Modernizing Password Usage in Computing

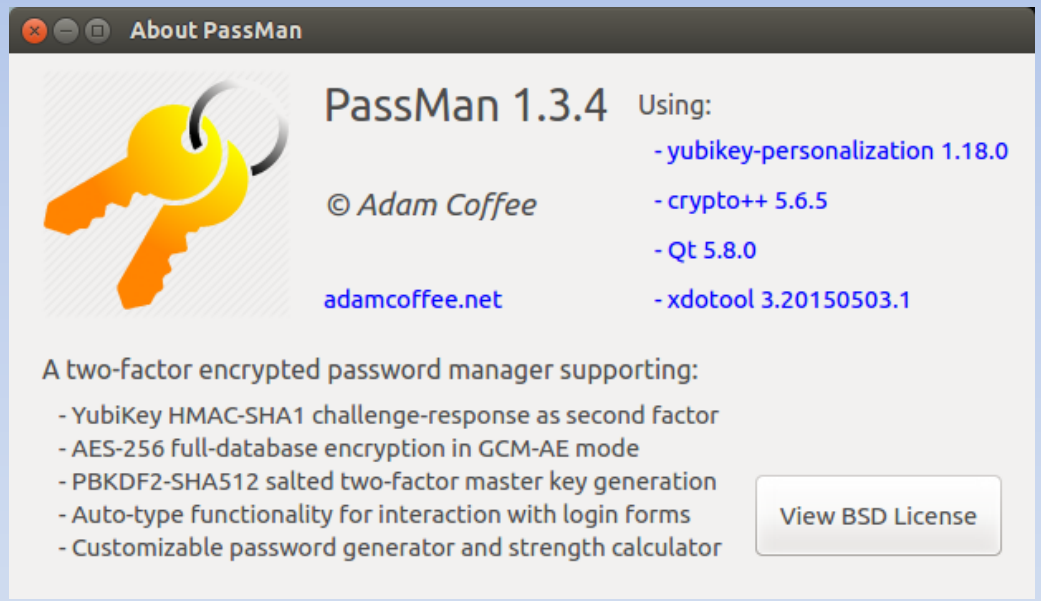
Adam Coffee

adamcoffee.net

May 5, 2017

Overview

- Motivations
- Goals
- Background
  - Algorithms
  - Techniques
- Usage
  - Demonstration
- Implementation
- Questions



Motivations

- Passwords are used *everywhere*
- Users are responsible for choosing and managing their collections
- Inherently difficult to remember strong passwords

Motivations: Core Issues

- Using similar passwords across accounts aids memory, but endangers account security
- Weak passwords are simply easier to use, but may be cracked faster
- Secure storage is ideal, but can be quite a hassle

Goals

- Remember user account information
  - Ensure *confidentiality* and *integrity* of all data
  - Require *two* authentication factors for recovery
    - Something they *have* and something they *know*
- Generate strong passwords
- Calculate password strength
- Automate login processes (auto-type)

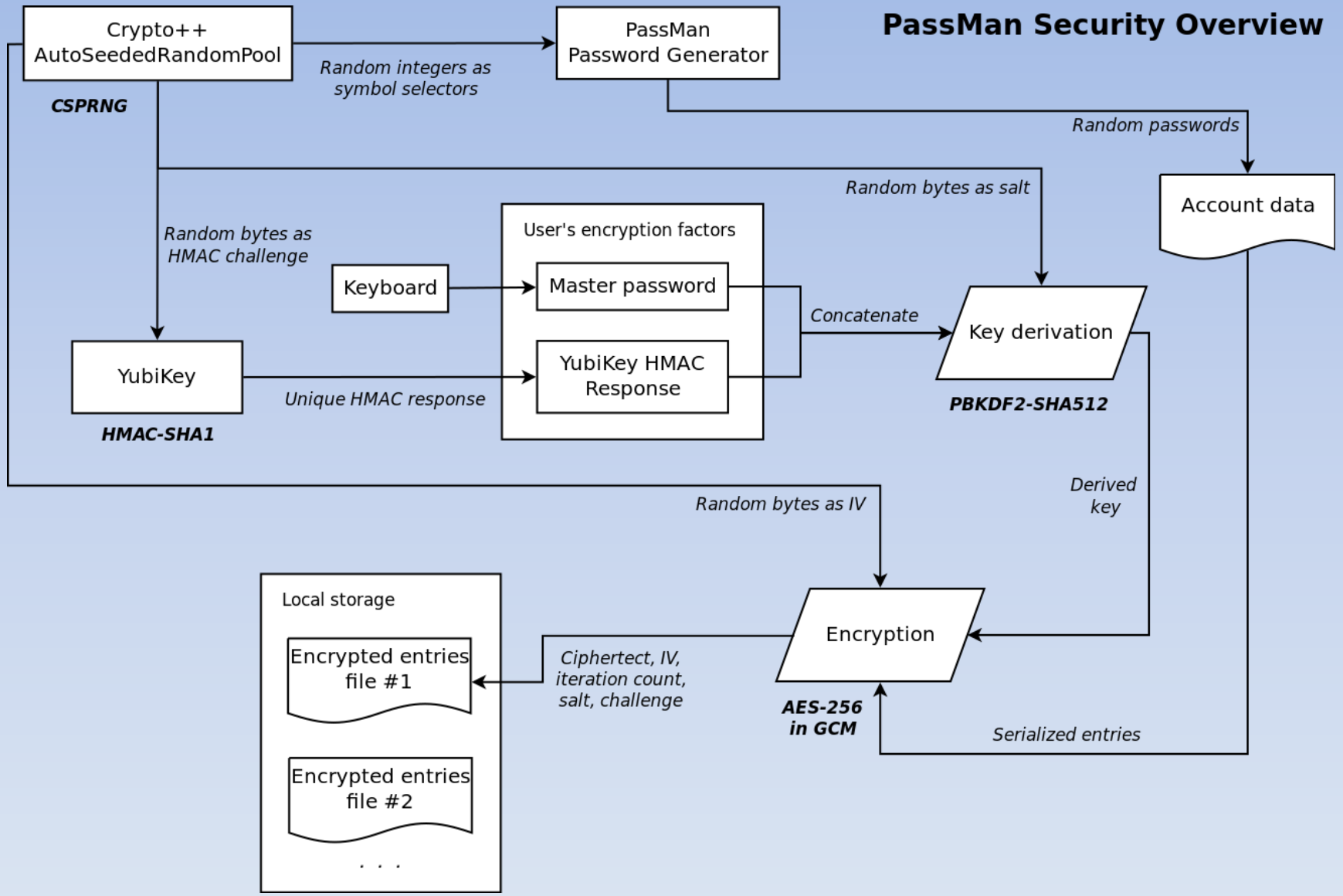
Abstract the user away from the continual processes of password creation, management, and storage

