Cryptography Whitepaper

PACE

December 2022

1 Introduction

PACE is a fitness tracker that respects your privacy. It is end-to-end encrypted and is designed to be secure.

Fitness and geolocation data are among the most sensitive and private data about a person. Anyone having access to these can derive a lot of information such as the health of an individu, the habits, the living location, etc. Articles are frequently published trying to raise awareness about the privacy issues of existing fitness apps and explain how to mitigate the privacy issues by tweaking the settings of the trackers. However, the security issues remain as the data are unencrypted therefore accessible by the apps providers or subject to leak in case of a security breach.

Fitness trackers are more popular than ever. They provide a solution to keep track of training over time and help athletes and sport enthusiasts to improve their physical preparation before a competition. With PACE, we aim at offering a secure, end-to-end encrypted platform to record, analyse and share outdoor and fitness activities without compromising the privacy. This document describes the cryptography and security protocols implemented by PACE.

2 End-to-end Encryption

When designing the security architecture of PACE, we consider the public clouds to be insecure and unable to store personal unencrypted data. While we follow the best practices to secure our infrastructure, we also assume it to be vulnerable to an malicious intrusion.

The security model of PACE relies on Elliptic Curve cryptography. All the data are encrypted on the users' devices. To achieve this, each user has a unique key pair for encryption and a unique key pair for signing the data. These key pairs are generated during the creation of an account and the private keys are never known by anyone else than their owner. It ensures that nobody, not even PACE, can read the users' data except the eventual other users they may decide to share with. It also protects again compromising and leaking some data in case of an intruder gains a malicious access to the infrastructure.

The encryption is implemented with NaCL using the following libraries:

- Libsodium[1] for Android and iOS. PACE has open sourced its cryptography library for Android and iOS based on Libsodium and is available on GitHub[2].
- tweetnacl.js[3] for the web.

3 Account Creation

During the account creation, a user profile is created and is composed of the encryption and signing key pairs and a profile key encryption. All the procedure is executed on the user's device that can be their mobile phone or their web browser. The first step consists in generating the profile data with the key pairs and different salts used for the profile encryption and during authentication:

1. User chooses a username and enters a password.
2. An encryption key pair is generated. The private key is randomly generated. The corre-

[1] `https://libsodium.org`
[2] `https://github.com/withpaceio/react-native-nacl-jsi`
[3] `https://tweetnacl.cr.yp.to`

sponding public key is computed over the Elliptic Curve Curve25519.

3. A signing key pair is generated. The private key is randomly generated. The corresponding public key is calculated using the Ed25519 algorithm.

4. A random 128 bits salt $salt_{password}$ is generated. It is used to derive a symmetric key from the password for the authentication and for the profile encryption.

5. A random 128 bits salt $salt_{encryption}$ is generated. It will be used to generate the profile encryption key.

6. A random 128 bits salt $salt_{token}$ is generated. It will be used for the authentication.

In the next sections, we call $profile_{data}$ the set composed of the signing and encryption key pairs and the salts $salt_{password}$, $salt_{encryption}$ and $salt_{token}$.

Once the profile has been generated, the private keys are stored on the secure storage of the device if it is executed on the mobile phone of the user. When running in browser, the private keys are never stored. In both cases, the private keys never leave the user's device unencrypted.

The second step consists in encrypting and authenticating the profile data using the *secretbox* model of NaCL:

1. A symmetric key $key_{password}$ is derived from the password using the Argon2id algorithm with $salt_{password}$.

2. A symmetric key $key_{encryption}$ is derived from $key_{password}$ using HKDF with the salt $salt_{encryption}$. The profile encryption key $key_{encryption}$ never leaves the mobile or the browser of the user.

3. The profile data are encrypted using the $xsalsa20-poly1305$ authenticated cipher with the symmetric encryption key $key_{encryption}$.

As the third and final step, the encrypted profile data, the salts $salt_{password}$, $salt_{encryption}$ and $salt_{token}$ and the encryption and signing public keys are sent to the backend. The private keys are end-to-end encrypted and the public keys remain publicly visible. The password and the profile encryption key $key_{encryption}$ are never sent over, not even in a encrypted form.

4 Authentication and Password Change

4.1 Authentication

The authentication is implemented using the Secure Remote Password protocol (SRP), version 6a. Using this protocol, the users' password are never sent over the network. This is critical because the key pairs are computed based on the password. Using SRP ensures that the password, or any derived data from it, are never sent to the servers.

Once the authentication is successful, the server sends a JWT access token, a JWT refresh token, the encrypted profile data and the salts $salt_{password}$ and $salt_{encryption}$. With these data, the profile encryption key $key_{encryption}$ can be derived from the password by following the same steps than during the account creation. Finally, the profile data can be decrypted.

4.2 Password Change

Because the profile is encrypted using a symmetric key derived from the password, changing it requires to re-encrypt the profile data using the key derived from the new password.

Changing the password can be done from the mobile application or the account settings web page. In both cases, the user must be authenticated. Once the user has chosen a new secure password, the second step from the account creation is executed resulting in a new encrypted profile and salts. These are sent to the server as the final step of changing a password.

5 Account Recovery

The users can choose to configure a recovery email address and to generate a recovery passphrase for them to store in a secure place. They will be used to recover their data in case of a password loss. Their data being end-to-end encrypted, it is not possible for PACE to decrypt and recover their activities. The account recovery requires an email address so it is up to the user to choose whether to configure it or not.

