License Key Manager

Capstone Project | FEUP | 2022

André Machado
up201706280

Daniel Gonçalves
up201809384

Isla Cassamo
up201808549

Table of Contents

1. Introduction

Handling license models is a necessity for software distributors. Be it the generation of new licenses or the publishing of new products, it is crucial to keep things centralized into one single application. Regardless of the provided limits and various options, licensing your software must be a straightforward task.

Our Key License Manager aims to achieve this by creating a web application that handles the distribution of software. To achieve that, we focus on a readable interface that offers its managing capabilities behind a small learning curve. We accomplish that without sacrificing features and keeping the application as efficient as possible.

1.1. Background

Nowadays, software development is a perennial occurrence. A partial statistical reference **[1]** indicates that the revenue of O.S and Productivity software publishing has hit 133 billion US dollars just in the year of 2021. This is a remarkable inference as it proves that the Software industry continues to be an ever-evolving ecosystem. But these numbers fail to show the dark background.

According to another reference **[2]**, the commercial value of pirated software is worth $19 billion in North America plus Western Europe. This value grows to $27.3 when talking about the rest of the world. 37% of the software installed on personal computers is unlicensed. These statistics damage the software industry and a general solution towards “cracking” is still far from being achieved.

Software or Operating Systems with offline validators such as Windows XP, are defenseless against these practices. More recent hardcore solutions demand a consistent online connection to ensure the software does not get devalidated. Denuvo came up as a recent solution towards anti-tampering with Software, but it’s not bulletproof, with the company eventually confirming that every software or game is bound to be cracked eventually. Other games and software have managed to avoid this by introducing an entirely online-based network, which is persistently validated.

It’s close to impossible to prevent the evasion of these validations but online-based validations are seen as a crucial tool to contain these acts.

1.2. Our Proposal

Our project seeks to satisfy two crucial goals.

First, we want to let companies manage their licenses in an organized and efficient way. The interface must not be complex or obsolete, and we want the learning curve to be small enough to make it efficient. In addition, the project offers administrative tools that let administrators monitor most changes.

Secondarily, the web app must aim to solve some of the problems mentioned previously. This online-validation solution covers some of the vulnerabilities of an offline validator while keeping everything contained in a pseudo-private web app environment.

It is vital to protect the software and facilitate license management. Our project permits multi-product management to introduce scalability; this allows larger companies to manage the license of multiple products, each with its differences.

In addition to that, each generated license contains information about its assigned client. Finally, these same licenses are manageable, with each change registered into a changelog to track any unusual activity.

2. Development Process

Initial meetings were used to set-up the foundation of our work. As such, the team decided to implement the web application using the Flask microframework, written in Python. We kept our work versioned and worked together using Github.

During these moments we also acknowledged some of the user stories and requirements that defined our Minimum Viable Product (MVP). These were:

- **Conception of a minimal CPanel (Control Panel)**

  Administrators should be allowed to manage their own licenses as easily as possible. The information should be available to them behind a comfortable display that they can browse through fluently.

- **Products and Licenses**

This defines the multi-purpose state of our web application. Our license manager software should allow administrators to introduce new products into their database. Each license will then be bound to a specific product.

- **Client communication**

  Administrators have their management area, which is given to them should they manage to log into the CPanel. Clients, however, should not partake in this login procedure. Instead, a public endpoint (route) should be available to them, where they can validate their own license keys.

- **Communication encryption**

  Products should have their own unique API, Public and Private keys. Additionally, each license should have a unique serial number. The encryption is done on the client side with the Public key and API key, and it's the server's responsibility to decrypt this information and then handle the request accordingly.

With the definition of our MVP, the team started working on the project. The web application went through multiple feedback reports over the following weeks, things that were considered in order to progress with our work.

The development cronogram below (and its extended description) states each step taken by the team in order to achieve the development of the application.

