Virgil Security's PHE Service Technical Paper

What is PHE

Password-Hardened Encryption (“PHE”) is a two-party protocol that brings password-based security to a new level in three ways:

1. Replaces password hashing in a way making it impossible to run offline and online attacks.
2. Adds unique data encryption key to every user record. This key can be revealed only after providing a correct password.
3. Makes stealing user records useless by performing key rotation procedure.

Virgil Security, Inc. gives developers a security toolbox to protect their application data using end-to-end encryption and password security, and has built a service around the PHE protocol.

Authors of the PHE protocol: Russell W. F. Lai, Christoph Egger, Manuel Reinert, Sherman S. M. Chow, Matteo Maffei and Dominique Schroder.

Advantages of PHE

1. User password is never transmitted to the PHE service in any form.
2. Offline attacks are not possible. You cannot tell if a password was entered correctly before PHE service replies.
3. Scheduled or on-demand key rotation is a part of the protocol. This key rotation renders previous user enrollment records useless.
4. Compared to pairing-based protocols like Pythia, it is ~10 times faster with thousands of requests per core. Also, there is no need for third party crypto libraries, as standard NIST p-256 elliptic curve does the job.
5. For each user your backend stores a number T = A + B, where A is a number representing user’s password and B is a pseudorandom number received from PHE service. So the only way to check if the password is valid is to try to reconstruct B and send it back to the service so that it could verify it and prove you that it was done correctly.
6. PHE service proves each step with Zero Knowledge Proof and the backend validates all responses using the server’s public key. So there’s no way PHE server can compromise the backend with its answers.
7. User enrollment records are additionally protected by the salt and backend private key to which PHE service has no access.

Prerequisites overview

App Backend — application server that provides registration and authentication operations, stores users passwords and personally identifiable information (PII).

- **App Backend** performs cryptographic operations and communicates with PHE Service using a special SDK.
- **App Backend** owns the Private Key (X) that is generated by the developer using CLI on a local machine or on Virgil Security's Dashboard.
- **App Backend** gets a Public Key of the PHE service.

PHE Service — a standalone service in Virgil Infrastructure dedicated to implementing the password-hardened encryption protocol.

- **PHE Service** generates a keypair. The Private Key (Y) is stored on the PHE Service and the Public Key is sent to the App Backend.

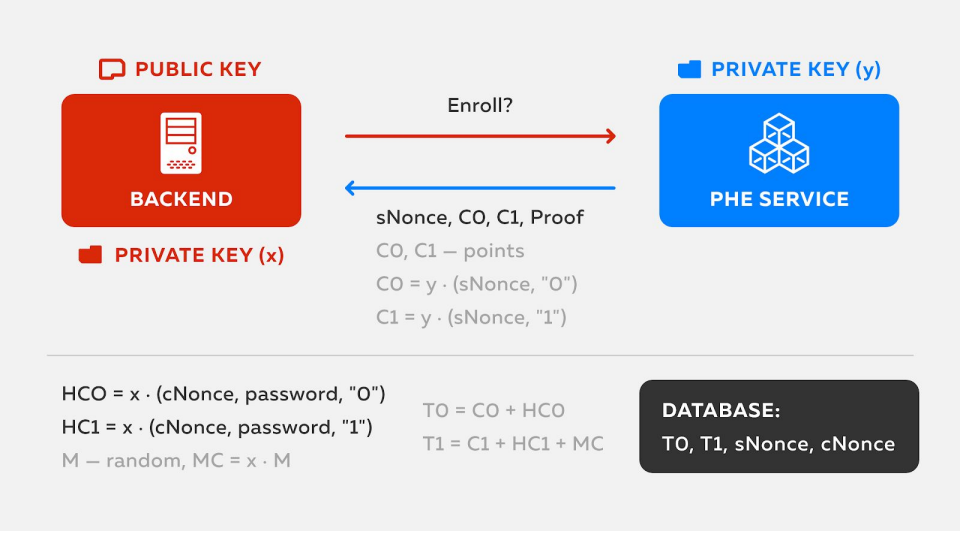
Cryptography inside:

Elliptic curve	NIST P-256
Hash	SHA-512
Hashing to the elliptic curve point	Shallue-Woestijne-Ulas algorithm
KDF	HKDF
PII Data Encryption	AES-256-GCM
Zero Knowledge Proof (ZKP)	Generalized Schnorr protocol

Record — a unique data that is associated with a specific user's password (1 password = 1 record).

