SecAuth: Stronger authentication with keystroke features

Report by: Sanya Ralhan (903248565)
Ravi Prakash Giri (903238554)

To build:

Command: sudo python main.py <input_filename>
For error correction: sudo python main.py <input_filename> -e
Note: only one input file should be run at one time.

Requirements:

python 2.7.x and pip
simple-crypt 4.1.7
pycrypto 2.6.1
numpy 1.10.1

Initialization:

- As per this step we select a random 160 bit long prime q (generated using Crypto.util.number.getPrime(), h_pwd<q and h = 5 (we store features for the first five attempts and start checking from the 6th login)
- We generate a random polynomial ie (coefficients are randomly selected between 1 to 100) of order m-1 where m is size of password (we have assumed a max size of password as 25)

Creation of Instruction Table:

To avoid small number subtraction problem we are subtracting 0.001 in deciding whether feature is distinguishing or not in $|\mu - ti| > k\sigma$
We have list which puts the calculated α and β values as per paper ie (SHA hash function is used for hashing)
For slow feature:

α =alpha_cal(pwd, i+1, poly)
β = beta_cal(pwd+str(random.randrange(0, 1000)), i+1, polynomial_gen(max_features-5, random.randrange(0, q_val-1)))

For fast feature :

α = alpha_cal(pwd+str(random.randrange(0, 1000)), i+1, polynomial_gen(max_features-5, random.randrange(0, q_val-1)))
β = beta_cal(pwd, i+1, poly)

For undistingushable feature

α=alpha_cal(pwd, i+1, poly)
β=beta_cal(pwd, i+1, poly)

Basically for each features, if it slow then α is valid and β is random and if it fast then β is valid and α is random.
Note: Maximum password length is 25 but if the actual password is smaller say 13 characters long the instruction table is created only for m-1 features.

Login attempt:

- If feature is fast , we select β and α (slow or undistinguishable) otherwise.
- X,y values are derived using the value selected from above step.
- New h_pwd is calculated using lagrange interpolation and the above selected x and y values.
- Without error correction is file doesn't decrypts correctly print 0 else 1.
- All values are reinitialized ie q, polynomial and instruction table id recalculated based on the average of feature values coming from history file +current features. Current features are written into history file.

Error Correction:

We have a flag that is used to determine if error handling should take place or not. Whenever this flag is set in the input command, error handling will occur. If that is set then for each feature we pick the β value if α was picked before in the previous login and vice versa. So basically we try to correct one feature at a time and decrypt the file with that and if it succeeds for any feature we print 1(success) else 0. This reduces false negatives if the difference in 1st and 2nd login is just of one feature. If there are more than one features incorrect that is considered as an unsuccessful attempt.

History File:
History file is stored to disk only after encryption. Size of history file is fixed to 500 and padded with S character and “$$$$” is used as a redundant separator between the padding and file text. Hence size of history file is fixed at the start of the program as 500 +(size of 5 rows of m-1 features).SHA of h_pwd is used as key to encryption. This is used to retrieve data after decryption Encryption is done using simple-crypt-4.1.7 library functions.

Input Validation:
if length of password entered >25 or number of features is not m - 1 where m is password length, the program exits

Exception Handling:
We are handling File decryption errors by marking the login attempt as 0 and continue with the next input line