

Report On Usable PGP

By:

Student Name: Sourabh Varshney

Roll Number: U16CO111

Contribution: Worked in PGP Backend Bug solving, Report, Preview Tour, Overall Designing, tooltips, Testing.

Student Name: Harsh Surti

Roll Number: U16CO049

Contribution: Complete PGP backend implementation in Python using PGPy, Startup automation for windows.

Student Name: Maulik Chevli

Roll Number: U16CO024

Contribution: Add Readme. Application architecture design. Implementation of application (flask) backend which integrates Front end, PGPy service, and Key server. Handle file storage on user local. UI functions like user notifications. Startup automation script for unix-like systems.

Student Name: Rishabh Rath

Roll Number: U16CO051

Contribution: Implemented major portion of UI using html, css, and Javascript. Form/Input error control. Fixing bugs in the Application backend. Testing of UI.

Student Name: Vidhey Joshipura

Roll Number: U16CO057

Contribution: UI implementation and worked on Icons as per the suggestions mentioned in the paper.

Student Name: Suprit Bhattacharjee

Roll Number: U16CO055

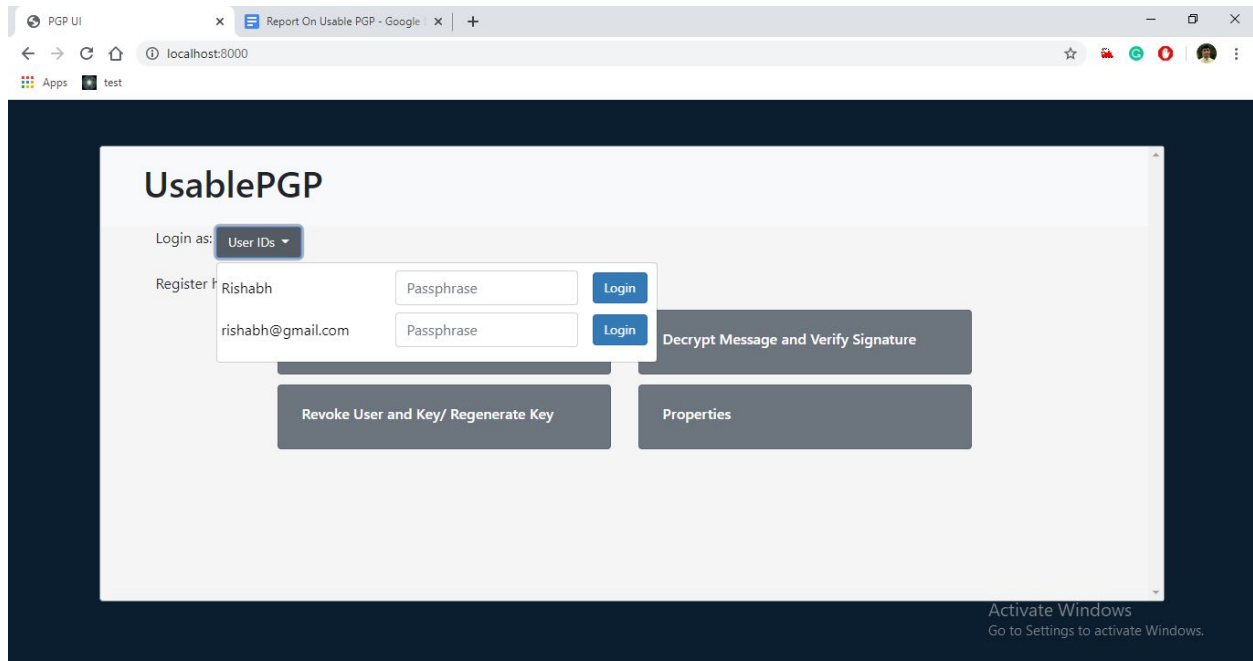
Contribution: Worked on Key Server API calls(for Key Management), Key Database, PGPy Backend(Protection of key with passphrase and salt), User email suggestions using AJAX

Introduction

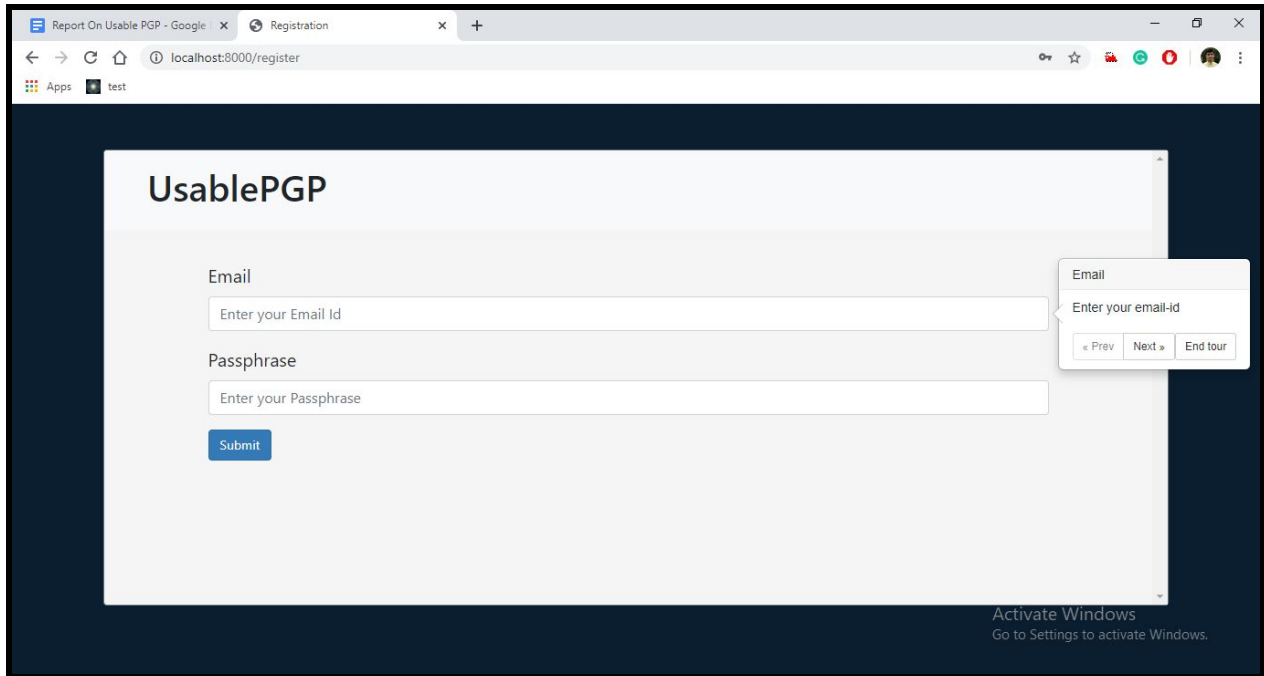
UsablePGP is an encryption programme that supports cryptographic privacy, confidentiality and authentication for communication of data over other softwares without worrying about the security of these softwares. Usable PGP overcomes the shortcomings of PGP 5.0 as discussed in the paper “Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0 By Alma Whitten and J. D. Tygar”. Usable PGP performs cryptographic operations on the local system, thereby overcoming the vulnerability of sacrificing confidential information over the internet. It has a secure key server holding the public key information of its users. Currently, it provides encryption, signing, decryption, verification, key revocation and key regeneration features. It also supports multiple users on the same device without compromising anyone’s security. It’s interface is easy to use even by novice users and even they will be able to use the UI to perform their task.