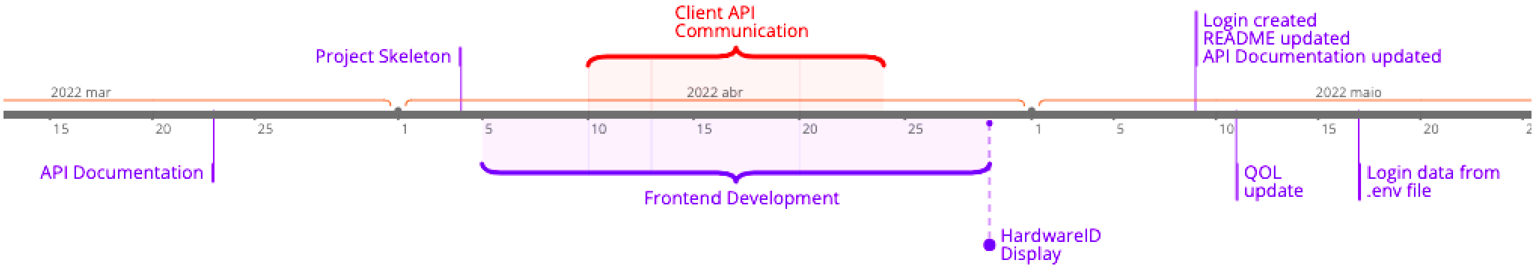

Fig.1 - Development cronogram

23 Mar, 2022 | API Documentation:

Preparation of a mockup RESTFUL API that will serve as the base for our app.

4 Apr, 2022 | Project Skeleton:

Commit of the default repository of a typical FLASK-auth + SQLite application.

5 Apr, 2022 - 29 Apr, 2022| Frontend Development:

Throughout this stage, multiple iterations and improvements of the three CPanel pages (main page, product page and license-key page) were made.

Although it's not listed, the Database matured throughout this step.

10 Apr, 2022 - 24 Apr, 2022| Client API Communication:

This stage was focused on the development of a client mockup, simulating a validation request to the server. We sought encryption tools and augmented the Database with tables to support this step.

29 Apr, 2022| HardwareID display:

The license-key page of the CPanel was augmented with additional information, allowing the administrator to see assigned HardwareIDs to the license key.

9 May, 2022| Large-scale update:

The web app was updated with a login feature (locking the CPanel functions to authenticated users only). Additionally, the README.md file was updated with crucial information about the project and the initial API documentation was reworked on to match the final result of the application.

11 May, 2022| QOL (Quality of life) update:

The web app was changed to automatically generate a new database if the SQLite file did not yet exist (or was deleted). It also started to automatically insert the login data of the administration in the database if such data did not yet exist.

11 May, 2022| Limited licenses:

The license-keys were updated to have an optional expiration date. If it exists, then the key will stop being valid after the expiration date is no longer valid. The expiration date is based on the POSIX timestamp.

17 May, 2022| Login data updated:

The web application now loads the login data for the administration account from the .env file, rather than having it hard coded into the main.py file.

3. Application

3.1. RESTful API

The application uses the FLASK microframework **[3]** (version 2.1.2), written in Python 3.9 **[4]** which is based on a RESTful API format. Before the implementation of each service route, we developed a mockup for our groundwork. As of the final version, the API only handles GET and POST HTTP Methods and is specified in detail below.

You can check the details of our API in the **Resource 7-A** annex which contains the details of each service route and all essential parameters for each request. The HTTP Method is also indicated along with the login requirement.

3.2. Authentication

A crucial point for the development of a secure license management system lies in a robust and organized communication structure.

As seen in Section 3.1, most paths of the HTTP Requests are locked behind an authentication requirement. Modifying, adding or even deleting contents from the Database are privileged commands that must be safeguarded with authentication.

We implemented this requirement using the **Flask-Login** component for session management. The session data is stored in the Database and the credentials are defined in the environment file on the root directory. *This is the only way of defining credential access seeing that the creation of new users is not a supported feature*.

According to the initial specifications, creating new users was not useful at all and therefore only one active user is allowed at a time.

3.3. Validation

The validation endpoint is the only endpoint that modifies existing Database contents and is available for non-authenticated users. Unlike other endpoints, the client does not need a browser to validate as the request is done by the software to this specific route.

In order to confirm the identity of any client that wishes to activate their own license, they are required to provide the following parameters:

- Serial Key
- API Key
- Public Key
- Hardware ID

The *Public Key* is associated with the specific product the client has purchased and is unique for every product. It is used to encrypt the payload before sending it to the server.

The *API Key* is also unique to each product and is used to prevent DDoS attacks.

The *Serial Key* identifies the license the customer has purchased. It has to be unique regardless of the product. This means no two products can have the same Serial Key.

The Hardware ID is likewise unique and depends on the components of the client's machine.

