**CAC Reader**

**Current platform support**

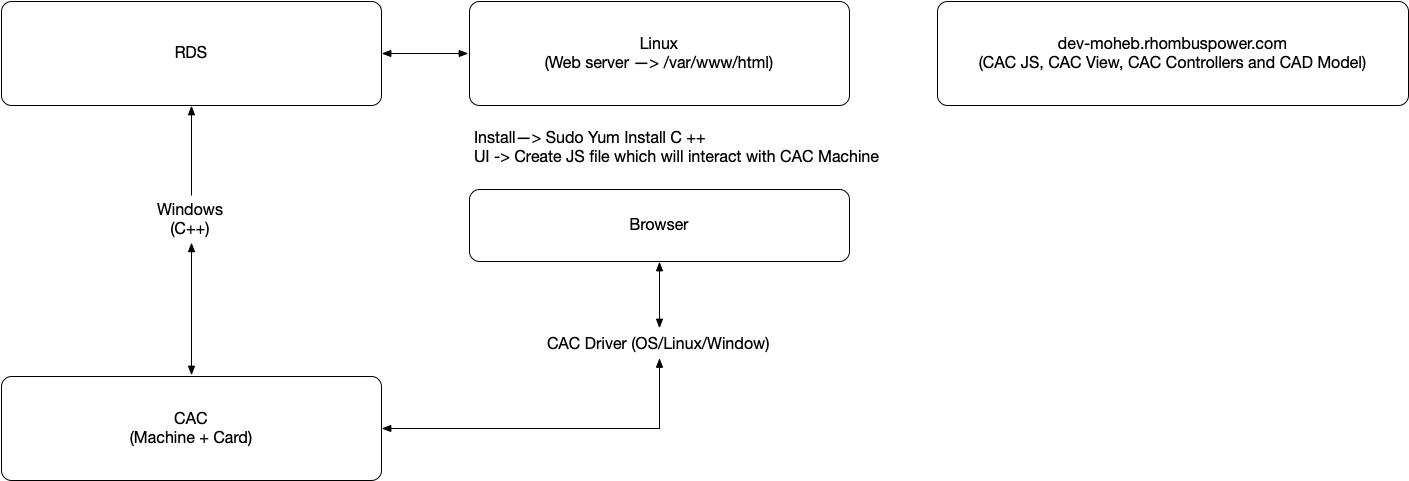
1. Windows
2. Linux

***Note:*** Support for Mac OS is work in progress

**Setup instructions**

1. Install the SCR3310 card reader driver for the supported platforms from the Identiv official website: <https://support.identiv.com/scr3310v2/>
2. Install OpenSC for the supported platforms from the OpenSC repository: <https://github.com/OpenSC/OpenSC/wiki> (Download section). For linux, use the linux distribution package manager to install opensc (i.e sudo apt-get opensc or sudo yum opensc).
3. Make sure the supported platform has the OpenSSL library installed, this condition should already be met by default for all common operating systems (Windows, Linux and Mac OS).
4. Check that the opensc tool is installed correctly and added to the PATH variables for windows. Alternatively, provide the application a full path to the opensc tool executable.
5. The executable, along with the dynamically loaded libraries, for the main program will be compiled in c++ and provided by the Rhombus power Kingston team to the card reader holder.

**Framework**



**Figure 1:** Basic framework for the CAC reader UI, RDS and C++ executable integration. By Dr. Sumit Kalia, May 29th, 2020.

**Option 1:**

1. Run the C++ executable on a supported machine with a physical USB port available for the card reader.
2. Connect the card reader to the machine, allow for the following basic operations to be performed on the card through the C++ executable:
   1. Register a card
   2. Remove, block, timeout or blacklist a card
   3. Identify a card holder
   4. Track the current status of the card (In use/ Not in use)
3. The card serial id should be encrypted, perform a handshake with both the card reader and the browser with the card signature, not its serial id, to avoid admitting spoofed serial ids into the system.
4. All the card’s relevant information, credentials are stored in the Rhombus power designated database.
5. Use WebAuthn protocol, similar to the yubikey, to send the card credentials through the browser.

*Pros*:

1. Does not need user input for identification (since the browser would speak directly, via WebAuthn protocol, so the card identity holder is known). Thus, it’s faster to implement.
2. May use encryption algorithms already implemented by WebAuth.

*Cons:*

1. Has incompatibility issues with some browsers. [[1]](#footnote-1)[[2]](#footnote-2)[[3]](#footnote-3)
2. Has security concerns, needs careful research for the encryption algorithms that will be used.[[4]](#footnote-4)

**Option 2:**

1. Follow the steps 1-4 in option 1.
2. On the browser side, the login UI would be listening through a web socket to the card reader, confirm the card credentials with the database on card insertion.

**Note:** Another user input field needs to be in-place to confirm the card holder with the database, otherwise the identity of the card holder is ambiguous to the database.

1. The database would manage the user session by a token for the inserted card.

*Pros:*

1. Flexibility with the encryption algorithm implemented.
2. User input allows for another layer of security (Similar to 2FA)

*Cons:*

1. Everything needs to be implemented from scratch, including the encryption method as opposed to using WebAuthn

**Work done so far**

C++ executable, database connection and interaction with the card.

**Work in progress**

The login UI and integration with the database.

1. <https://caniuse.com/#search=webauthn> [↑](#footnote-ref-1)
2. <https://bugs.chromium.org/p/chromium/issues/detail?id=997538> [↑](#footnote-ref-2)
3. <https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/qCJhuuZH5p0> [↑](#footnote-ref-3)
4. <https://paragonie.com/blog/2018/08/security-concerns-surrounding-webauthn-don-t-implement-ecdaa-yet> [↑](#footnote-ref-4)