ChaChat

Crypto by Django

Cryptography

Yang, Heetak (Leslie)

Introduction

Previously, while looking at WireGuard, the encryption algorithm used in it was ChaCha20 and the hash algorithm Poly1305. ChaCha20 and Poly1305 are used a lot in today's programs, and it was very interesting, but unfortunately, it was very difficult to put an algorithm into and apply it.

In this case study, I am implementing a project to communicate by encrypting a password and encrypting a message through an encryption algorithm and hash that are mainly used on the web while implementing a chat application.

Background

In order to study and test the cryptographic algorithm, I thought about what to do as a project. I thought it was the chat application that I thought was best. There are five important points in information security. Confidentiality, Integrity, Availability, Authenticity and Non-Repudiation (Accountability) must all work to ensure proper security. I think the chat application I envisioned satisfies five things. The following is a description of each part, and is the same as the order of application operation.

1. Availability: The application is web-based, allowing access to any device. It prevents connection blockage through backup to prevent DDoS attacks, etc. and synchronization with other servers.

2. Authenticity: Prevent unauthorized people by creating logins. The security is further enhanced through authentication through OTP, and the record is recorded in the logged-in ID when chatting.

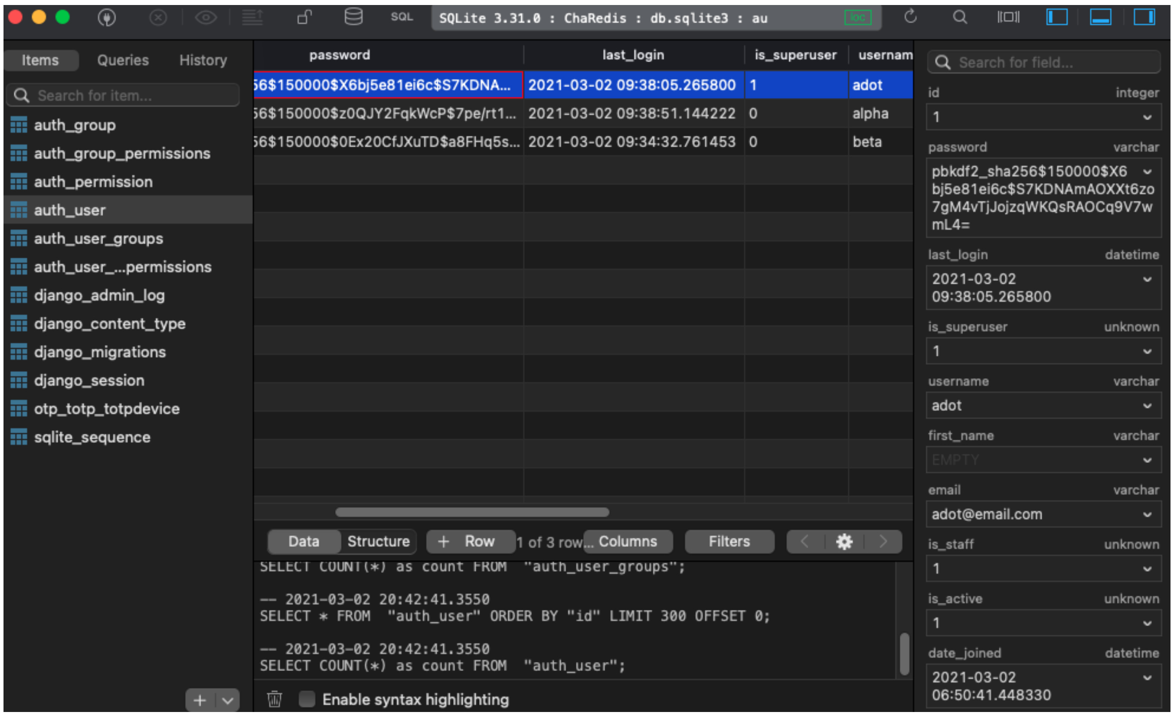
3. Confidentiality: Encrypted with an encryption algorithm. Passwords must be encrypted so that even administrators cannot see them, and even if the password is leaked due to cracking of the server, it is encrypted so that the real password is not known. The message is the same reason as the password, and it is encrypted with PBKDF2.