Sign Up (Enrollment)

Creates an Enrollment Record for *each* user's password on PHE.



Phase #1. The App Backend asks PHE Server for an Enrollment

App Backend:

1. Sends empty request to server Enroll endpoint.

PHE Server:

1. Generates 32-byte random salt.
2. Hashes salt with two different domains into two curve points HS0 and HS1.
3. Performs scalar multiplication of HS0 and HS1 by its Private Key (Y) to get points C0 and C1.
4. Calculates Zero Knowledge Proof which proves that C0 and C1 were indeed calculated using server's Private Key (Y).
5. PHE Server replies with the following data:
    a. 32-byte random salt
    b. Points C0 and C1
    c. ZKP

Phase #2. The App Backend generates an Enrollment Record

App Backend:

1. Receives enrollment from the PHE Server.
2. Verifies ZKP using server's Public Key.
3. Generates 32-byte random salt.
4. Hashes a user's password using salt and two different domains into two elliptic curve points HC0 and HC1.
5. Generates random elliptic curve point M.
6. Multiplies HC0, HC1 and M by the App Backend's Private Key (X) to get points HC0`, HC1` and M`.
7. Calculates enrollment points T0 and T1 in the following way:
    a. T0 = C0 + HC0`
    b. T1 = C1 + HC1` + M`
8. Saves enrollment record to the database. Enrollment record contains:
    a. Server 32-byte random salt
    b. App Backend 32-byte random salt
    c. Points T0 and T1

Phase #3. User's PII (data) encryption

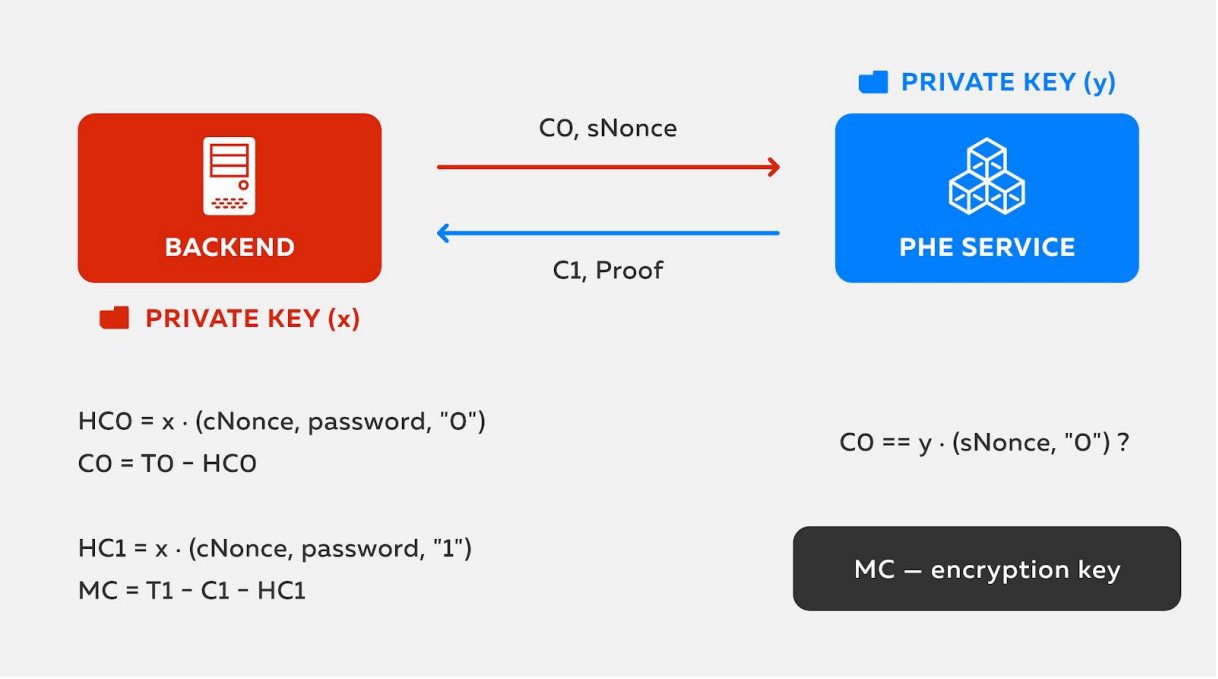
App Backend:

1. The point M is serialized and then put through HKDF to receive the Encryption Key (K).

2. Encrypts a user's PII data by performing the following steps:
    a. Generates 32-byte random Salt.
    b. Performs HKDF(K, salt) to receive per-data AES key and Nonce.
    c. Encrypt data with AES key and Nonce.
    d. Save ciphertext concatenated with Salt.

Login (Verify Record)

Verifies user's Record in the database at the login step.



Phase #1. App Backend extracts point C0 from T0

App Backend:

1. When a user tries to log in, they supply their user ID and password. The user ID is used to retrieve Enrollment Record from the database.
2. Then App Backend performs the following steps:
    a. Hashes user-provided password using salt from the Enrollment Record and the first domain into elliptic curve point HC0.
    b. Performs scalar multiplication of HC0 by App Backend's Private Key (X) to obtain HC0`.
    c. Subtracts HC0` from T0 to receive C0.
    d. Sends Server Salt and C0 to the PHE Server.

Phase #2. PHE Server verifies C0

App Backend:

1. The PHE Server receives Salt and C0.
2. Hashes Salt with the first domain into curve point HS0.
3. Performs scalar multiplication of HS0 by its Private Key (Y) to get the original point C0.
4. The server compares the calculated point C0 and the received C0.
5. If the original point C0 matches the received one, the PHE Server calculates point C1 in the same way it did during the enrollment process and sends C1 and ZKP back to the App Backend.
6. If the calculated C0 at the PHE Server does not match the calculated C0 at the App Backend then PHE Server calculates ZKP of incorrectness and sends it back to the App Backend.

Phase #3. App Backend completes password verification

App Backend:

When the App Backend receives the answer from the PHE Server it looks at the result, verifies the ZKP and then performs the following steps:

1. If the result is “verification failed” then abort the login process as the password was entered incorrectly.
2. If the result is “verification succeeded” then proceed with the login process and extract the Encryption Key:
   a. Calculates HC1` using Client Salt, in the same way, it did during the enrollment process.
   b. Extracts point M = (T1 − C1 − HC1`) * 1/x.
   c. Master Encryption Key (K) = HKDF (M).

Key Rotation (Update User’s Record)

Key rotation allows the App Backend and the PHE Server to jointly create a new set of keys and seamlessly update user records. It is performed in the following phases:

Phase #1. App Backend asks for the Update Token

1. App Backend asks PHE Server to rotate Keys.
2. The PHE Server creates two 256 bit random numbers, A and B.
3. The Server calculates its new Private Key (Y`) = Y * A + B.
4. PHE Server sends A and B to the App Backend. A and B are called the Update Token.

Phase #2. App Backend updates its keys

After the App Backend receives the Update Token, it is able to calculate new keys:

1. App Backend calculates its new Private Key (X`) = X * A.
2. App BAckend calculates new Server Public Key (PUB`) = (PUB * A) + (G * B) where G is P-256 base point.

Note that the PHE Server never sends its new Public Key to the App Backend. The App Backend is able to calculate it itself.

Phase #3. App Backend updates user records

After the App Backend receives the Update Token, it is able to calculate new values for the user enrollment records in the following way:

1. Retrieves the old record from the database.
2. Extracts T0 and T1 points.
3. Extracts Server's Salt Ns.
4. Calculates HS0 and HS1 points in the same way as during the Enrollment process.
5. T0` = (T0 * A) + (HS0 * B).
6. T1` = (T1 * A) + (HS1 * B).
7. Saves new T0` and T1` inside user enrollment record.

From now on, your backend is able to verify the same passwords and extract the same keys as before rotation but using the new set of keys you created with the help of Update Token.

> Note: User password and Encryption key (M) remain the same after key rotation, so that the rotation process is seamless.

For more information please visit VirgilSecurity.com/purekit

Last revision: April 2019