Assuming the four parameters are in check, the communication process is done through the following steps:

1. The user's software will encrypt the Hardware ID and Serial Key parameters using the Public Key already available inside the software's installation package.

2. The encrypted message will then be encoded in base64. The final payload will have the encoded message and the API key in it.
3. When the Server serves this request, it will check the API key and match it to a product. If it succeeds, it will then decode the message from base64 to its encrypted format.
4. Finally, the Server checks the matched product's Private Key and attempts to decrypt the message. If it succeeds, it will then extract the Hardware ID and Serial Key, which will serve to assign a registration in the Database, should preconditions be met.

The preconditions mentioned on step 4. are related to the device limit of each License Key. The Device limit is a parameter bound to Licenses that sets the limit of active registrations. If a client attempts to register a software with another device and the limit has already been reached, then the Server will reject this attempt with an error.

The steps listed previously can be seen in the image below (Fig.2), which transmits the same process through a sequence diagram:

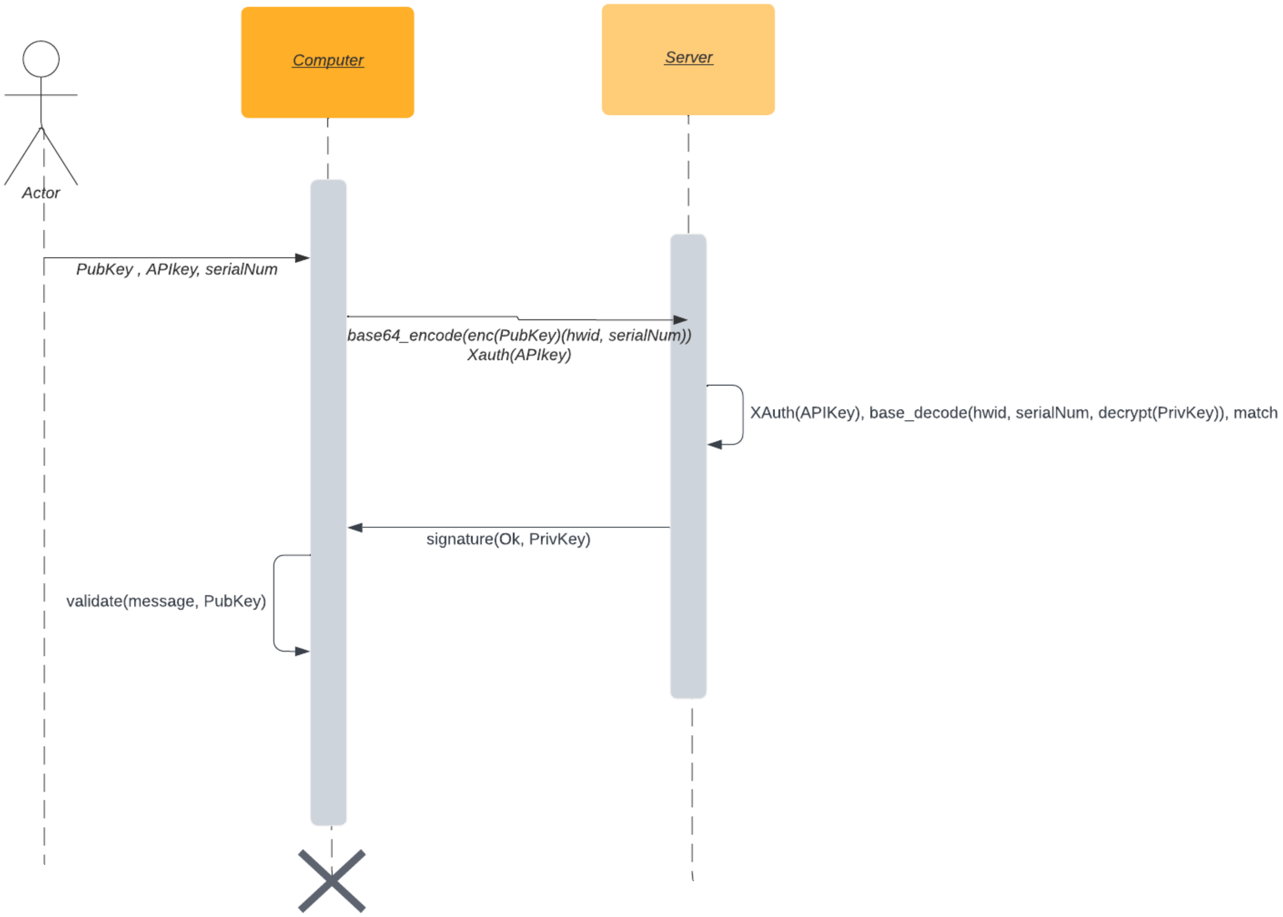
Computer
Server
Actor
PubKey , APIkey, serialNum
base64_encode(enc(PubKey)(hwid, serialNum))
Xauth(APIkey)
XAuth(APIKey), base_decode(hwid, serialNum, decrypt(PrivKey)), match
signature(Ok, PrivKey)
validate(message, PubKey)