4. If you encrypt only with an encryption algorithm, decryption may be possible relatively easily. So, add salt value and mix. It is impossible to decrypt the encryption algorithm without knowing the hash value. It uses the representative SHA256 algorithm.

5. Non-Repudiation: Also known as tracking of accountability, cryptographic hash functions through digital signatures are widely used to verify the authenticity of digital data. Through this, it checks whether there has been any transformation when logging in or sending and receiving messages, and encryption is also performed during communication so that information can be transmitted and received safely. It is encrypted through OpenSSL, and secure version of WSS is used for HTTPS communication and websocket communication.

Cryptography process used

First, I created a login page using auth, a feature installed by default using Django. Basically, there is an admin function in Django, and based on this, the function of auth is provided. The login and register pages were created with the function of Django's auth, and the password uses SHA256 for PBKDF2 encryption and hash, and can be checked by accessing the admin page. After encryption, some of them are hidden and cannot be checked, but the encrypted information can be checked in detail in the database. SECRET_KEY is automatically created in config/setting.py.



auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_...permissions
django_admin_log
django_content_type
django_migrations
django_session
otp_totp_totpdevice
sqlite_sequence

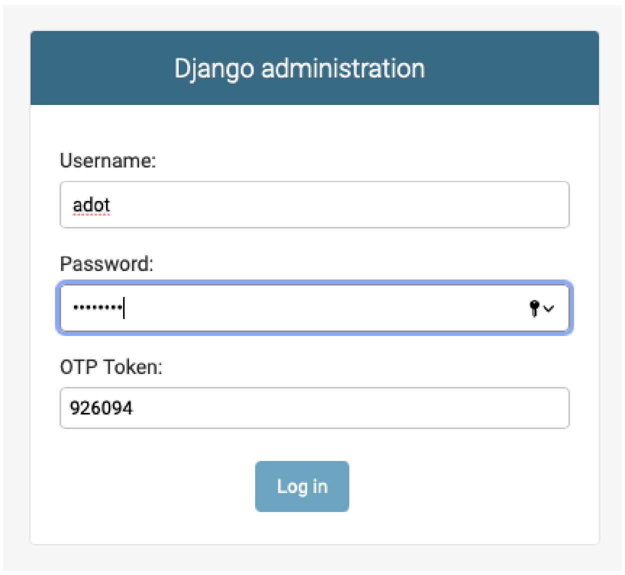
password	last_login	is_superuser	usernam
56$150000$X6bj5e81ei6c$S7KDNA...	2021-03-02 09:38:05.265800	1	adot
56$150000$z0QJY2FqkWcP$7pe/rt1...	2021-03-02 09:38:51.144222	0	alpha
56$150000$0Ex20CfJXuTD$a8FHq5s...	2021-03-02 09:34:32.761453	0	beta

```
SELECT COUNT(*) as count FROM  "auth_user_groups";

-- 2021-03-02 20:42:41.3550
SELECT * FROM  "auth_user" ORDER BY "id" LIMIT 300 OFFSET 0;

-- 2021-03-02 20:42:41.3550
SELECT COUNT(*) as count FROM  "auth_user";
```

id	integer	1
password	varchar	pbkdf2_sha256$150000$X6bj5e81ei6c$S7KDNAmAOXXt6zo7gM4vTjJojzqWKQsRAOCq9V7wmL4=
last_login	datetime	2021-03-02 09:38:05.265800
is_superuser	unknown	1
username	varchar	adot
first_name	varchar	EMPTY
email	varchar	adot@email.com
is_staff	unknown	1
is_active	unknown	1
date_joined	datetime	2021-03-02 06:50:41.448330

[Picture1.]

