Secure Auth RS - Security Features

Comprehensive Security Architecture Documentation

Version: 1.0
Date: November 13, 2025
Security Classification: Internal Use Only

1 Cryptographic Security

1.1 SHA512 for TOTP Operations

- All TOTP (Time-based One-Time Password) operations use SHA512 algorithm
- Provides stronger security compared to default SHA1
- Implemented in `src/crypto/totp.rs` with consistent SHA512 usage across all TOTP functions
- Prevents hash collision attacks and provides future-proofing

1.2 Argon2id for CPR Hashing with PEPPER Protection

- CPR (Danish Personal ID) numbers are hashed using Argon2id algorithm
- Secure parameters: 19456 KiB memory, 2 iterations, 1 parallelism
- PEPPER protection via environment variable `CPR_PEPPER` prevents rainbow table attacks
- CPR is combined with pepper before hashing: `format!("{}:{}", cpr, pepper)`
- Memory is securely cleared using `zeroize` after processing
- Located in `src/crypto/cpr.rs`

1.3 AES-256-GCM for TOTP Secret Encryption

- TOTP secrets are encrypted at rest using AES-256-GCM
- Random nonces for each encryption operation
- Tamper detection through authentication tags
- Prevents TOTP secret exposure even with database access

2 Authentication & Authorization

2.1 Passwordless Authentication with 16-Character Account IDs

- Account IDs are 16-character alphanumeric strings [A-Za-z0-9]
- Generated using cryptographically secure `OsRng`
- Provides 95 bits of entropy ($16 \times \log_2(62)$)
- No passwords to steal or crack
- Implemented in `src/crypto/account.rs`

2.2 Three-Phase Authentication Flow

1. **Authentication**: Account ID + TOTP verification = Authenticated (but not authorized)
2. **Authorization**: CPR number submission and validation = Authorized
3. **Access**: Full access to protected endpoints

2.3 Mandatory Multi-Factor Authentication

- **TOTP 2FA MUST**: All users must complete TOTP verification
- **CPR MUST**: All users must submit and validate CPR number
- Both factors are enforced at different layers for defense in depth

3 Transport & Session Security

3.1 HTTPS with Password-Protected Certificates

- TLS certificates are password-protected via `TLS_KEY_PASSWORD` environment variable
- Application-level password validation with constant-time comparison
- Prevents unauthorized certificate usage even with file access
- Proper TLS configuration in `src/tls/mod.rs`

3.2 HttpOnly Secure Cookies

- Authentication cookies use `http_only(true)` to prevent XSS attacks
- `secure(true)` ensures cookies only sent over HTTPS
- `same_site(SameSite::Strict)` provides CSRF protection
- Located in `src/middleware/auth.rs`

4 Endpoint Protection

4.1 Universal CPR Authorization

- All protected endpoints require CPR authorization
- Implemented via `cpr_protected_routes` middleware in `src/main.rs`
- Only exception: `/api/account/cpr` endpoint (necessary for CPR submission)
- Located in `src/middleware/cpr.rs`

4.2 Protected Admin Endpoints

- Admin endpoints require database-driven admin privileges
- `require_admin` middleware checks `account_roles` table
- Admin-only routes: `/api/admin/users`, `/api/admin/users/{account_id}`
- Located in `src/routes/admin.rs` and `src/middleware/auth.rs`

5 Additional Security Features

5.1 CSRF Protection

- One-time use CSRF tokens for all POST routes
- 1-hour token expiration
- Prevents cross-site request forgery attacks
- Implemented in `src/middleware/csrf.rs`

5.2 Rate Limiting

- IP-based rate limiting (5 requests/minute for auth endpoints)
- GDPR-compliant implementation
- Prevents brute force attacks
- Located in `src/middleware/rate_limit.rs`

5.3 PII Protection

- Custom Debug implementations redact sensitive data (CPR, TOTP secrets)
- Prevents accidental logging of personal information
- Located in `src/db/models.rs`

5.4 Memory Security

- Sensitive memory is cleared using `zeroize`
- Constant-time comparisons for password verification
- Prevents memory-based attacks and timing attacks

6 Security Assessment Summary

All claimed security features are fully implemented with the following observations:

6.1 Strong Cryptographic Practices

- Uses industry-standard algorithms (SHA512, Argon2id, AES-256-GCM)
- Proper key generation and management
- Secure random number generation

6.2 Proper Memory Management

- Implements zeroization for sensitive data
- Constant-time operations for security-critical comparisons
- Prevents data leakage through memory dumps

6.3 Layered Security Architecture

- Multiple independent security controls
- Defense in depth approach
- No single point of failure

6.4 Compliance and Privacy

- GDPR-compliant rate limiting and data handling
- Thoughtful implementation of PII protection
- Audit logging for security events