Fig.2 - Validation Protocol (Sequence Diagram)

The Server always returns a code, along with an explanation. The codes are listed in the table below, along with their detailed description.

CODE	Description
SUCCESS	The validation request was successful and the device was registered.
OKAY	The current registration is still valid for the indicated device.
KEY_REVOKED	The key has been revoked and the software must succeed in validating.
KEY_EXPIRED	The key has expired and therefore the license is no longer valid.
KEY_DEVICES_FULL	The key can't support more devices. The validation fails.

On a side note, it is important to understand that much of the process is done through physical means which are beyond our Application's reach. Our application is not responsible for guaranteeing that the client's software will require a license validation. Handing out the Serial Key, API Key and Public Key isn't the Application's responsibility either.

The integrity and security relies on the goodwill of the client and other agents to not modify and crack the software handed to them. The Web Application merely ensures a stable and fluent communication process with the software, in order to certify, hand-out and validate Licenses. The software should also check frequently for the current state of the license as it is not our app's responsibility to notify each software of any changes.

3.4. Database

The Database used in our web application is the SQLite Database. Depending on the current state of the application, the database [file] is automatically generated if it does not exist. Doing so will clear all current data and restart the web application from fresh.

The modeling of the database was done using SQLAlchemy; a toolkit and object-relational mapper for Python. This simplified the workflow, and allowed us to treat each Table as an Object we could easily access to (or even construct).

Our database has five tables in it, each one described in the subsections below.

If you prefer a more visual representation of the database, the image in the next page (Fig.3) describes a class diagram that represents the tables and fields in our database along with their respective relationships.