In addition, security was improved through OTP when logging in. However, this is set only on the Admin page, and it has not been added to the login page yet.



Django administration

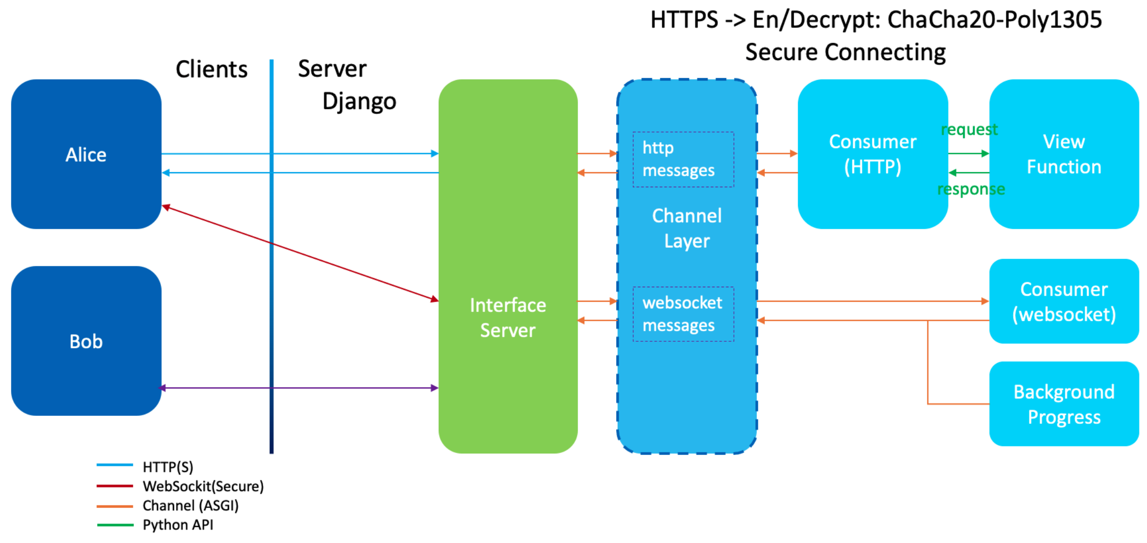
Username: adot

Password: ••••••••

OTP Token: 926094

[Picture2.]

Use the new module when chatting. The channel is changed so that django can write code asynchronously, and through the synchronous core of django, the project can handle not only HTTP but also protocols that require long-term connection Websockets, MQTT, chatbots, amateur radio, etc. Do it. It is useful because it allows you to choose a style of code while preserving django's synchronous and easy-to-use properties. The synchronous feature of the django view style, completely asynchronous, or a mix of both. Most of all, it is easy to extend the HTTP-only project to other protocols by integrating with django's auth and session system. The following is a description of the channel layer used in Django.

