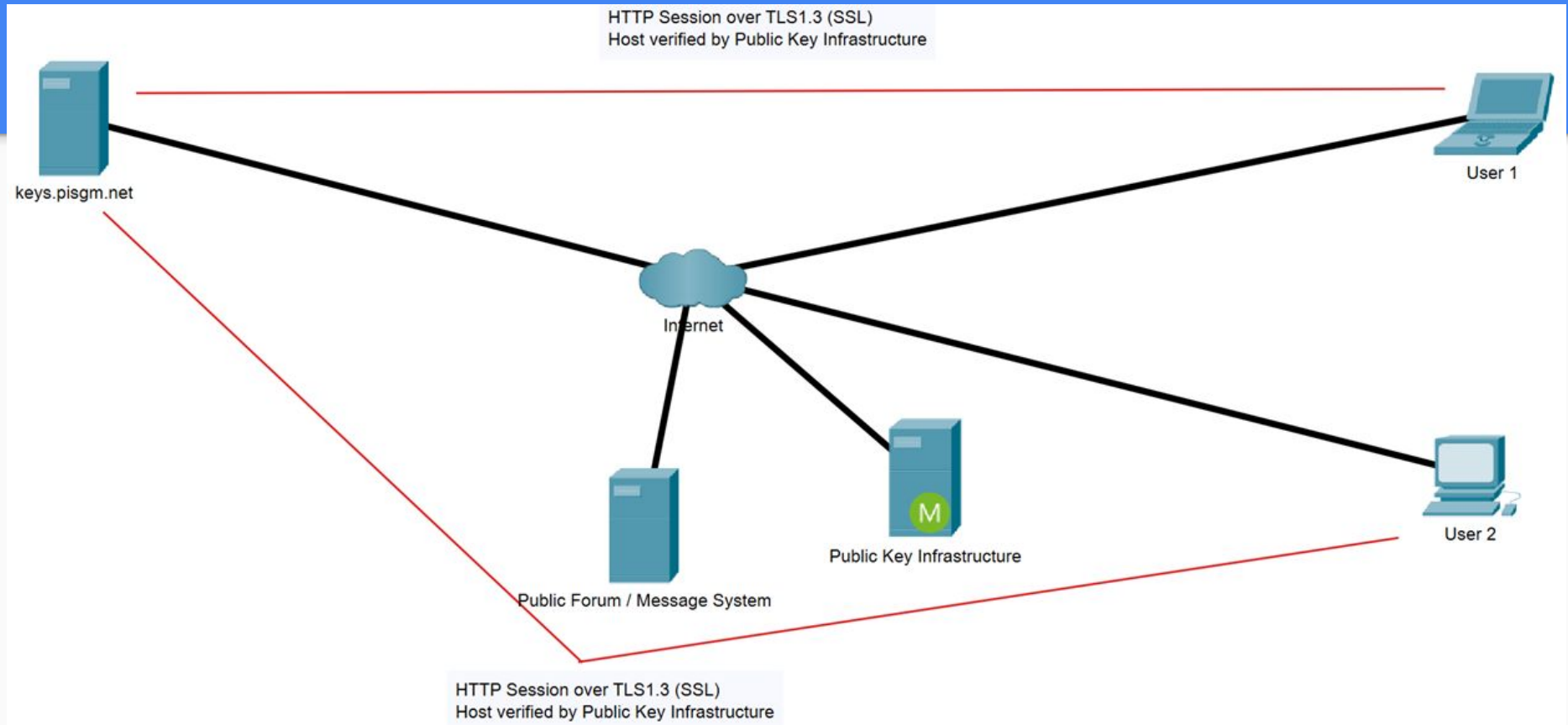
# Public Images as Secure Group Messages (PISGM)