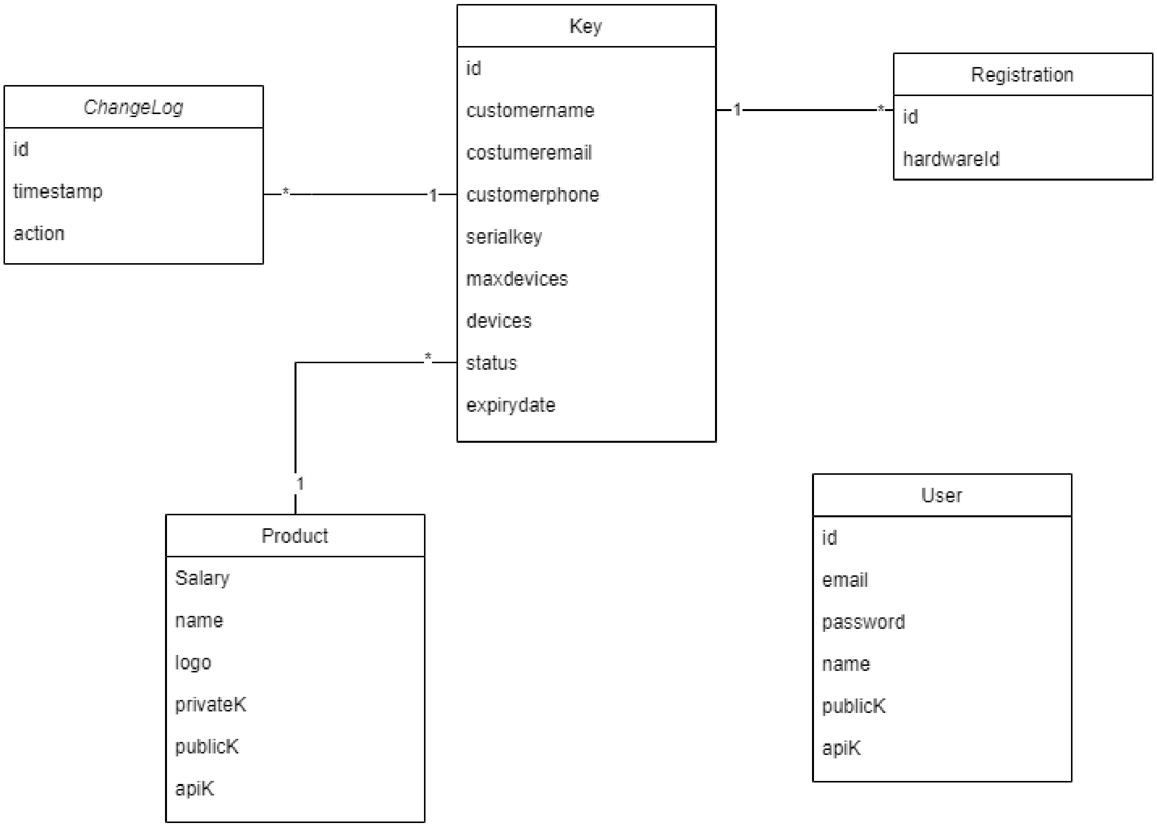

Fig.3 - UML Class Diagram of the Database

3.4.1 USER Table

This table is used as an extension of Flask-Login's required schema. The table stores the login data of each user in our application and is checked whenever any user attempts to login.

It is possible to have multiple users/administrators, by changing the environmental variables every time the server is started, but this is not something that is supposed to happen despite still being safe. Most of the time the USER table should have at most one tuple.

The table below illustrates the structure of our USER Table.

USER Table	Type	PK	UQ	AI
id	INT	X		X
email	TEXT		X	
password	TEXT			
username	TEXT		X	

3.4.2 PRODUCT Table

When a local administrator creates a Product, the server will automatically generate a public key, a private key and an API key. Each of these three fields are unique in order to ensure the inexistence of key collision cases.

Below, you can see the PRODUCT table used by our application. The field 'name' is a more user-friendly display of the Product, but the Server will always work with the id despite the former's UQ attribute. The optional 'logo' field stores an URL image that will be used to display on the CPanel front. The API Key is generated as a UUID object according to RFC 1422 and is limited to a combination of 2^{122} keys, which represents the maximum number of concurrent products in our application. Private Keys and Public Keys are much larger than the API keys and do not act as inhibitors.

Finally, the application does not allow the removal of existing Products. This was not necessary according to the specification, therefore tuples are bound to the table when created.

PRODUCT Table	Type	PK	UQ	AI
id	INT	X		X
name	TEXT		X	
logo	TEXT			
privateK	TEXT		X	
publicK	TEXT		X	
apiK	TEXT		X	

3.4.3 KEY Table

Whenever a License is created a Serial key is generated. Each key is made of 20 alphanumeric characters (26 letters and 10 digits), which leads to a maximum number of 36^{20} licenses. Bellow you can see the table used in our database model.

KEY Table	Type	PK	UQ	AI
id	INT	X		X
productid	TEXT	FK		
customername	TEXT			

customeremail	TEXT			
customerphone	TEXT			
serialkey	TEXT		X	
maxdevices	INT			
devices	INT			
status	INT			

The administrator is allowed to specify multiple data fields about the client along with an expiration field that defines the end of the license's uptime. The expiration date is not mandatory. If left empty, the Server will assume the administrator has created a lifetime License.

The 'maxdevices' field defines the limit of concurrent registrations to this key while the 'devices' field is used to track the number of registrations. This is a redundant implementation as the number of registrations can be checked by the REGISTRATIONS Table that is detailed next.

3.4.4 REGISTRATION Table

Every time a client successfully validates their own license with any given device, a record is created in this table. The 'keyid' field is a FOREIGN key that links the value to the KEY Table while the 'hardwareid' defines the id of the hardware used by the client.

The illustration of the table is handed next.

PRODUCT Table	Type	PK	UQ	AI
id	INT	X		X
keyid	INT	FK		
hardwareid	TEXT			

3.4.5 CHANGELOG Table

The Server keeps track of modifications on Keys in order to ensure safety and prevent misuse of the privileged commands by trusted members. Each time a modification is made, a record is added to the table below. The 'action' field reveals the context of the change and helps pinpointing what happened.