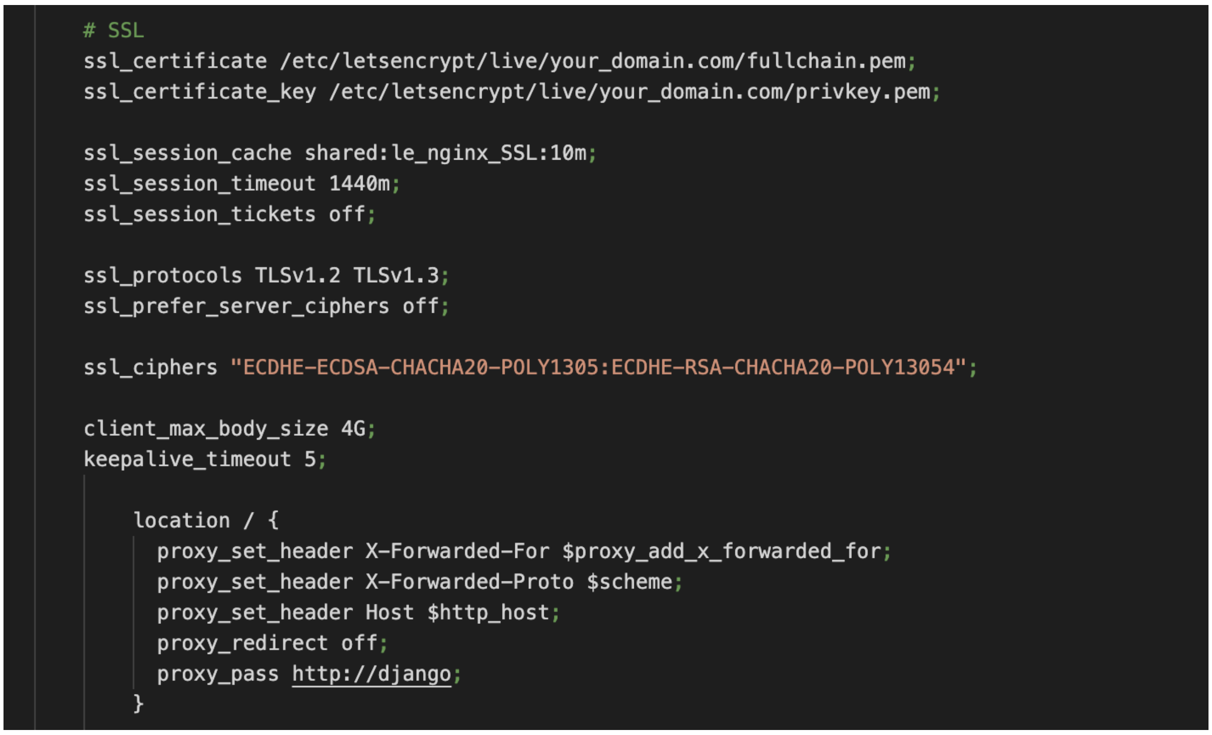


[Picture3.]

This builds a chat app, which is stored in the channel's Redis database. When stored in Redis, it is encrypted through RSA256 with an entropy of 32 bytes. Basically, it is encrypted using SECRET_KEY in Django setting.py as it is.

What I couldn't implement

It wasn't able to pull the login session all the way to the chat, which is a huge flaw in leaving exactly who sent what messages. In the case of HTTPS, Nginx basically redirected and used HTTPS, and SSL was received and applied from Let's Encrpyt. In the case of WSS, Daphne was running as a daemon to receive and run the proxy of Nginx. It was being tested and to some extent, but it wasn't complete either. When HTTPS communication is performed, SSL part is added in config/nginx.conf, and encryption is set in a communication method using ChaCha20-Poly1305.



```
# SSL
ssl_certificate /etc/letsencrypt/live/your_domain.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/your_domain.com/privkey.pem;

ssl_session_cache shared:le_nginx_SSL:10m;
ssl_session_timeout 1440m;
ssl_session_tickets off;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off;

ssl_ciphers "ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY13054";

client_max_body_size 4G;
keepalive_timeout 5;

    location / {
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header Host $http_host;
      proxy_redirect off;
      proxy_pass http://django;
    }
```

[Picture4.]

Conclusion

There were various difficulties in development. To be honest, it was very confusing, and the difficulty level was very high for me. In addition to dystonia and migraine, as well as gastroenteritis, my health was very poor. But this was my challenge. I wanted to try it with ChaCha20 somehow, but it was close to impossible. So, I developed with the power of the Django framework to develop it as quickly as possible. Although it wasn't all complete, making it while searching for materials made me enjoyable. If the body was okay, it would have been completed even if the quality was not high, but this is a regretful part.

Reference

"https://docs.djangoproject.com/en/3.1/" - Django -

"https://channels.readthedocs.io/en/latest/tutorial/part_1.html" - Django: Channels -