Key Features

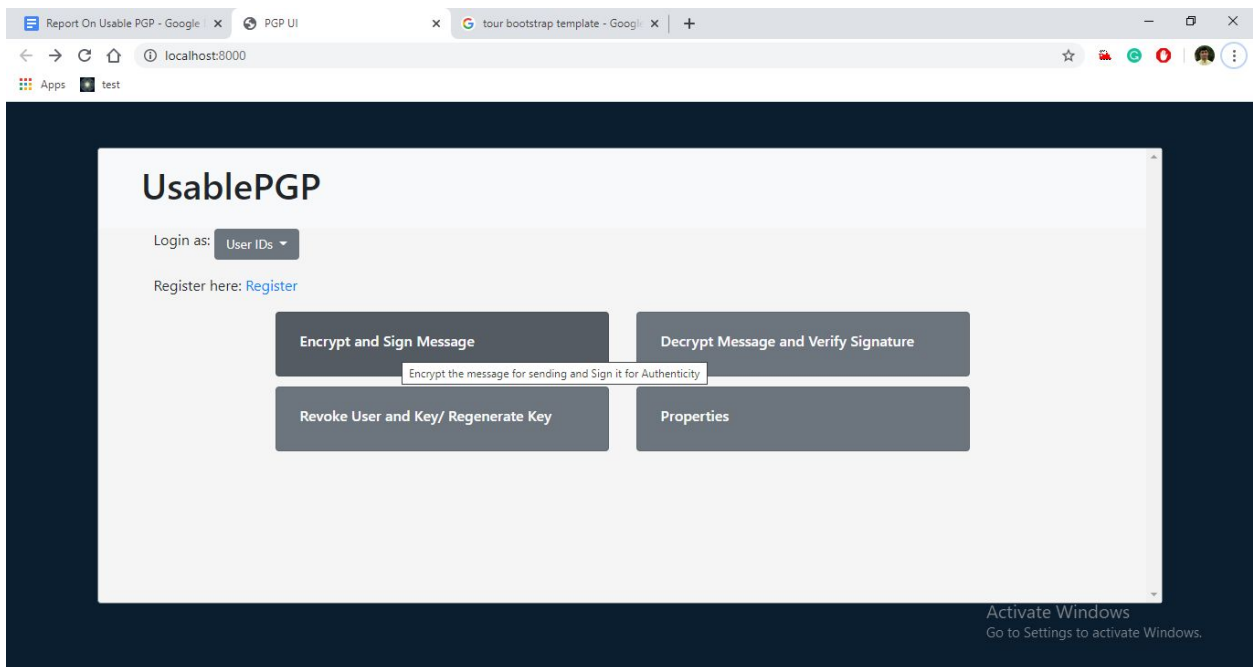
1. Encryption: It provides encryption of text or file input. The encryption standard used is 256 bytes RSA.
2. Signature: It provides digital signature on both text and file input. Signing is done using 256 bytes RSA private key.
3. Decryption: It provides decryption of text or file input. The decryption standard used is 256 bytes RSA.
4. Verification: Verification of digital signature is done using 256 bytes RSA.
5. Key Revocation: Users are allowed revocation of their keys. This can be easily done by selecting it from the menu.
6. Key Regeneration: Users can regenerate a new key by selecting the Key Regeneration option from the menu.
7. Properties: Users can get all the information about key properties, user available and other information of Usable PGP from this window.
8. Multiple Login, 1 device: We are providing secure use of multiple users on a single device.
9. Preview Tour: An initial Tour for each page is provided to ease and guide users how to use the app.
10. Tooltips: Tooltips are added on every possible tool with proper explanations.



