Argon2 Tabanlı Şifreleme/Hashleme Modülü ve Uyum Spesifikasyonu

Bu doküman, sistemimizde parolalar ve hassas token/kodlar için kullanılan Argon2id tabanlı hashleme yaklaşımını, parametreleri, entegrasyon akışlarını ve test vektörlerini teknik olarak açıklar. Yönetim ve diğer projeler ile uyumluluk sağlamak amacıyla hazırlanmıştır.

1) Algoritma ve Parametreler

- Algoritma: Argon2id (PHC String Format, v=19)
- Varsayılan parola hash parametreleri:
  - timeCost (t): 3
  - memoryCost (m): 65536 (≈ 64 MiB)
  - parallelism (p): 2
  - salt: 16 byte (kriptografik rastgele, her hash için farklı)
- Token/Kod hash parametreleri:
  - timeCost (t): 2 (diğer parametreler aynı)

Çıktı formatı PHC standardıdır ve şu şablona uyar: `$argon2id$v=19$m=<memory>,t=<time>,p=<parallelism>$<salt_base64>$<hash_base64>`

Notlar:

- PHC string içinde parametreler ve salt gömülüdür. Ek alan tutmaya gerek yoktur.
- Doğrulama aşamasında kütüphaneler gerekli parametreleri PHC string içinden okur.

2) Modül API'si (Önerilen)

- hashPassword(plain: string): Promise
  - 16 byte rastgele salt üretir, Argon2id ile t=3, m=65536, p=2 kullanır.
- verifyPassword(hash: string, plain: string): Promise
  - PHC string içinden parametre ve salt'ı okuyarak doğrular.
- hashToken(plain: string): Promise
  - 16 byte rastgele salt üretir, t=2 ile (diğerleri aynı) Argon2id uygular.
- verifyTokenHash(hash: string, plain: string): Promise
  - PHC string ile doğrulama yapar.

Node.js referans uygulaması için bkz. `src/shared/helpers/crypto.ts`.

3) Entegrasyon Akışları

- Parola Kaydetme:
  1. Kullanıcının parolasını `hashPassword` ile hashleyin.
  2. Ortaya çıkan PHC string'ini kullanıcı kaydında `password` alanına saklayın.
- Parola Doğrulama (Giriş):
  1. Kullanıcının kayıtlı PHC string'ini alın.
  2. `verifyPassword(storedPhc, candidatePlain)` çalıştırın.
  3. Sonuç `true` değilse giriş başarısızdır.
- Token/Kod (E-posta doğrulama, parola reset, refresh token ID):
  1. Üretilen kod/ID'yi `hashToken` ile hashleyin, PHC string'i veritabanında saklayın.
  2. Kullanıcıdan gelen kod/ID'yi `verifyTokenHash(storedPhc, receivedPlain)` ile doğrulayın.

4) Güvenlik Hususları

- Argon2id, GPU/ASIC dirençli bellek-zor fonksiyondur; `memoryCost=65536` uyumluluk için seçilmiştir.
- Salt rastgele ve her hash için benzersizdir. PHC içinde saklanır.
- Hash'ler geri döndürülemez; yalnızca doğrulama yapılır.

- Parametreler ileride donanım gücüne göre artırılabilir. PHC formatı sayesinde eski hash'ler doğrulanmaya devam eder.

5) Test Vektörleri (Deterministik)

Sabit salt ile üretilmiştir; yalnızca karşılaştırma/entegrasyon testleri içindir. Üretimde sabit salt kullanılmaz.

Salt (hex): `000102030405060708090a0b0c0d0e0f`

- Parola:
  - Plain: `P@ssw0rd!`
  - Parametreler: t=3, m=65536, p=2, argon2id
  - PHC:
    `$argon2id$v=19$m=65536,t=3,p=2$AAECAwQFBgcICQoLDA0ODw$USxA6CUhf8+EdMRdqSJkjCsZk6JNOwe4Ax+QKwsP3eQ`
- Token:
  - Plain: `12345678-1234-1234-1234-1234567890ab`
  - Parametreler: t=2, m=65536, p=2, argon2id
  - PHC:
    `$argon2id$v=19$m=65536,t=2,p=2$AAECAwQFBgcICQoLDA0ODw$qzBXfVfjKnj/GEE8M8gou3dbmz34lLVOyMXQki605I4`

Kaynak dosya: `src/docs/argon2-test-vectors.txt`

6) Dil/Kütüphane Uyumluluğu

- Node.js: `argon2@^0.40.1` ile tam uyumlu.
- Python: `argon2-cffi` ile PHC string'ler desteklenir.
- Diğer dillerde de (Go, Java, .NET) PHC formatını destekleyen kütüphanelerle birlikte çalışır.

7) Konfigürasyon ve Versiyonlama

- Parametreler kodda sabitlenmiştir; merkezi konfigürasyona taşınabilir.
- Versiyon yükseltmeleri (örn. timeCost artırımı) aşamalı uygulanabilir. Yeni kayıtlar yeni parametreyle hashlenir; doğrulama eski parametrelerle de çalışır.

8) Güvenli Kullanım Önerileri

- Üretimde güçlü parola politikası ve oran sınırlama (rate limit) uygulanmalıdır.
- Hatalı girişlerde zaman tabanlı koruma ve hesap kilitleme kullanılmalıdır.
- Hash'ler yalnızca PHC string olarak saklanmalı; salt ayrı alanlarda tutulmamalıdır.
- Log'larda PHC string'leri ve düz metin değerleri yazmayın.

Soru ve entegrasyon desteği için ilgili modül: `src/shared/helpers/crypto.ts`.