Background: Overview

An alphabet soup of algorithms…

- Advanced Encryption Standard (AES) with Galois/Counter Mode (GCM)
- Password-based Key Derivation Function (PBKDF)
- Cryptographically Secure Pseudo-random Number Generator (CSPRNG)
- Hash-based Message Authentication Code (HMAC)
- Secure Hash Algorithm (SHA)

Background: Overview



Background: Libraries

- Unwise to ‘roll-your-own’ cryptography
- Crypto++ library provides most schemes
  - Written in C++, in the public domain
  - NIST-validated under FIPS 140-2, no security issues
  - Dynamically linkable
- Yubico YKChalResp software does the rest
  - Open source via BSD license
  - Communicates with YubiKey token
  - Programmatically execute binary in separate processes

Background: YubiKey

- Hardware authentication device over USB
- Supports HMAC-SHA1, among other authentication methods
- Models have passed FIPS 140-2 with highest assurance
- Also an OpenPGP smartcard!

Yubico's YubiKey NEO

Background: AES and Galois/Counter Mode

- AES is a well-researched, strong symmetric block cipher
  - NSA-approved for *confidentiality* at top-secret
- GCM is a mode of operation for block ciphers
  - Authenticated encryption for *integrity* assurance
    - Authentication tag signals corruption of ciphertext
  - Allows AES to be used with arbitrary-length data
- Initialization vector (IV) ensures random ciphertext regardless of cleartext

Background: Hash-based Message Authentication Code

$$HMAC(K, m) = H((K' \oplus \text{opad}) \,||\, H((K' \oplus \text{ipad} \,||\, m))$$

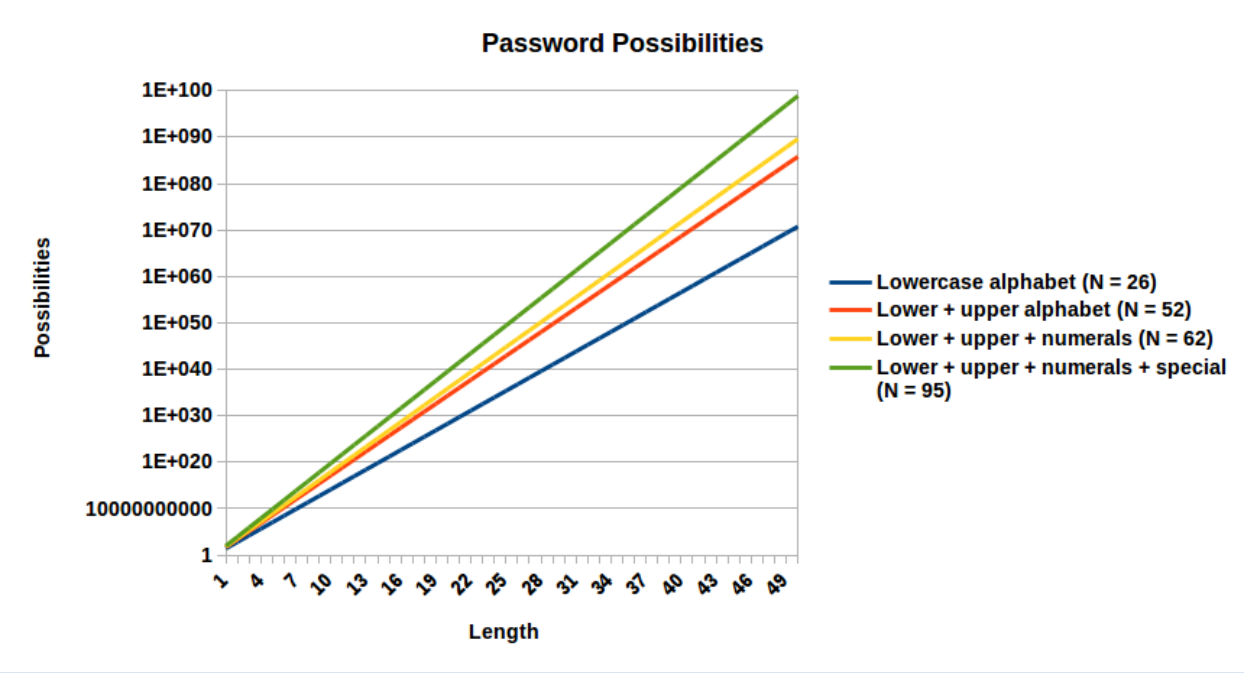
- Confirms authenticity of message m with key K with hash function H (after some padding)
- Second authentication factor from YubiKey in PassMan
- Standardized in RFC 2104
- Also used in IPSec and TLS protocols
- Reliable despite collision attacks on SHA1 and MD5
  - Proven that collision-resistance not required