Multiple user login from one computer



Preview Tour



Tooltips

Workflow

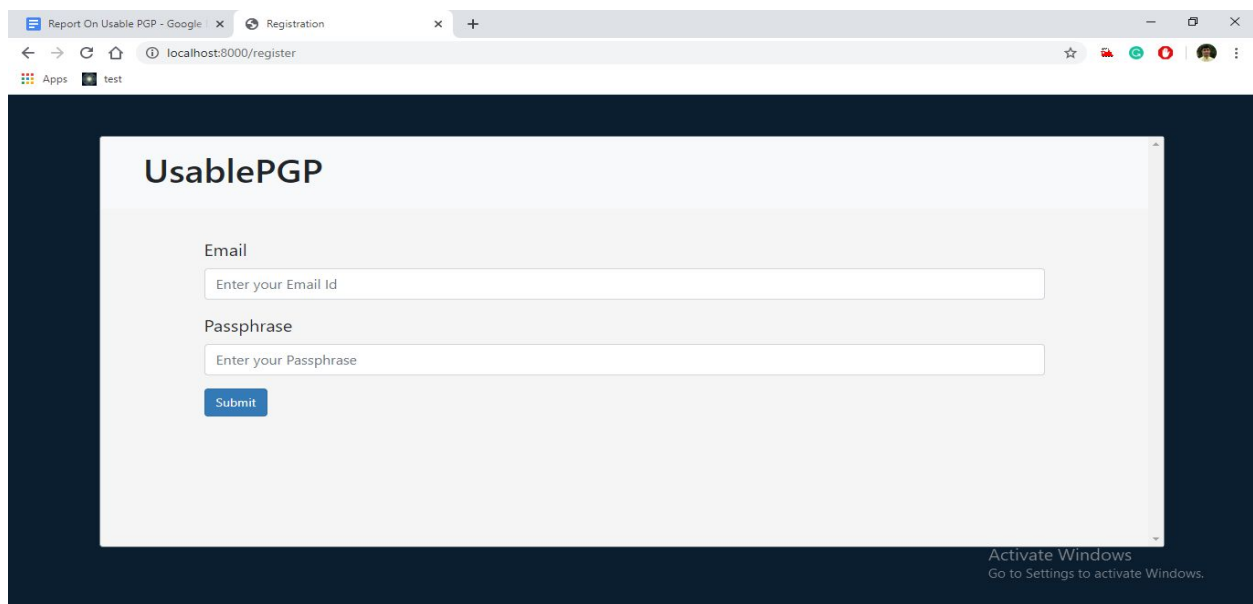
1. User Registration and Login:

Registration:

Users have to provide their Name, UsablePGP passphrase for using their private key while registration. The user is registered.

Background Process:

1. A public-private key pair is generated for the user.
2. A random salt is generated for the user.
3. The private key is encrypted using passphrase and salt and is saved in a hidden folder on the PC. This passphrase is only known to the user.
4. The hashed passphrase, salt and public key is stored along with the user name on the secure key server.



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/register'. The page title is 'UsablePGP'. The form contains two input fields: 'Email' with the placeholder text 'Enter your Email Id' and 'Passphrase' with the placeholder text 'Enter your Passphrase'. Below these fields is a blue 'Submit' button. The browser's taskbar at the bottom shows 'Apps' and 'test' icons. An 'Activate Windows' watermark is visible in the bottom right corner of the browser window.

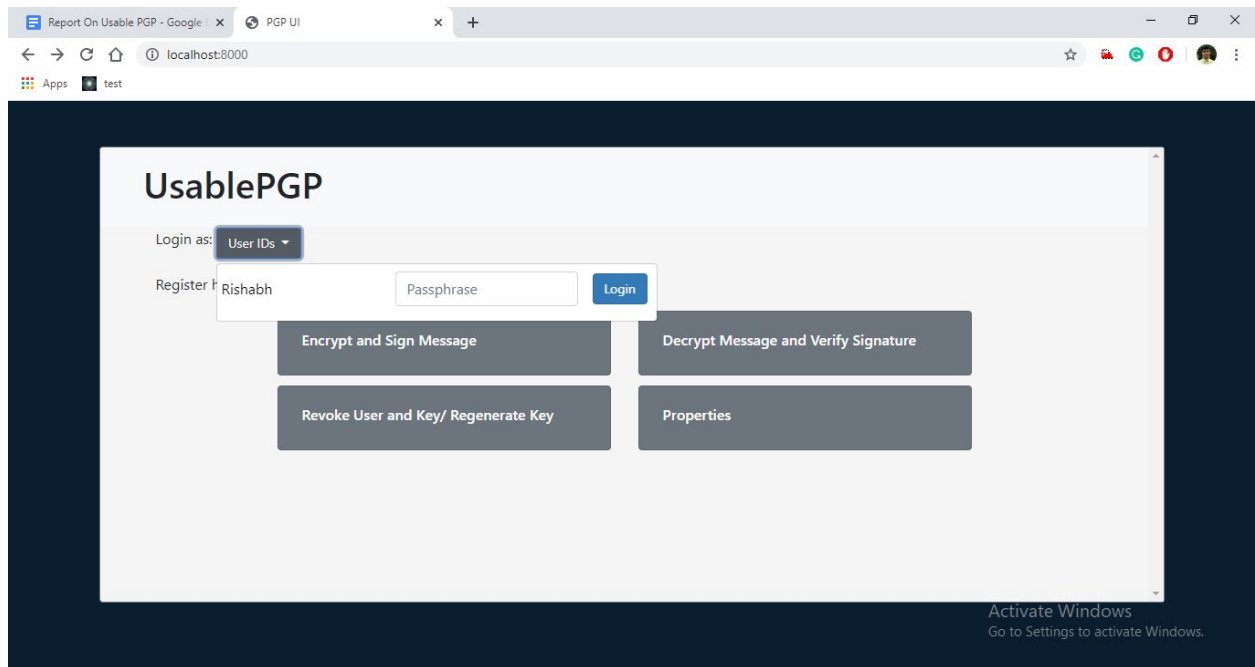
Registration

Login:

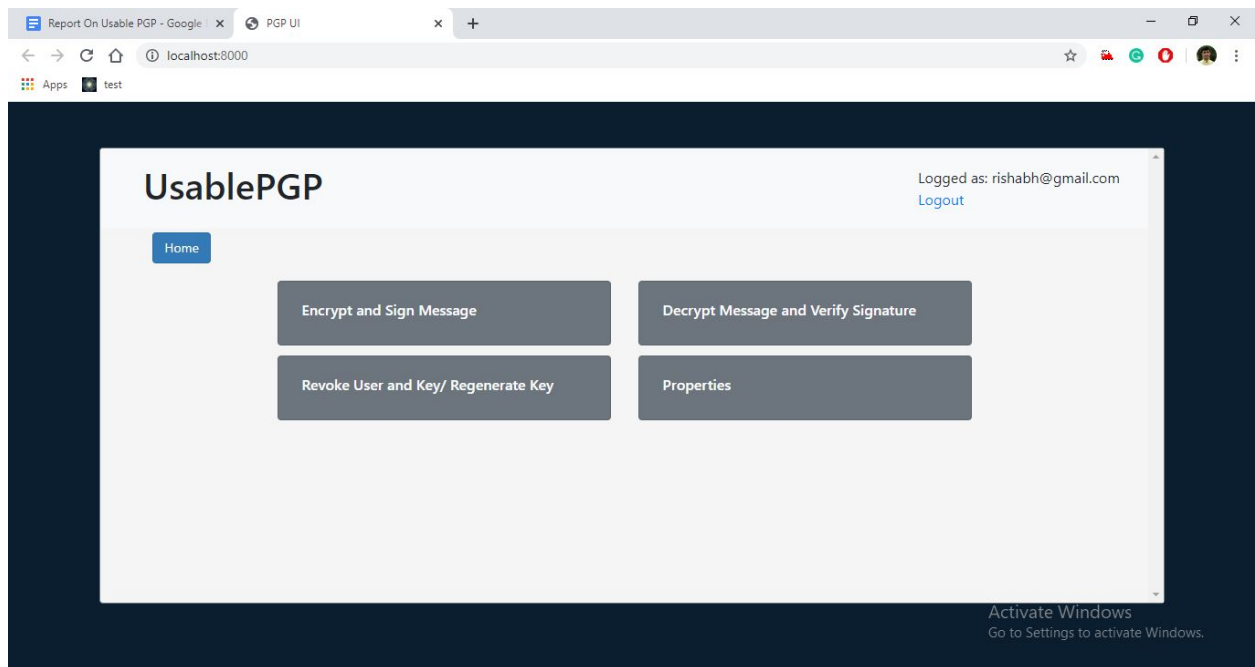
User will provide their Name and passphrase and upon providing correct information, the user will be logged in.

Background Process:

1. The user name and his provider's passphrase hash will be matched with the saved user's name and hash. If found matching, the user will be logged in.