By Nathan Percival, Conor Finegan, and Charles Cook

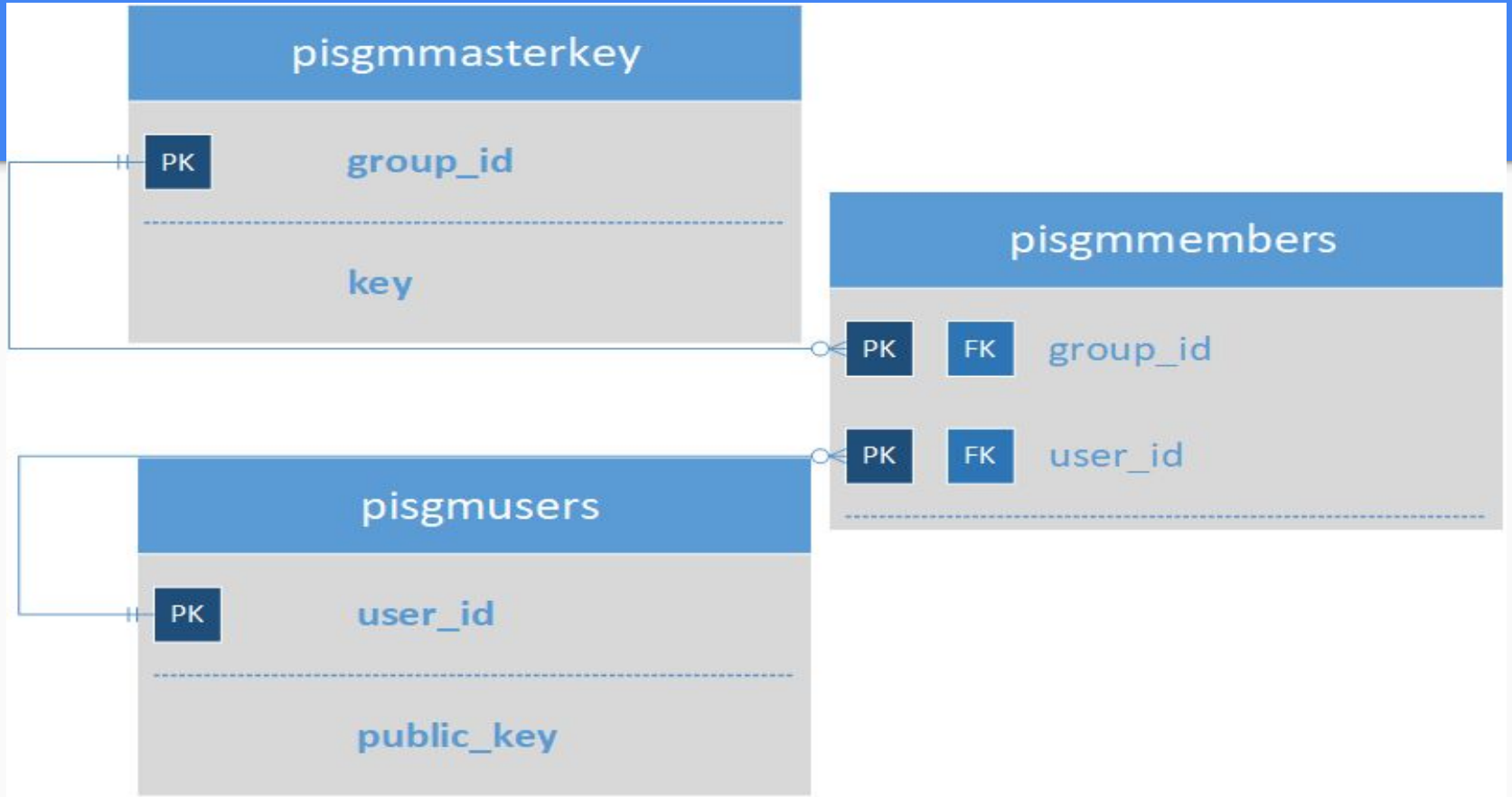
# PISGM Motivation

- There is a desire to pass secret messages using insecure systems such as Reddit, 4chan, Twitter, Facebook,  or other online systems.
- These systems allow message to be shared easily but there is no system provided to allow direct, secure communication to a group via these public forums.  Most of these systems do allow the posting of images.
- A group of people wishing to communicate securely and want to be able to directly read the message without the need to have access to a shared key
    - The key to decrypt any message should not be usable to decrypt any other message
    - The key will only be available to members of the group
    - Communications of the key will be secure between the server and client

# System Architecture



# Backend Database



# Server Setup

- Purchased pisgm.net domain to allow the use of full public key infrastructure.
- Used existing system and tunneled port 5610 through a NAT system to server on system.
- Setup and registered server public key using Let’s Encrypt - A nonprofit Certificate Authority providing TLS certificates.
- Configured an Apache Webserver on port 5610 using HTTP over TLS1.3 (HTTPS) protocols
- Configured server to execute Python code using the Common Gateway Interface capabilities of Apache Webserver for server-side code execution.