Background: Password-based Key Derivation Function

- Transforms passwords into suitable encryption keys
- Described in RFC 2898
- Mitigates vulnerabilities with passwords
  1. May brute-force password with *fast* hardware
  2. Accelerate brute-force with pre-computed ‘rainbow tables’
- Applies random ‘salt’ to password, disabling (2)
- Repeatedly applies hash function to input, slowing (1)
  - Known as ‘key stretching’
  - Time-tunable (PassMan aims for 0.5 second derivation)

Background: Password Strength

- A concatenation of symbols from some set
- Ideally chosen via uniform probability distribution
- Unfortunately, users produce low-entropy passwords
  - Birthdates, common english words, simple substitutions…



$$H(L,N) = \log_2 N^L = \mathrm{L} * \log_2 N$$

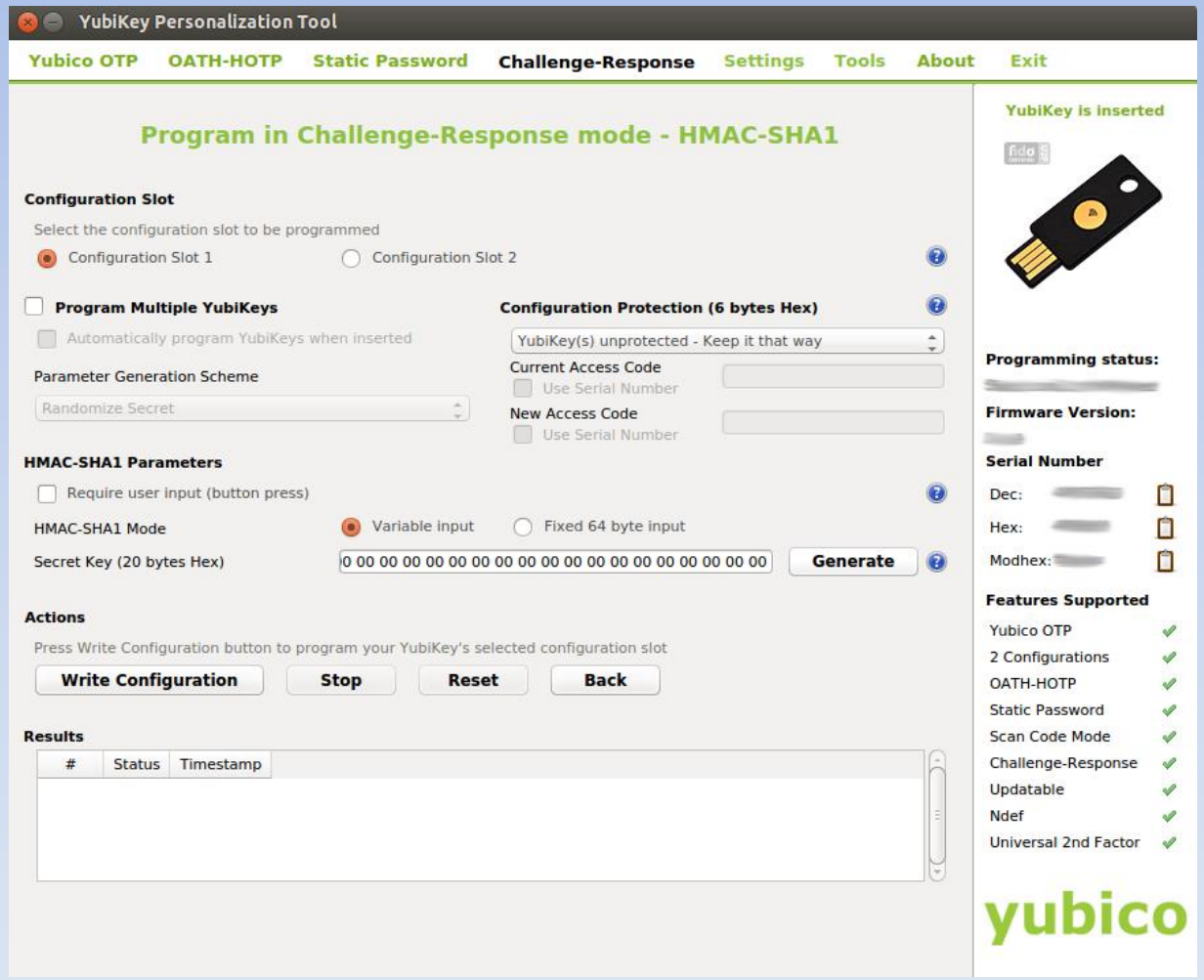
Information entropy in bits

Usage: Security

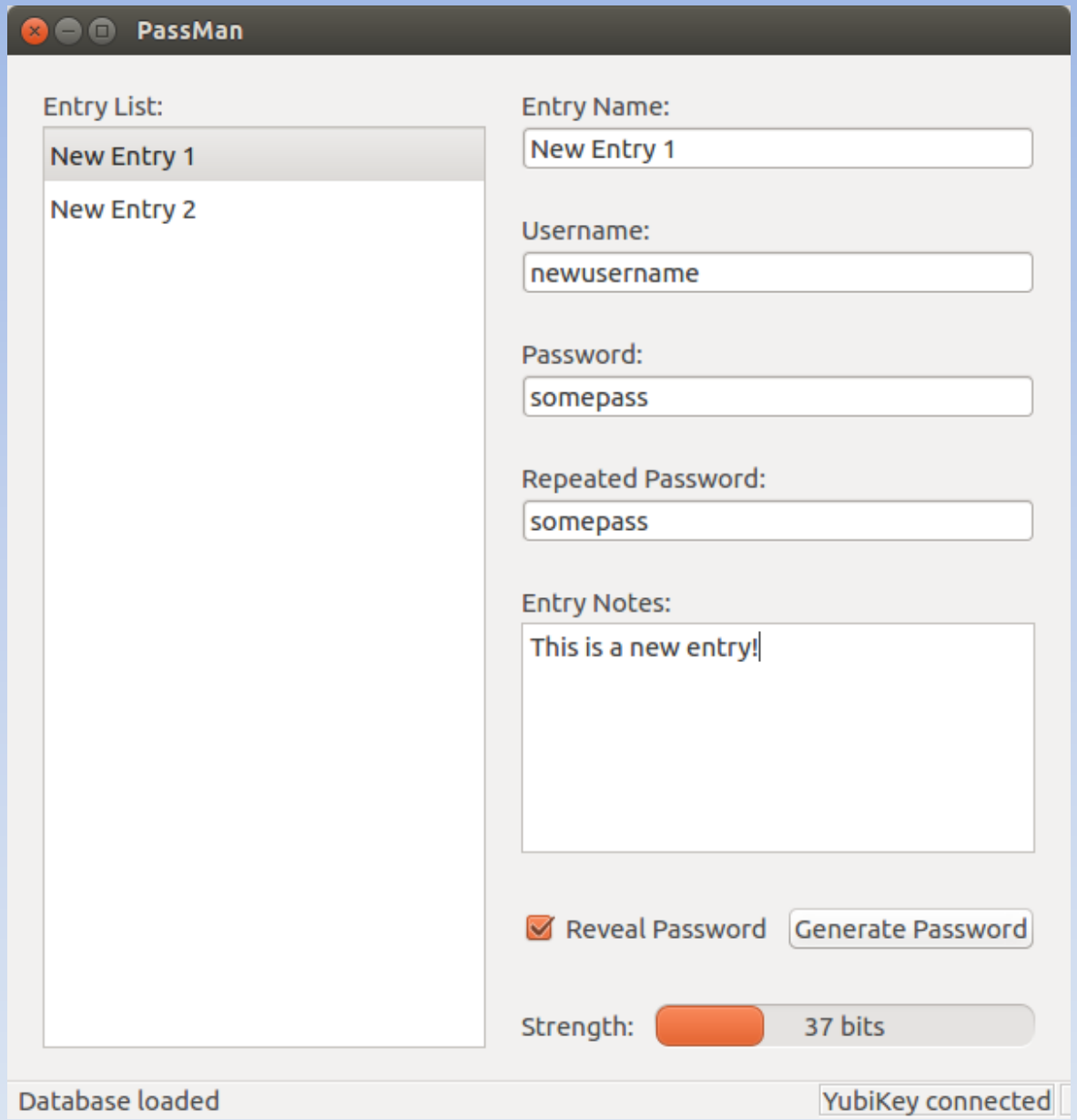
- All account information encrypted
  - Account names
  - Usernames
  - Passwords
  - Associated notes
- Two factor used to encrypt
  - Master password (something the user *knows*)
  - YubiKey HMAC response (something the user *has*)
- Make your master password strong!
  - It is now the only one you must remember

Usage: YubiKey Configuration

- Use Yubico's YubiKey Personalization Tool to set a unique HMAC key for Challenge-Response mode