Login



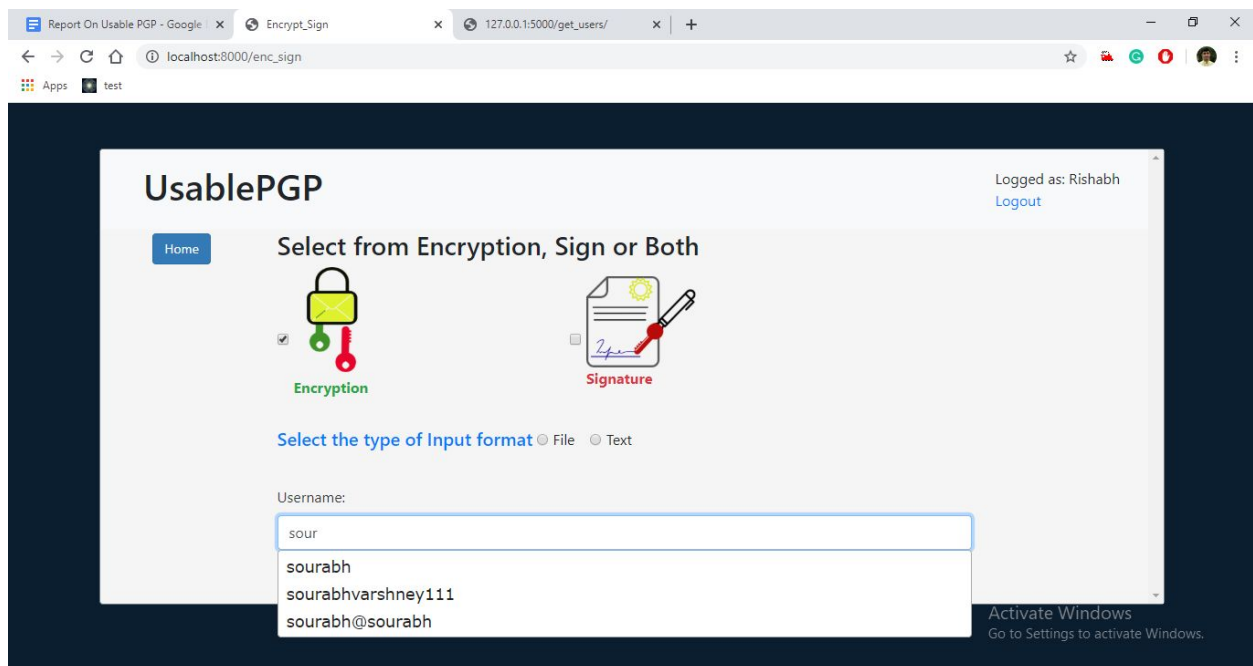
Main page after Login

2. Encryption and Signing:

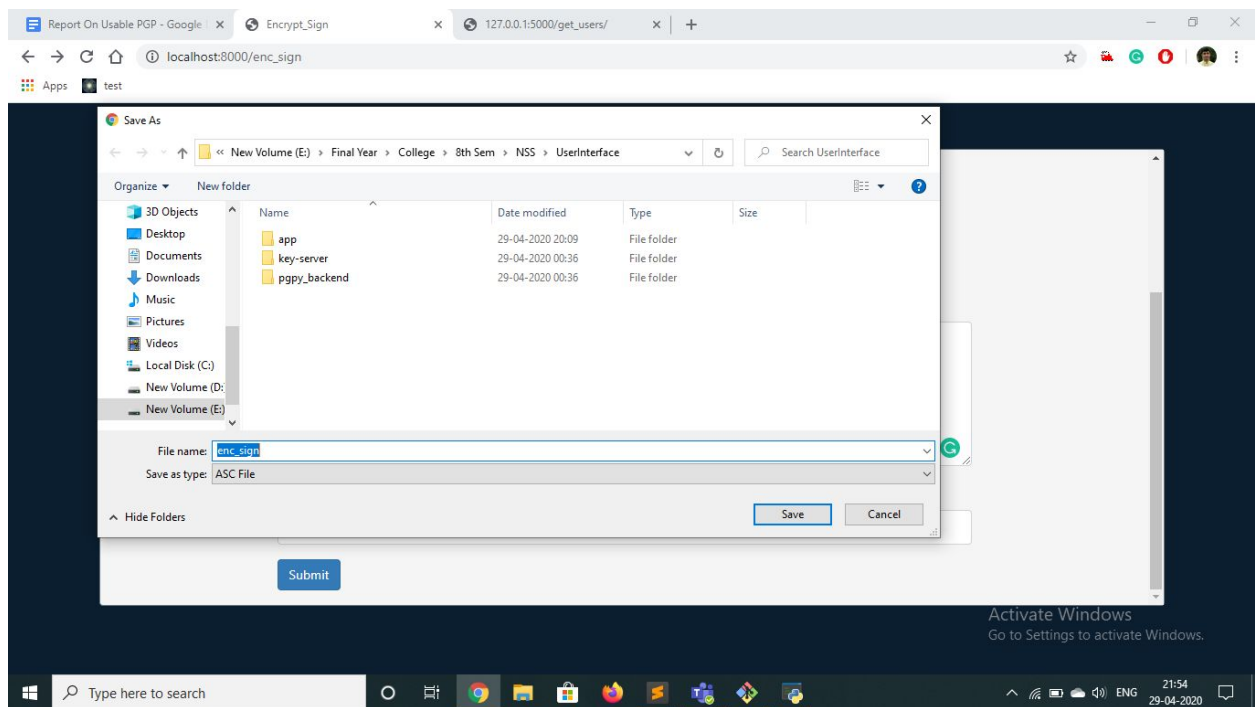
The user chooses among encryption, signing or both. Then, he inputs the message he wants to send in text or file format. If signing is done, the user is required to enter the passphrase again. If the user chooses encryption, he is allowed to name the user for which the message is intended to. The resultant operation(s) is/are performed on the input and resultant output file is downloaded.

Background Process:

1. If the user chooses encryption, then the user enters the name of the intended user. The message is then encrypted with the intended user's public key.
2. If the user chooses signing, then the user's entered passphrase is checked for verification. If correct, the private key of the user is used to sign the document. The signature is appended to the end of the message.
3. The file is downloaded with the encrypted/unencrypted and signed/unsigned message.