# Server Functions

- Users are registered with the server including their Public Key
- Groups are created by the system administrator
- Users are associated with a group by the system administrator
- User submit request for an image key including
  - User ID, Group ID, Poster ID, Timestamp and Nonce
  - Include RSA signature of these items
- Server returns an RSA encrypted message containing an AES-256 key
  - AES-256 key is unique to the Group ID, Poster ID, Timestamp and Nonce.
  - Based on a base group AES-256 key (only on server)

# Libraries & Resources

- **PIL** (Python Image Library; image encoding/decoding)
- **Crypto** (PyCryptoDome; pre-made implementations of RSA, AES, & SHA)
- **time** (accessing UNIX timestamps)
- **cgi** (Common Gateway Interface for server-side processing)
- **sqlite3** (light-weight database for backend on server)
- **random** (generation of random IDs)
- **requests** (transmission of HTTP messages)
- **base64** (type conversions for HTTP messages)

# Encryption Objects: RSA (pisgmRSA.py)

**keyObj**(keySize : *int*, keyImport : *NoneType* or *dict*)

keyU : Crypto.PublicKey.RSA.RSAKey
keyR : Crypto.PublicKey.RSA.RSAKey
publicCrypt : Crypto.Cipher.PKCS1_OAEP.PKCS1OAEP_Cipher
privateCrypt : Crypto.Cipher.PKCS1_OAEP.PKCS1OAEP_Cipher
publicSign : Crypto.Signature.pkcs1_15.PKCS115_SigScheme
privateSign : Crypto.Signature.pkcs1_15.PKCS115_SigScheme

encryptU(message : *str* or *bytes*) -> *bytes*
decryptR(ciphertext : *str* or *bytes*) -> *bytes*
signR(message : *str* or *bytes*) -> *bytes*
verifyU(message : *str* or *bytes*, sig : *bytes*) -> *bytes*
exportU() -> *bytes*
exportR() -> *bytes*

**publicKey**(rawKeyOrFilename : *bytes* or *str*)

key : keyObj

encrypt(message : *str* or *bytes*) -> *bytes*
verify(message : *str* or *bytes*, sig : *bytes*) -> *bool*
export() -> *bytes*

**privateKey**(rawKeyOrFilename : *bytes* or *str*)

key : keyObj

decrypt(ciphertext : *str* or *bytes*) -> *bytes*
sign(message : *str* or *bytes*) -> *bytes*
export() -> *bytes*

# Encryption Objects: AES (pisgmAES.py)

| **aesEncrypter**(key : *bytes*, nonce : *bytes*, message : *str* or *bytes*) |
| --- |
| cipher : Crypto.Cipher._mode_eax.EaxMode<br>output : bytes |
| |

| **aesDecrypter**(key : *bytes*, nonce : *bytes*, ciphertext : *str* or *bytes*) |
| --- |
| cipher : Crypto.Cipher._mode_eax.EaxMode<br>output : bytes |
| |

| **aesRadio**(key : *bytes*, nonce : *NoneType* or *bytes*) |
| --- |
| key : *bytes*<br>nonce : *bytes* |
| encrypt(message : *str* or *bytes*) -> *aesEncrypter.output*<br>decrypt(ciphertext : *str* or *bytes*) -> *aesDecrypter.output* |

# Imaging Functions (pisgmImaging.py)

- **bestBox**(n : *int*) -> *list*
  - Finds the two closest-together factors of n (for the most square dimensions given area n)
- **bytesToGrayImage**(b : *bytes*) -> *PIL.Image.Image*
  - Wrapper for PIL.Image.frombytes
- **grayImageToBytes**(i : *PIL.Image.Image*) -> *bytes*
  - Wrapper for PIL.Image.Image.getdata
- **saveImage**(i : *PIL.Image.Image*, filename : *str*)
  - Wrapper for PIL.Image.Image.save