PRODUCT Table	Type	PK	UQ	AI
id	INT	X		X
keyid	INT	FK		
timestamp	INT			
action	TEXT			

The actions are part of a closed set and can be ‘Created’, ‘Revoked’, ‘Reactivated’, ‘Reset’, ‘Unlinked HardwareID’ and ‘Activated’.

3.5. Challenges

Throughout the development, the team struggled to conceptualize the Validation Protocol. Defining the tools we would use for asymmetric encryption was equally difficult as most failed to deliver an encryption method. Nonetheless, our meetings helped us move forward and overcome our challenges, and through a more deep research we arrived at a library capable of satisfying the encryption method that met our needs.

The design of the interface was not difficult but the team had to go through successive interactions to augment it with additional functionalities and user experience improvement.

The Database construction was also straightforward but learning SQAlchemy’s querying methods took some time. SQLAlchemy does not support a safe SQL Raw querying method, and instead relies on its object-relational features. Although it is possible to make raw SQL queries in SQAlchemy, it is still vulnerable to SQL Injection as it does not bind parameters.

4. Conclusion

We believe the development of this project was helpful for us and for any users who intend to make use of our application. We embraced this project as an essential tool for publishers and developers who wish to manage and contain their own licensing management.

In addition to its offers and utility, we enjoyed the experience and opportunity to put pressure on everything we learned. It supplemented our experience in Project Management but also helped us learn things that we didn’t know before.

Working with Flask, handling Cryptography and even documenting a functional RESTful API were useful opportunities to learn more and apply in other projects that the team may encounter in the future.

5. Contributions

Daniel Silva Gonçalves (80%)
Development of the Database schema. Development of the Front-End. Development of most of the backend along with the API of the Database. Writing of the Report.

Isla Patrícia Loureiro Cassamo (8%)
Attempted to implement Login Features. Creation of Dockerfile and creation of two technical images for this paper.

André Machado (12%)
Successful implementation of the Validation process. Failed to fulfill his assignments to issues.

6. References

[1] Vailshery, Lionel Sujay. “US Productivity software publishing revenue 2021.” *Statista*, https://www.statista.com/statistics/293408/revenue-of-operating-systems-and-productivity-software-publishing-in-the-us/
Accessed 8 June 2022.

[2] Panda Security, “What is Software Piracy?” *Panda Security*, 22 April 2019
https://www.pandasecurity.com/en/mediacenter/panda-security/software-piracy/
Accessed 8 June 2022.

[3] Flask, *Flask — Flask Documentation (2.1.x)*, 1 April 2010
https://flask.palletsprojects.com/en/2.1.x/.
Accessed 12 June 2022.

[4] Python 3.9.13, *Python 3.9.13 - Documentation*, 5 October 2020
https://docs.python.org/3.9/
Accessed 12 June 2022.

7. Resources

A - API Documentation

HTTP Method: **POST**
Path: /login
Parameters:
- usernameData
- passwordData

Description: The service route handles a login request from an unauthenticated user.

HTTP Method: **POST**
Path: /logout
Parameters: N/A
Description: Any request to this route will de-authenticate the authenticated user that made it.

HTTP Method: **GET**
Path: /
Parameters: N/A
Description: Displays the login page.

[Login Required]

HTTP Method: **GET**
Path: /cpanel
Parameters: N/A
Description: Displays the main tab of the CPanel administration.

[Login Required]

HTTP Method: **GET**
Path: /cpanel/product/id/<?productid>
Parameters:
- productid (?path)

Description: Displays a Product’s page with access to its licenses.

(continuation)

[Login Required]

HTTP Method: **GET**
Path: /cpanel/keydata/id/<?keyid>
Parameters:

- keyid (?path)

Description: Displays the page of a specified license (key).

[Login Required]

HTTP Method: **POST**
Path: /cpanel/product/create
Parameters:

- name
- imageURL

Description: Creates a product and stores it in the database.

[Login Required]

HTTP Method: **POST**
Path: /cpanel/product/id/<?productid>/createkey
Parameters:

- productid (?path)
- name
- email
- phoneNumber
- maxDevices

Description: Creates a key with the specified body parameters for the indicated product.

[Login Required]

HTTP Method: **POST**
Path: /cpanel/editkeys
Parameters:

- keyList
- action

Description: Modifies the state of a given key.

HTTP Method: **POST**
Path: /validate
Parameters:

- key
- value
- b64(HWID:SERIAL)

Description: Validates a license key into a product, based on the specified fields on the body.