Public Images as Secure Group Messages (PISGM)

By Nathan Percival, Conor Finegan, and Charles Cook
Led (by initiative) by Charles Cook

https://github.com/cSquaerd/pisgm
Description revised May 8th, 2020

1 Abstract

PISGM is a client-server software tool that affords the creation of a PKI so that users can send messages to other groups of users securely. The security is two-fold: One layer of encryption, via AES keys delivered under RSA, and one layer of obfuscation, by way of encoding the AES ciphertext as pixel values in grayscale images. These grayscale images, contained in standard PNG files, can be posted losslessly to any public and potentially unsecured web location, and will give no indication as to being generated by PISGM, at least explicitly. As long as connections can be established by clients with a central image key management server, indefinite amounts of messages of arbitrary sizes can be delivered as PISGM images, ensuring confidential and directed communication.

2 Project Review

Our project, coded entirely in Python, with some Apache Webserver for the key server, made extensive use of the PyCryptoDome standard module, which had many cryptographic primitives built into it. As for the security building blocks we used, there was 2048-bit RSA PKC, 256-bit AES encryption under EAX Mode, the SHA256 hash for signature verification via RSA, and a random bytes generator function *get_random_bytes*, that maps to *os.urandom* in Python, which itself, on Linux, maps to the *getrandom* system call, which is fit for cryptographic use.

Our current version of PISGM can make use of a key server located at *keys.pisgm.net:5610* to succesfully create new image keys, use said keys to create images holding AES ciphertext, and authorize re-transmission of the image keys to users who want to decipher an image and are allowed to do so per image. We did not worry about creating a registration form of some kind during development, though this would be one of the first things we would work on in continued development. Additionally, we have heard word that our framework would be ideal for use with stenography, as all the cryptographic data is handled as raw *bytes* objects in Python, and the Imaging tools we used from PIL are rather intuitive.

Finally, as for the division of labor, Mr. Cook wrote all of the encryption objects and Imaging wrapper functions to do the backend work, Mr. Finegan wrote a CLI tool as a front-end for users, and Mr. Percival wrote the server-side CGI scripts, and he configured the Apache Webserver.

Also, for sample code, and example output, see the testing/results folder for several Interactive Python Notebooks (.ipynb) that contain the demonstrations shown off during our presentation. They can be viewed easily on the Github page as well.