# Main Functions (pisgmMain.py)

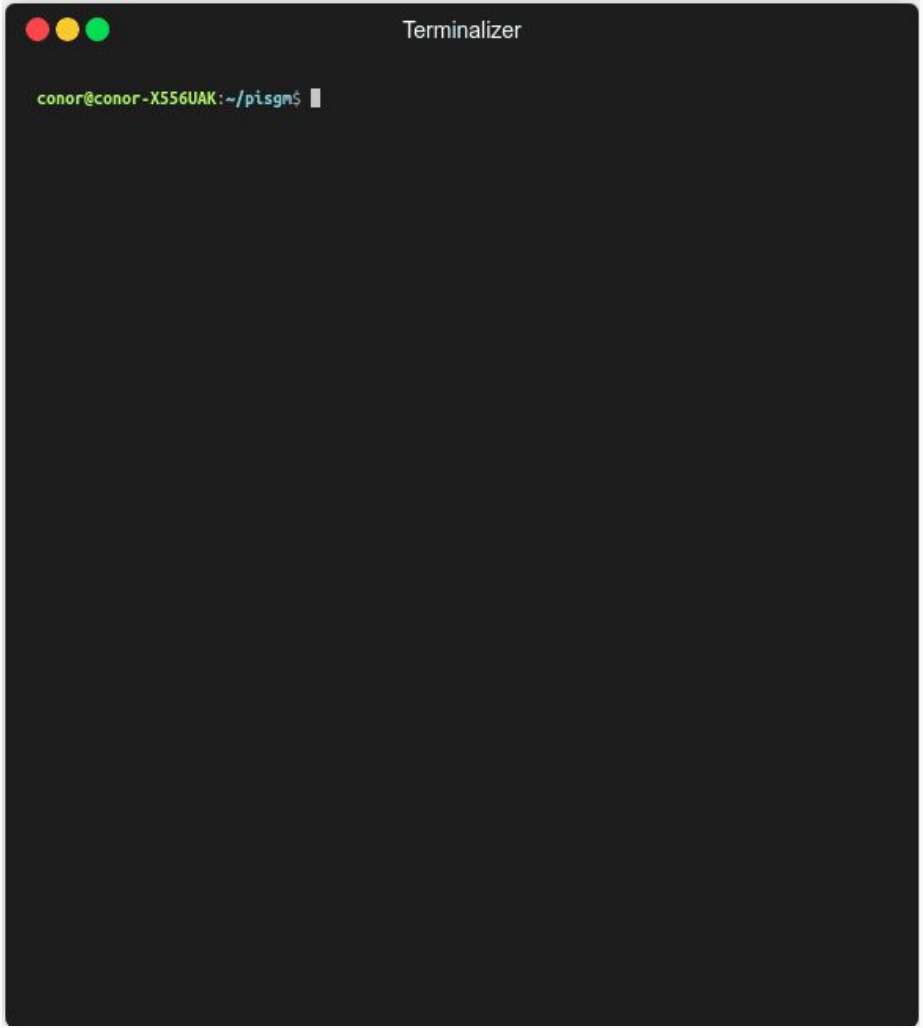
- **makeRequest**(keyR : *privateKey*, uid : *int*, gid : *int*) -> *Reply*
  - Request a new image-key (AES) for some message (not needed at this stage)
- **makeImage**(message : *str*, uid : *int*, reply : *Reply*) -> *PIL.Image.Image*
  - Use the image-key in the Reply object to encrypt the message and encode it into an image
- **decodeImage**(image : *PIL.Image.Image*,
  keyR : *privateKey*, uid : *int*, gid : int) -> *str*
  - Extract the header data from the image (timestamp, nonce, author's UID), request the image-key, and (potentially) decrypt the ciphertext and return the original message

# Interface

- Simple command-line interface for encrypting and decrypting
  - pisgm -e <text> <file> to encrypt
  - pisgm -d <file> to decrypt
- User settings (keys, user id, group id) are stored in local file
  - Right now this assumes that the user's machine is secure
  - In the future we should have some way to recover lost accounts
- Text can be provided via CLI argument or standard input
  - Helps make pisgm cooperate with other tools

# Demonstration

# Demonstration


Terminalizer
conor@conor-X556UAK:~/pisgm$