard required for MD5.

### **Non-Public Functions:**

These functions perform several insider tasks related to message digest generation. There is no need to call these functions directly.

### **Public Functions:**

These functions are called for the actual message digest generation operations in the MD5 class. `update()`, `md5final()`, `toHexString()`, `init()` came with the original MD5 code. `InputMess(String a)` and `outputSig()` were included later, and are required for the MD5 class to be compatible with the `messDigest` interface.

**Note:** `inputMess(String a)*` - Takes in a string (a) as an input for the MD5 digest and `outputSig()*` - Returns the digest



*\*Both these functions are required for the MD5 class to be useable by the verify function. Any other subsequent cryptography classes to be used must have these two functions for input and output.*

## **messDigest interface description**

This is simply the skeletal definition for any message digest or cryptographic hash class to be used for signature generation. The messDigest interface describes the functions required in the class to be used. Any class file (like MD5.java) that is used for signature generation is required by the messDigest interface to have the following functions:

```
inputMess()
outputSig()
toHexString()
```

The MD5 file described above matches this definition.

## **Operation**

The main operation is located in the verify.java file. All the changeable steps (including debugging and variable changing) must be done in the main function. The default authentication procedure is shown below:

```
MD5 md5Test = new MD5();
```

- a new message digest object is created. This can be MD5 or SHA.

```
Verify Fi = new Verify();
```

- a new verify object is created. This is needed for file verification.

```
Fi.FileSigToString(); //takes in the stored signature file..
```

- This method collects the authentic signature file, and converts the contents into a string for authentication.

```
Fi.FileToString()
```

- This will take the specified application file, and convert it to a string for authentication.

```
md5Test.inputMess(input)
```

- This takes in an input for the messDigest object to create a digest. The input will be the application file in string form, so the method should ideally be in this form:

```
md5Test.inputMess(Fi.FileToString())
```

```
Fi.SignatureCheckMessage(md5Test)
```

- This will display an output, either confirming the authenticity of the input to the md5Test object, or stating that it is unauthentic.

## **Online Verification**

This can be done in the same way as outlined above, but instead of the **Fi.FileSigToString()** method, we use the **Fi.VerifySigURL**. This will take the signature online using the specified variables in the verify class, and saves it in the same location for the local authentic signature.

## Software Design Troubleshooting Tips

Possible problems while running the authentication utility include, but are not limited to:

1. No application file found
2. No signature file found
3. No online signature found
4. Improper signatures
5. Usually, the output on the console/GUI in use will state these problems. If these do occur, then a capable user can go into debugging mode to correct these problems.

### Debugging mode:

The authentication utility can be checked for problems by setting the **debug** variable in the verify class to **true**. Debugging mode gives the user the choice of seeing the following:

1. The filename of the application being verified
2. The name of the file containing the authentic signature
3. The URL of the authentic signature online
4. The pointer that leads the signature (for online verification only)
5. The authentic signature (Initially, it's the local authentic signature. When the online verification option is selected, then it is the online authentic signature.)
6. The application's signature

This can be extremely useful for debugging the application. The generated signatures are displayed on a console window when the utility is run in debugging mode.

To go into debugging mode, use the `setDebugFlag(true)` statement in the main method. To return to regular mode, use `setDebugFlag(false)`

The following checks should be made to ensure that the signature generation and verification procedure work correctly:

- The user must make sure that the authentic signature file contains only the signature (no carriage returns after the signature), as this will affect the correct parsing of the authentic signature. Also remember to define the signature file name and location in the `sigFileString` variable, via `setSigFileString(sigfile.extension)`.
- The file to be verified must be located in the same location specified in the `appFileString`. This can be checked by placing a `getAppFileString()` function in the main function, (place in a display method, like `system.out.println()`) or running in debugging mode. The location & target of the file can be changed by making a `setAppFileString(../folder1/newfile.extension)` statement directly in the main function.

- If the URL verification returns as invalid even after changing the application and signature, then
  1. The URL specified in the program might be expired. The URL being used can be checked by placing a `getURLString()` function in a display method in the main function, or running in debugging mode. The URL can be changed with the `setURLString(www.newURL.com/newfile.html)` function placed directly in the main function.
  2. The URL pointer might be invalid. Check this by placing a `getURLPointer()` function in a display method, or running in debugging mode. This value can be set by placing the `setURLPointer(pointer string)` statement in the main function.
- The default values for the software authentication utility are shown below. These values also show the proper format for the variables.

```
SigURLString = "http://www.geocities.com/ike9484/MD5sig1.html"  
AppFileString = "../app.class";  
SigFileString = "sig.dat"  
StringPointer = "Here:";  
URLcheck = false;  
deBug = true;
```

When the authentication utility shows that the program and/or the signature are compromised, then a new application and/or signature should be downloaded online.

## Citations

USB Flash Drive Alliance:

[http://www.usbflashdrive.org/usbfd\\_faq.html](http://www.usbflashdrive.org/usbfd_faq.html)

Microsoft:

<http://www.microsoft.com/whdc/device/storage/usbfaq.msp>

PC911:

<http://www.pcnineoneone.com/howto/autorun2.html>

Hubley, Robert:

MD5 message *Digest Algorithm for Implementing Digital Signatures*,

<http://www.freevbcode.com/ShowCode.asp?ID=741>