Encryption/Signature



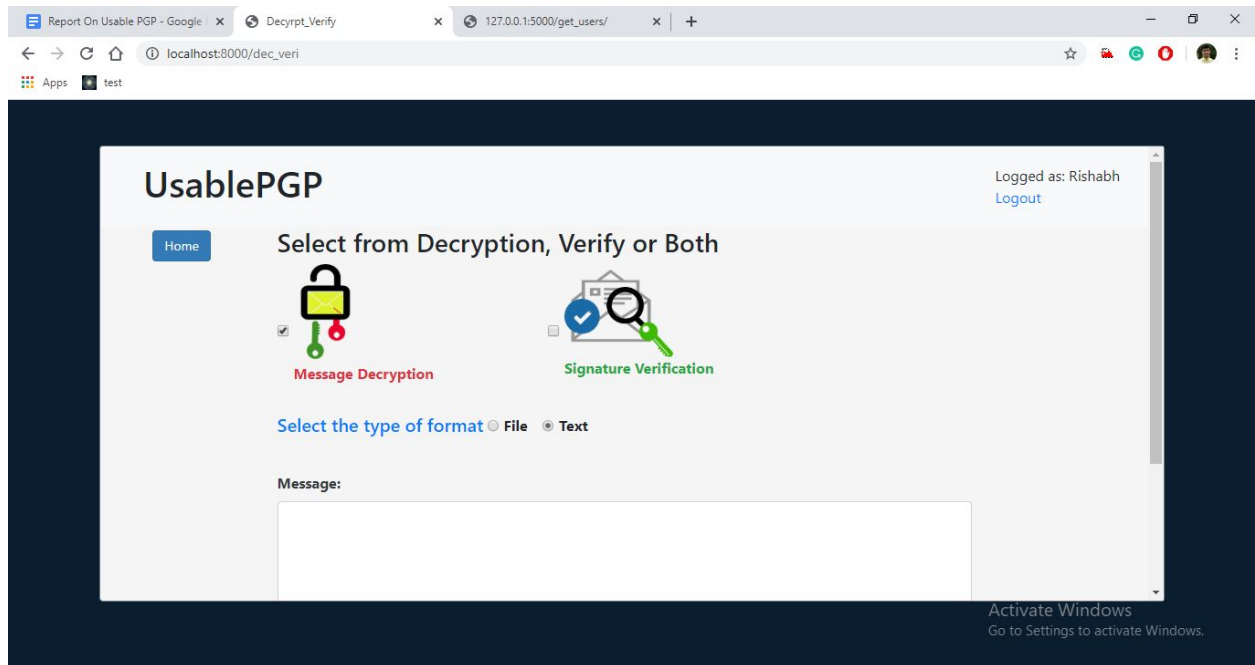
Downloading file when Encrypted/Sign

3. Decryption and verify:

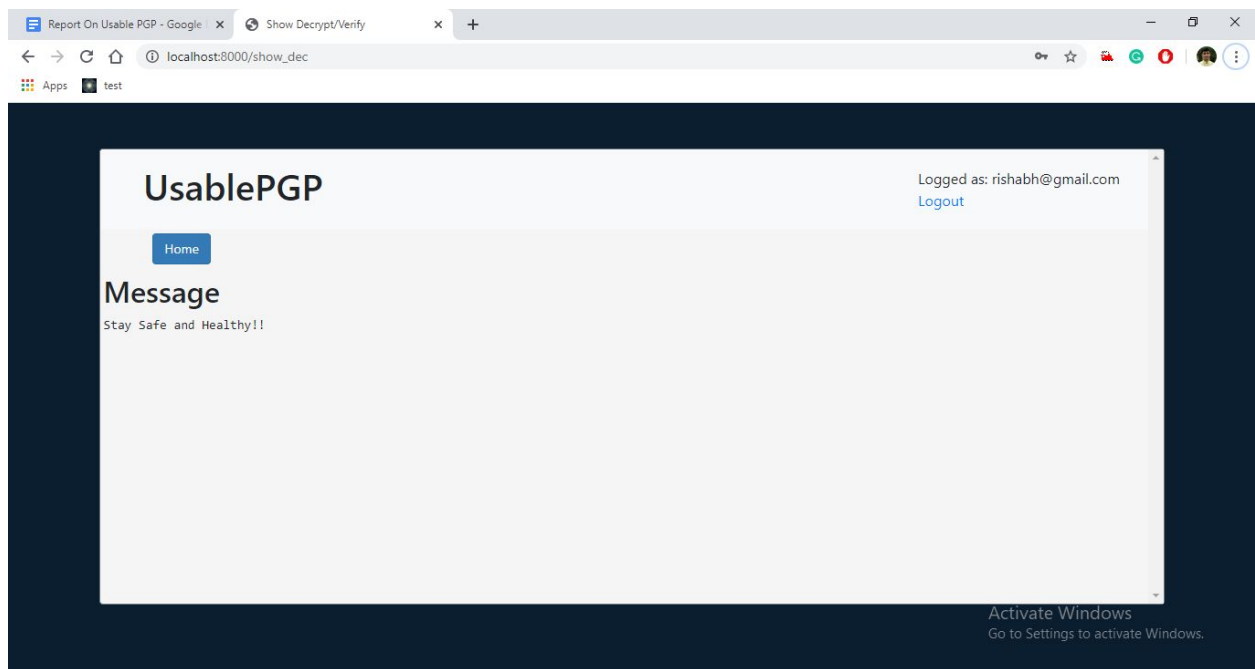
The user chooses among decryption, verification of signature or both. Then, he inputs the message he has received in text or file format. If the user chooses decryption, he is required to enter his passphrase. If verification is required, the user is required to enter the user who has sent him that message. The resultant operation(s) is/are performed on the input, then the decrypted text and result of signature verification is displayed to the user.

Background Process:

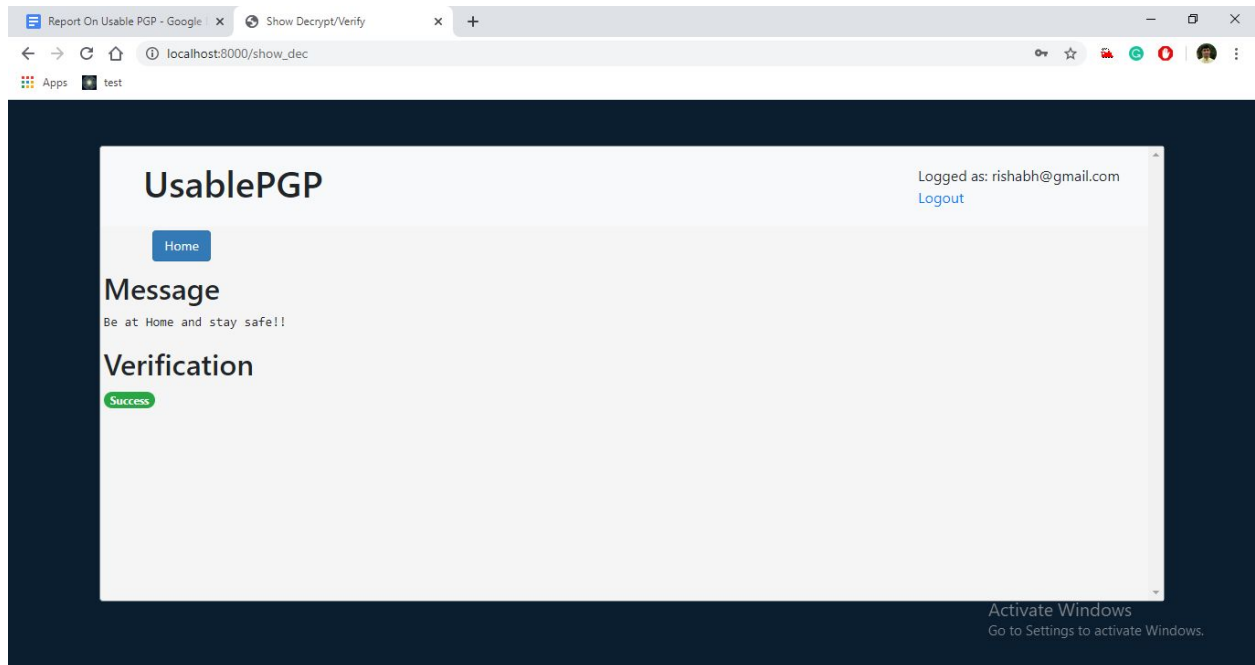
1. If the user chooses to verify the signature, then the user is requested to enter the sender of the message. Then, if the signature is present, then it is verified using the sender's public key. If correct, success is shown. Otherwise failure is shown.
2. If the user chooses to decrypt the message, then the user is requested to enter his/her passphrase. If the passphrase is correct and the message is encrypted using the user's public key, then the message is decrypted using the user's private key.
3. Final result is shown to the user on the next page.