5.1 Account Recovery Configuration

At first, the user needs to configure an email address that will be used to recover their account. After entering their email address, it is sent to the server and stored after being encrypted using the algorithm $AES-GCM-256$ to prevent a leak of the email addresses in case of a malicious access to the database. A link with an expiring verification token is sent to the email address. The passphrase configuration starts from here and once the user is authenticated.

The recovery is based on the *secretbox* model of NaCL and is configured following:

1. A BIP39 mnemonic is generated.
2. A random 128 bits salt $salt_{recovery}$ is generated.
3. A symmetric key $key_{recovery}$ is derived from the mnemonic using HKDF with the salt $salt_{recovery}$.
4. The profile data are encrypted using the $xsalsa-poly1305$ authenticated cipher with the symmetric encryption key $key_{recovery}$.
5. A random 128 bits salt $salt_{key_recovery}$ is generated.
6. The key $key_{recovery}$ is hashed on the client side using the Argon2id algorithm with the salt $salt_{key_recovery}$.
7. The encrypted profile data, the salts $salt_{recovery}$ and $salt_{key_recovery}$ and the hash of the recovery key are sent to the server. The mnemonic and the key $key_{recovery}$ never leave the mobile or the browser of the user.

The user is in charge of storing their mnemonic in a safe place of their choice. They may decide to change their recovery email address. In such case, the whole procedure has to be reexecuted and a new mnemonic will be generated. The previously existing one will not be useful any more and can be deleted by the user. At any point, there is at most one valid recovery passphrase per user.

5.2 Recovering an account

When a user wants to recover their account, the following steps are executed:

1. The user enters their username.
2. An email with a link containing an expiring one-time recovery token is sent to the corresponding recovery email address.
3. The user clicks on the link to prove their identity and the recovery token is verified by the server.
4. The server sends the salts $salt_{recovery}$ and $salt_{key_recovery}$.
5. The user enters their BIP39 mnemonic.
6. The recovery key $key_{recovery}$ is derived from the mnemonic using HKDF with the salt $salt_{recovery}$.
7. The key $key_{recovery}$ is hashed on the client side using the Argon2id algorithm with the salt $salt_{key_recovery}$.
8. The hash of the key is sent to the server.
9. The server verifies that the hash of the key is identical than the one stored. If successful, the server sends the encrypted profile data.
10. The profile is decrypted using $key_{recovery}$. From this point, the user has gain access to their secret signing and encryption keys, therefore are able to decrypt their data.

The final step of the account recovery is changing the password. Once the user has chosen a new secure password, the random 128 bits salts $salt_{password}$, $salt_{encryption}$ and $salt_{token}$ are regenerated the second and third steps of the account creation are repeated.

6 Activity Encryption

PACE uses the *box* model of NaCL to encrypt and authenticate the activities and the *secretbox* model of NaCL to encrypt and authenticate the encryption keys. The activity encryption works as follows:

1. A symmetric encryption key $key_{activity}$ for the activity is generated.

2. The activity data are encrypted using the $xsalsa - poly1305$ authenticated cipher with the encryption key $key_{activity}$.

3. After using Elliptic Curve Diffie-Hellman (ECDH) over the curve Curve25519, the result is hashed using the algorithm HSalsa20 to obtain a shared secret between the public and the private key of the user $secret_{activity}$.

   At the moment of writing, PACE does not support activity sharing between several users. In the future, this step will be repeated for each public key of the users to share the activity with.

4. The activity encryption key $key_{activity}$ is encrypted using $xsalsa - poly1305$ authenticated cipher with the shared secret $secret_{activity}$.

5. The encrypted $key_{activity}$ and the encrypted activity data are sent to the server.

The $key_{activity}$ is never sent to the server and when running in the browser, the encryption keys are never stored.

On the mobile device of the users, the encrypted activity data are stored on the local file system. With the signing and encryption secret keys store in the secure storage of the devices, it enables the support for offline mode without compromising the data security. In case an activity is saved on the device without connection, the encrypted data will be automatically uploaded when the device will be back online.

6.1 Activity Sharing

We briefly described how sharing an activity will work in a future version of PACE. Sharing an activity is done by sharing its symmetric encryption key. For this, the owner needs to encrypt the activity symmetric encryption key $key_{activity}$ with the public key of the user to share with and to upload it to the servers. By doing so, the storage footprint is reduced: only the activity encryption key is encrypted separately for each user rather than all the activity data.

6.2 Activity Decryption

Decrypting an activity is performed by following the same steps than during the encryption in reverse order. Using their encryption private key, the activity symmetric key is decrypted and result in $key_{activity}$ in clear. The activity data can now be decrypted.

6.3 Activity Metadata Encryption

Metadata can reveal meaningful information without access to the data. As an example, with the date of an activity, it would be easy to see if the user practices sport on a regular basis and at which frequency without having access to the activities themselves.

PACE strives to have the minimal amount of metadata stored in database. To continue with our example, the date of the activities are encrypted using the activity encryption key $key_{activity}$. It ensures that this critical piece of information remains known by the user only and behavioral patterns cannot be deduced from the metadata.

When fetching the list of activities, the server returns them by order of insertion in the database. This has the downside of having potential reordering happening on the client side once the activities are decrypted. However, we consider this tread off being a much better solution than having the activities dates stored unecrypted, resulting in a much lower respect of the privacy.

7 Conclusion

We described the cryptography and security protocols implemented in PACE to protect the privacy of the users while using a fitness tracker. We presented how the activity data are encrypted and stored on the mobile devices.

Geolocation, fitness and health data are all collected by fitness tracker and used to provide very useful insights for training. Yet, these are very sensitive and can reveal critical information about the users. With PACE, our goal is to provide a fitness tracker that protects the privacy.