SurePack - Deep Analysis

Overview

The Public Key Server project is a sophisticated secure file transfer system that implements the Anonymous Certificate Enrollment (ACE) protocol. It provides end-to-end encrypted file sharing with anonymous identity capabilities, similar to PGP but with enhanced usability and anonymity features.

Architecture

Components

1. **Suredrop (Server)** - REST API server providing certificate management and file relay services
2. **Surepack (Client)** - Command-line and GUI client for creating, sending, and receiving encrypted packages
3. **ACE Protocol** - Anonymous Certificate Enrollment protocol based loosely on EST (RFC7030)

Key Technologies

- **Cryptography**: Bouncy Castle library for cryptographic operations
- **Classical Encryption**: RSA 2048-bit for key exchange
- **Post-Quantum Encryption**: Kyber1024 for quantum-resistant key encapsulation
- **Symmetric Encryption**: AES-256-GCM for file encryption
- **Hashing**: SHA-512 for signatures and verification
- **Storage**: Amazon S3 for certificate and package storage
- **Framework**: ASP.NET Core for the server, .NET for the client

Core Features

1. Anonymous Certificate System

The system generates anonymous aliases using three random words (e.g., `crow-mandate-current.publickeyserver.org`). Key features:

- **No Identity Required**: Users can obtain certificates without revealing their identity
- **Automatic Alias Generation**: Three-word phrases ensure uniqueness and memorability
- **Domain-based Validation**: Certificates can be verified by accessing the alias as a URL
- **Optional Identity Binding**: Users can optionally bind certificates to email addresses

2. Hybrid Encryption System

The project implements a sophisticated multi-layer encryption approach:

```
File → AES-256-GCM → RSA-2048 → Kyber1024 (Post-Quantum)
```

- Files are encrypted with AES-256-GCM using a random 256-bit key

- The AES key is encrypted with the recipient's RSA public key
- The RSA-encrypted key is further encrypted with Kyber1024 for quantum resistance
- Each recipient gets their own encrypted copy of the AES key

3. Surepack File Format

Surepacks are specially formatted ZIP files containing:

```
surepack.zip/
├── envelope               # JSON metadata (recipients, sender, algorithms)
├── envelope.signature     # Digital signature of envelope
├── manifest              # Encrypted file listing and metadata
├── manifest.signature    # Digital signature of manifest
└── [encrypted blocks]    # File data split into encrypted chunks
```

4. Certificate Lifecycle

- **Short-lived Certificates**: Designed to be temporary and disposable
- **Automatic Enrollment**: Simple POST request with public key
- **Custom Extensions**: Support for arbitrary data in X.509 extensions (OID: 1.3.6.1.4.1.57055)
- **Root CA Management**: Encrypted CA certificate storage with AES-GCM

Security Model

Cryptographic Security

1. **End-to-End Encryption**: Private keys never leave the client device
2. **Perfect Forward Secrecy**: Each package uses a unique AES key
3. **Post-Quantum Resistance**: Kyber1024 provides protection against quantum attacks
4. **Digital Signatures**: All packages are signed with SHA512withRSA
5. **Certificate Validation**: Multi-step validation including root fingerprint verification

Anonymity Features

- **Anonymous Enrollment**: No authentication required for certificate generation
- **Random Aliases**: Three-word combinations provide sufficient entropy
- **Optional Identity**: Users can choose between anonymous or identity-bound certificates
- **No Tracking**: Server doesn't store user information beyond certificates

Server Security

- **Encrypted Storage**: CA certificates stored with AES-GCM encryption
- **AWS Security**: Leverages Amazon S3 security features
- **Rate Limiting**: Package size and count limits to prevent abuse
- **Signature Verification**: All API calls require cryptographic signatures

API Endpoints

Certificate Management

- `GET /status` - Server status and statistics
- `GET /cacerts` - Retrieve CA certificate chain
- `POST /simpleenroll` - Enroll a new certificate with provided public key
- `GET /cert/{alias}` - Retrieve certificate by alias
- `GET /identity/{identity}` - Retrieve certificate by identity
- `DELETE /{alias}` - Delete a certificate

Package Management

- `POST /package/{recipient}` - Upload an encrypted package
- `GET /package/{recipient}/{package}` - Download a package
- `GET /list/{recipient}` - List available packages

Verification

- `GET /verify/{alias}` - Verify a certificate is valid

Client Operations

Surepack Commands

1. **create** - Generate a new certificate and key pair
2. **pack** - Create an encrypted surepack from files
3. **unpack** - Decrypt and extract a surepack
4. **send** - Upload a surepack to the server
5. **receive** - Download packages from the server
6. **list** - Show available packages
7. **verify** - Verify a certificate
8. **certify** - Certificate management operations
9. **delete** - Remove certificates or packages
10. **gui** - Launch graphical interface

Workflow Example

```
# Create a new identity
surepack create -a myalias@suredrop.org

# Pack files for recipients
surepack pack -i "*.pdf" -a recipient1,recipient2 -f myalias -o package.surepack

# Send the package
surepack send -i package.surepack

# Recipients can receive
surepack receive -a recipient1
```

Key Differentiators from PGP

1. **Ease of Use**: No complex key management or web of trust
2. **Anonymous by Default**: No identity required for basic usage
3. **Modern Cryptography**: Post-quantum encryption built-in
4. **Integrated Delivery**: Built-in server for package relay
5. **Automatic Key Discovery**: Keys retrieved automatically via aliases
6. **Simplified Validation**: URL-based certificate verification

Implementation Details

File Encryption Process

1. Files are compressed (GZIP/Brotli)
2. Split into blocks for streaming
3. Each block encrypted with AES-256-GCM
4. Blocks stored in the surepack with metadata

Certificate Validation

1. Retrieve certificate from server
2. Verify certificate chain to root CA
3. Check root fingerprint matches expected value
4. Validate sender and recipient share same root CA
5. Verify digital signatures on envelope and manifest

Package Limits (Configurable)

- Maximum package size: Configurable (default varies)
- Maximum bucket size: Per-recipient storage limit
- Maximum file count: Prevents resource exhaustion

Security Considerations

1. **Root Key Security**: The root CA private key must be carefully protected
2. **Password Protection**: Local private keys encrypted with user password
3. **Network Security**: All communications over HTTPS
4. **Temporary Storage**: Packages deleted after download
5. **No Persistent Storage**: Server doesn't retain decrypted data

Future Enhancements

Based on the codebase analysis, potential improvements could include:

1. **Multiple Root CAs**: Support for federated trust models
2. **Revocation Lists**: Certificate revocation mechanisms
3. **Audit Logging**: Enhanced logging for security monitoring
4. **Mobile Clients**: iOS/Android implementations
5. **Browser Extension**: Web-based encryption/decryption

Production Deployment

The reference implementation runs at https://suredrop.org with:

- AWS S3 for scalable storage
- ASP.NET Core on Linux servers
- Let's Encrypt for TLS certificates
- Serilog for structured logging

Conclusion

The Public Key Server project successfully addresses many of the usability challenges of traditional encryption systems like PGP while maintaining strong security guarantees. Its unique combination of anonymous certificates, post-quantum cryptography, and integrated delivery makes it a compelling solution for secure file transfer in an increasingly surveilled digital world.