Decryption/Verify



Decrypted Message



Decryption and Verification

4. Revocation and Regeneration

The user chooses between revocation and regeneration. After choosing, the following happens:

Revocation:

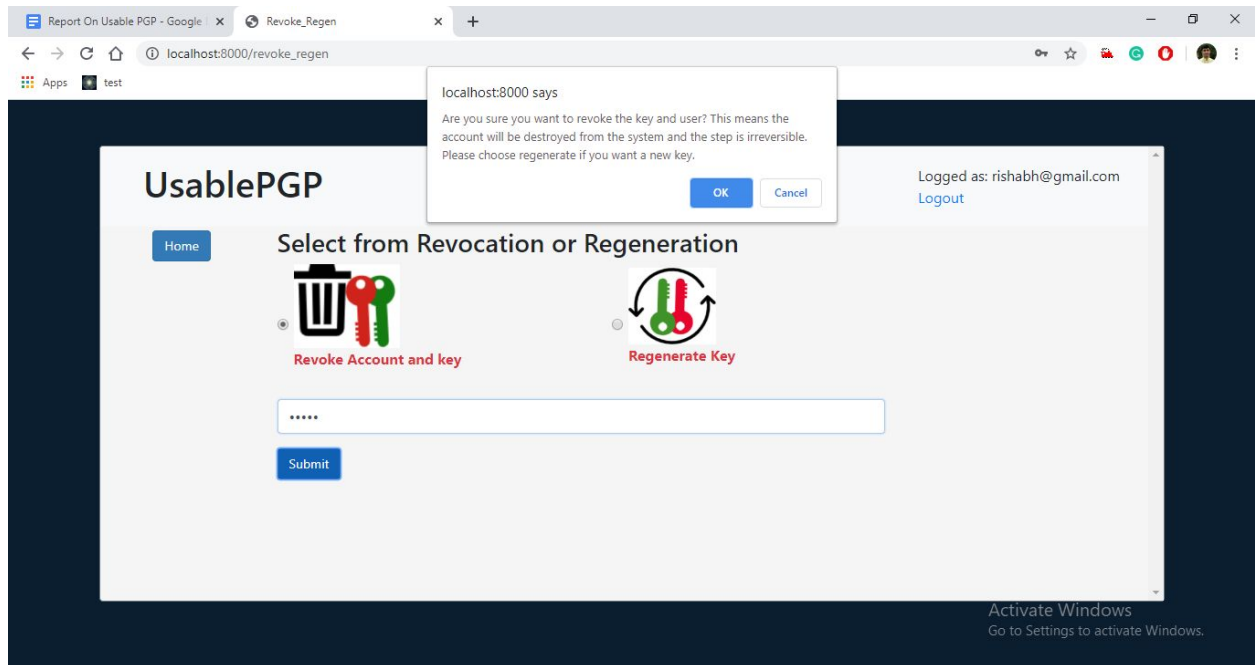
1. The user is required to enter his passphrase to verify his identity.
2. The user is warned of its dire consequences.
3. On choosing confirm, user and key both are removed from the system.

Regeneration:

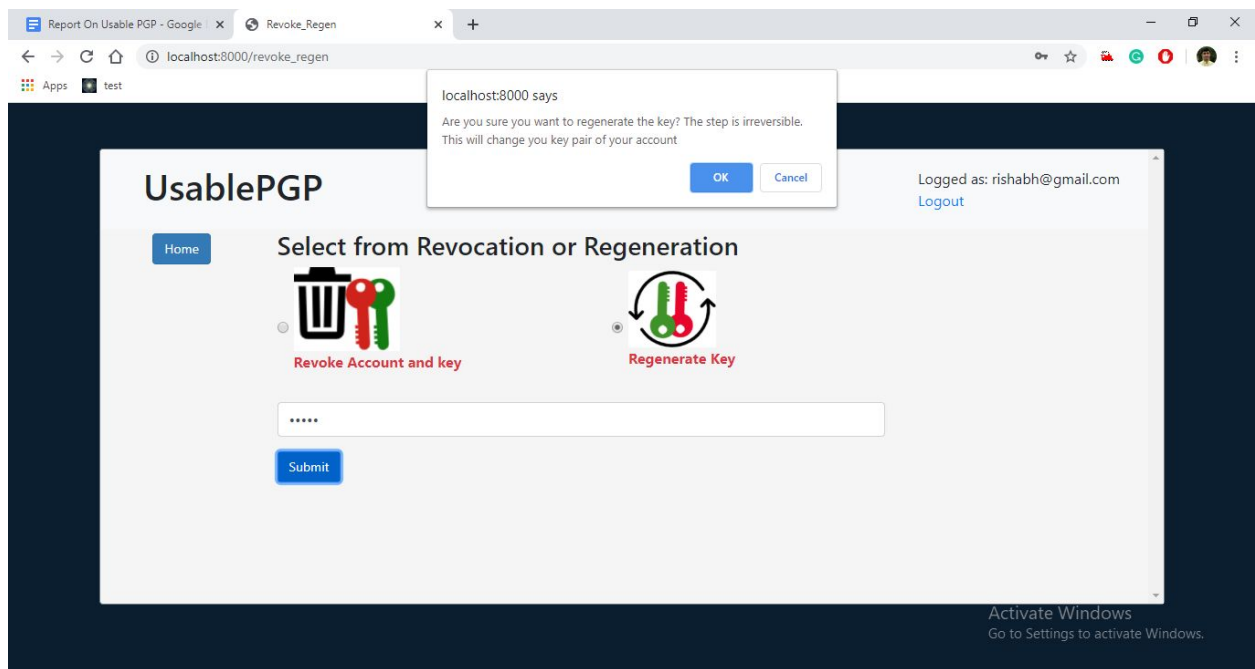
1. The user is required to enter his passphrase to verify his identity.
2. The user is warned of its dire consequences.
3. On choosing confirm, the current key is revoked and a new key is generated for the user.