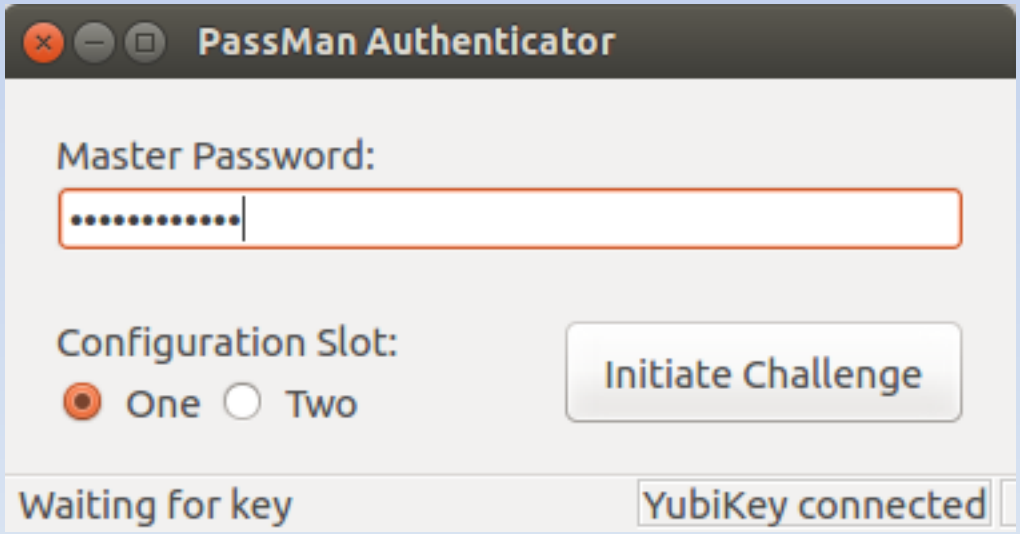


Usage: Entries File Creation

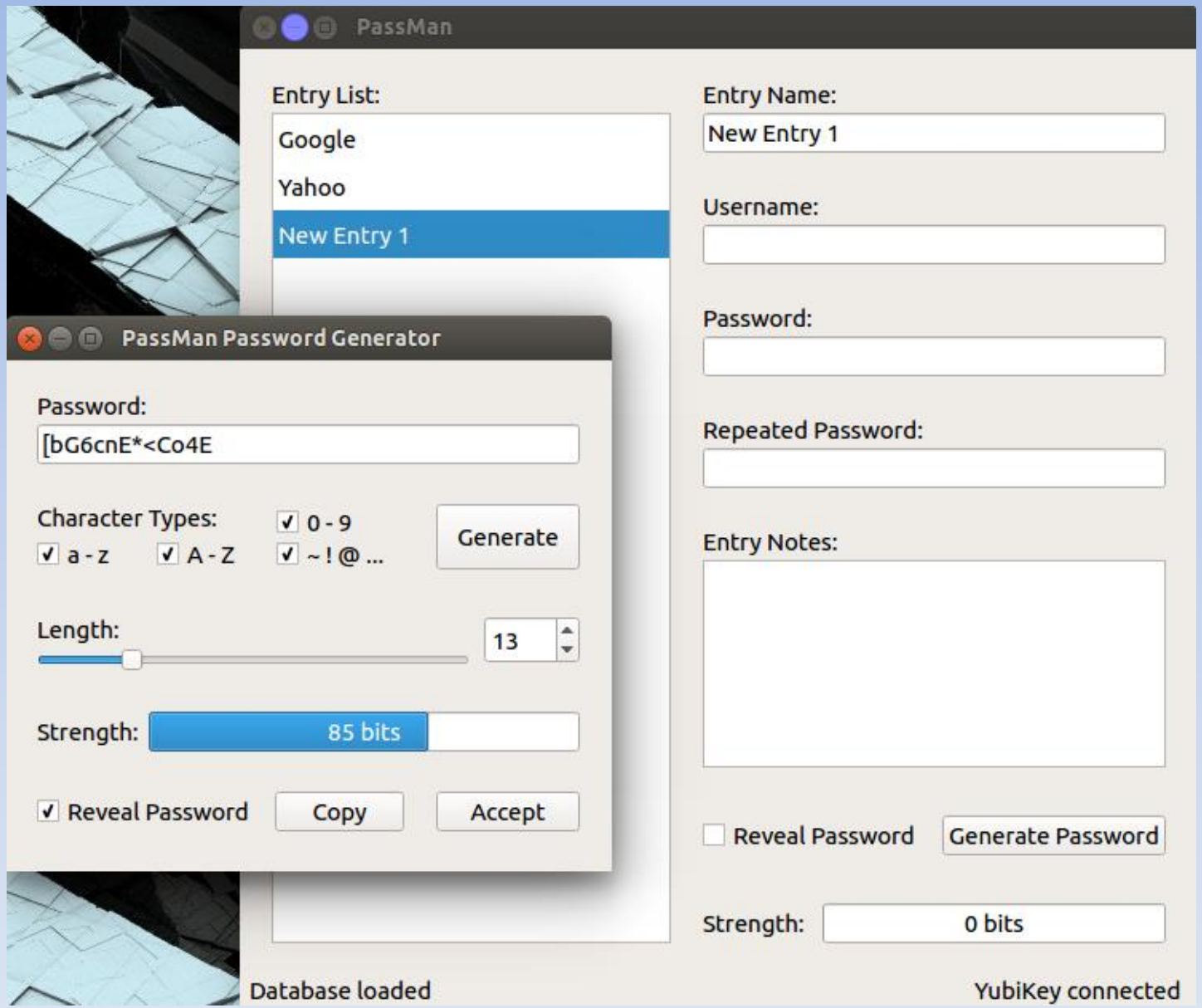


Usage: Saving

- Select $File \rightarrow Save\ Database$ to initiate encryption
- Enter password into authentication window
- Begin YubiKey challenge with $Initiate\ Challenge$
- The key is derived from both factors, the data encrypted, and written to storage



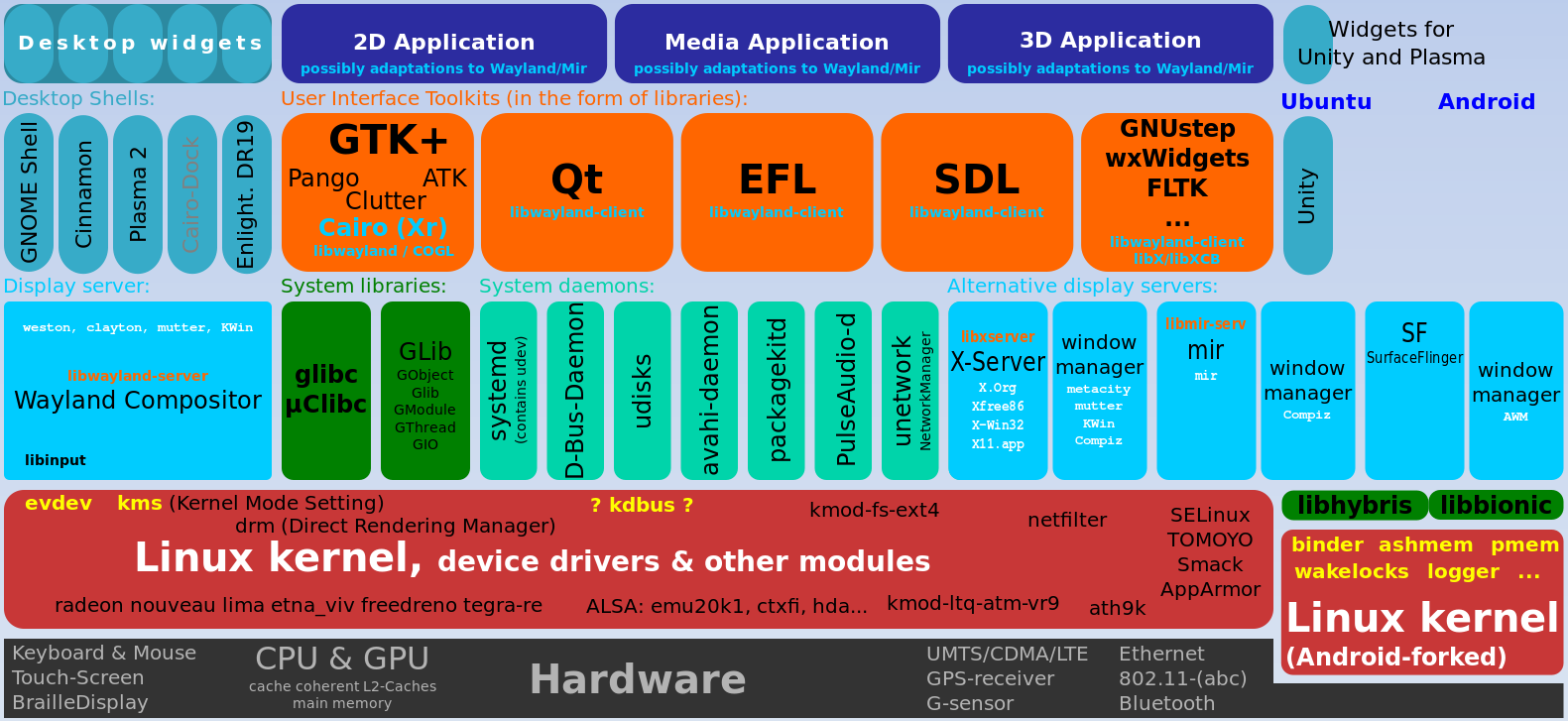
Usage: Password Generation



Demonstration

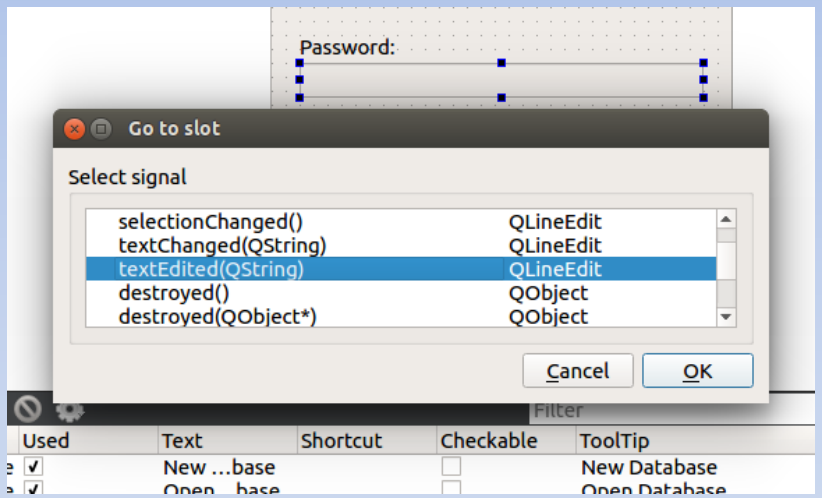
Implementation: Development Framework

- Qt (pronounced ‘cute’) used from The Qt Company
- Open-source via LGPL 3.0, cross-platform compatible
- Available for C++ or Python



Implementation: Observer Pattern Design

- Qt's language construct 'Signals and Slots' handles asynchronous events and avoids boilerplate code




```
1  void PassMan::on_passwordLineEdit_textEdited(const QString &arg1)   // Update entry
       password if changed
2  {
3      int strength = StrengthCalculator::naiveEntropyBits(ui->passwordLineEdit->text
           ());
4      if (ui->passwordStrengthBar->maximum() < strength) ui->passwordStrengthBar->
           setMaximum(strength);
5      ui->passwordStrengthBar->setValue(strength);
6      updatePasswords();
7  }
```

Implementation: Account Management

- Serialize/deserialize account information to/from JSON prior to encryption/decryption
- Database class manages instances of Entry class for each user account

```
1  void Database::read(const QJsonObject &json) // Extracts entry information from
       JSON object
2  {
3      entries.clear();
4      QJsonArray entryArray = json.value(ENTRIES_KEY).toArray();
5      for (int i = 0; i < entryArray.size(); i++)
6      {
7          QJsonObject entryObj = entryArray.at(i).toObject();
8          entries.append(new Entry(entryObj.value(NAME_KEY).toString(), entryObj.
               value(USERNAME_KEY).toString(),
9                                   entryObj.value(PASSWORD_KEY).toString(), entryObj.
                                        value(NOTES_KEY).toString()));
10     }
11     version = json.value(VERSION_KEY).toString();
12     emit readNewData(); // Notify watchers that database is loaded
13 }
14
```