Background Process:

1. After completing the above steps, the passphrase's hash is checked with the user's passphrase stored hash.
2. If matched, further steps are taken.
3. In case of revocation, the user and his public key is removed from the key server. The private key is removed from the local system.
4. In case of regeneration, a new key pair is generated. Public key is updated on the server and the private key is updated on the local PC.



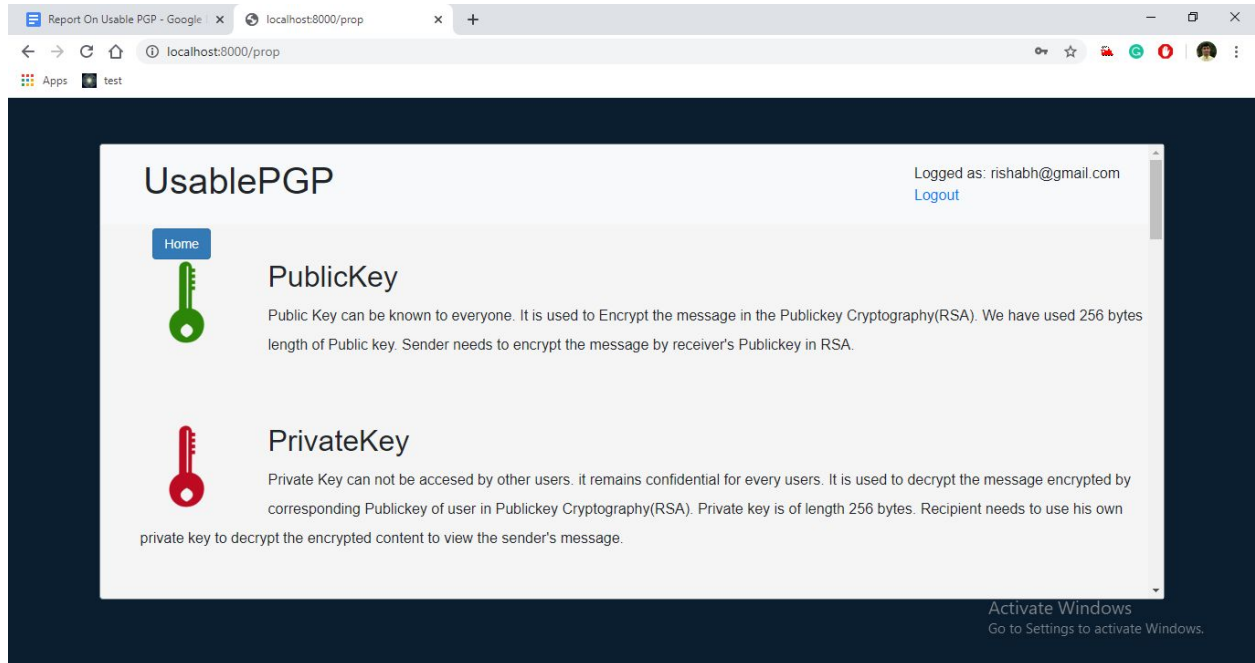
Key Revocation with alert



Key Regeneration with alert

5. Properties:

The user can see the system properties, key information and available users' information in the Properties window.



Key properties

Suggestions Included

1. Icon Visualization
2. Signature Metaphor
3. Verification metaphor
4. User Finding
5. Separate Key Servers
6. Complete Key Revocation
7. Accidental deletion of private key is somewhat made secure by hiding it safely on local PC.
8. Accidental Publicizing key is resolved by providing a regenerate option.
9. Accidental revocation is not possible as we are ensuring the user knows all the consequences of its actions.
10. Consistency is maintained across the app.
11. Information useful to the user is provided under the Properties option.
12. Ease of use for users is increased through Preview Tours and Tooltips.

Non included Suggestions

1. Identify key not user.
We provided users with only one key. Hence, this was not needed at all. Essentially, a username/email has only a single key attached to it.
2. No solution for forgetting the passphrase.
As we have not used third parties to authenticate users. So, he is not allowed to reset his passphrase. [The user can utilize a password manager.]
3. Backup key rings
Public key is stored on the server and the private key is hidden from the user. The only case of losing the keys is when the user formats his computer or loses his pc. Since, the user is warned all about the warnings, hence he is not allowed backup and recovery of revoked keys. [Given Time, we can let users to backup their private keys on a secure usb. However, we feel this is more of an advanced feature as novice users would not be tinkering with the app internals. And, if a user loses his pc, he can login on our website and revoke his keys.]
4. Server Access Record
Although this is a good stat to have but it is not necessary to implement. So, skipped it.
5. Trust
Since, the server is secure and the private key is encrypted using passphrase. So, we think the system is trustable and hence trust feature is not included. [Given time, we can verify if a user has access to his email and then sign his key using ours. Trust can then be computed using standard pgp procedure (web of trust)]