Implementation: File Format

PassMan Database File

File extension: .pmdb
Encoding: Base64 (colon-separated sections)

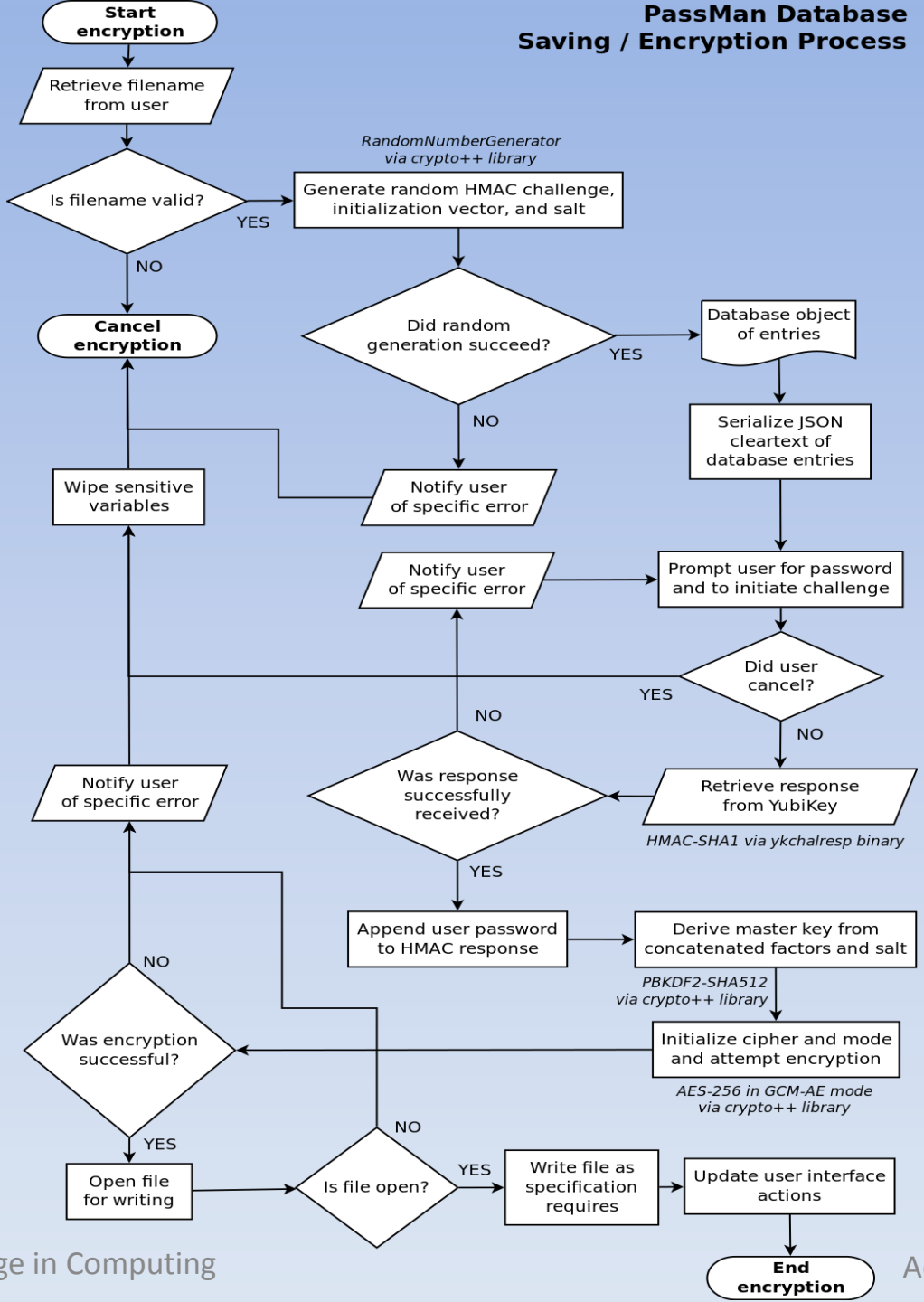
64 bytes
Challenge for HMAC-SHA1

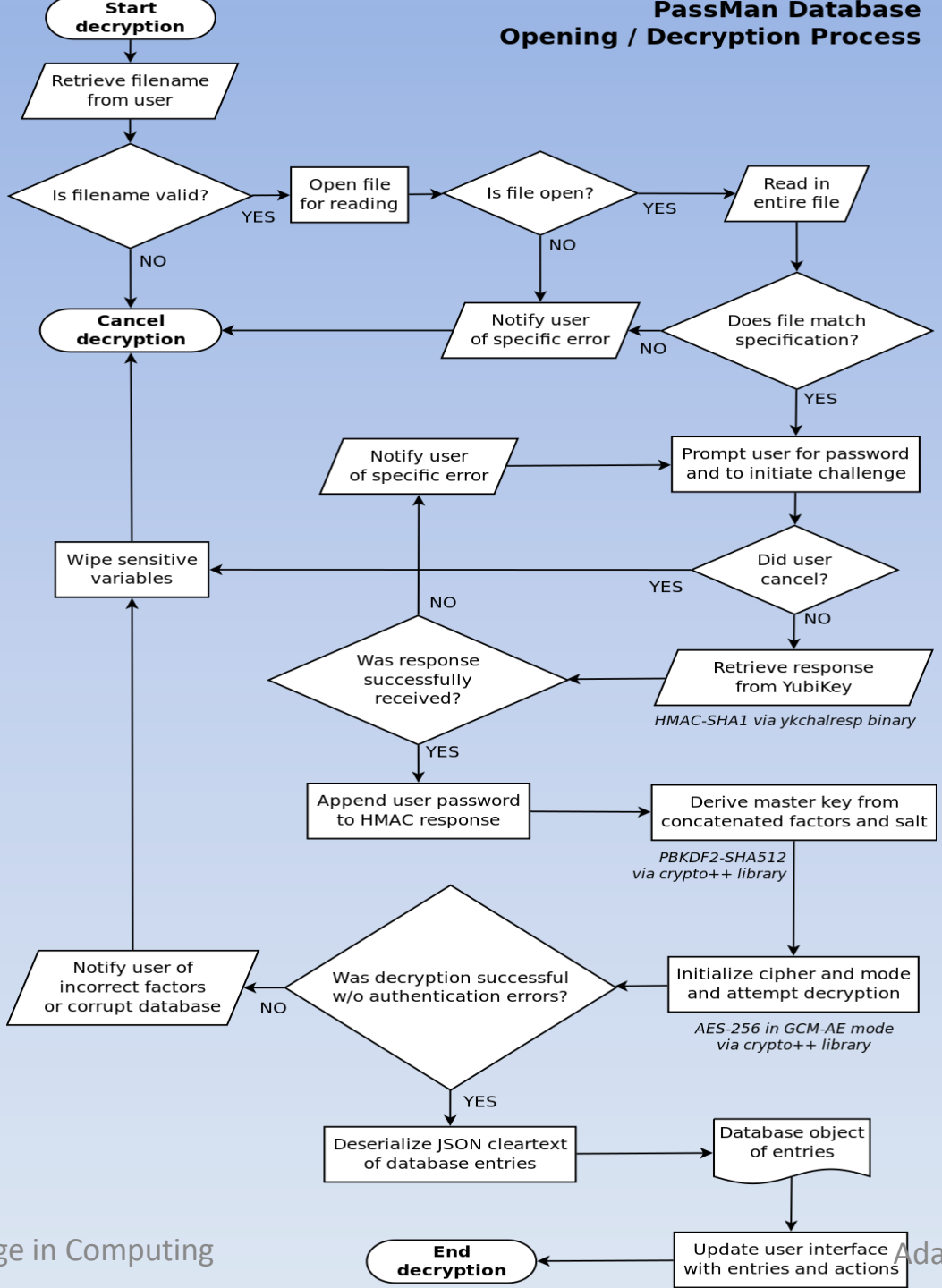
16 bytes
Salt for PBKDF2-SHA512

4 bytes
Iteration Count for PBKDF2-SHA512

256 bytes
Initialization Vector for AES-256 in GCM-AE mode

Variable bytes
Ciphertext of JSON-serialized database entries


PassMan Database
Saving / Encryption Process
Start encryption
Retrieve filename from user
Is filename valid?
YES
NO
Cancel encryption
RandomNumberGenerator via crypto++ library
Generate random HMAC challenge, initialization vector, and salt
Did random generation succeed?
YES
NO
Database object of entries
Serialize JSON cleartext of database entries
Notify user of specific error
Wipe sensitive variables
Notify user of specific error
Prompt user for password and to initiate challenge
Did user cancel?
YES
NO
Was response successfully received?
NO
YES
Retrieve response from YubiKey
HMAC-SHA1 via ykchalresp binary
Notify user of specific error
Append user password to HMAC response
Derive master key from concatenated factors and salt
PBKDF2-SHA512 via crypto++ library
Initialize cipher and mode and attempt encryption
AES-256 in GCM-AE mode via crypto++ library
Was encryption successful?
NO
YES
Open file for writing
Is file open?
NO
YES
Write file as specification requires
Update user interface actions
End encryption


PassMan Database
Opening / Decryption Process
Start decryption
Retrieve filename from user
Is filename valid?
YES
NO
Open file for reading
Is file open?
YES
NO
Read in entire file
Cancel decryption
Notify user of specific error
Does file match specification?
NO
YES
Notify user of specific error
Prompt user for password and to initiate challenge
Wipe sensitive variables
Did user cancel?
YES
NO
Was response successfully received?
NO
Retrieve response from YubiKey
HMAC-SHA1 via ykchalresp binary
YES
Append user password to HMAC response
Derive master key from concatenated factors and salt
PBKDF2-SHA512 via crypto++ library
Notify user of incorrect factors or corrupt database
NO
Was decryption successful w/o authentication errors?
Initialize cipher and mode and attempt decryption
AES-256 in GCM-AE mode via crypto++ library
YES
Deserialize JSON cleartext of database entries
Database object of entries
End decryption
Update user interface with entries and actions

Implementation: Password Generation and Strength Calculation

- Use random integers from CSPRNG to select symbol for each position along the length of the password
- Character sets split for compatibility with services
  - Lowercase alphabet
  - Uppercase alphabet
  - Numbers
  - Special symbols
- Utilize aforementioned entropy formula to calculate strength

Implementation: Auto-Typing

- Use ‘xdotool’ binary (available in system repos) to simulate keystrokes
- Execute in separate process after minimizing PassMan window to return focus to desired application
- Send username, Tab, and password to STDIN of xdotool process
- Finish with a simulated *Return* keystroke

Questions?