ASPECT ORIENTED REFACTORING OF A REMOTE PASSWORD MANAGEMENT SYSTEM

Abdullah Atmaca, Cem Mergenci, Muhammet Kabukçu

Agenda

- Password Management
- Requirements Analysis
- Object Oriented Design
  - Patterns
- Problem Statement
  - Crosscutting Concerns
- Aspect Oriented Programming
- Discussion

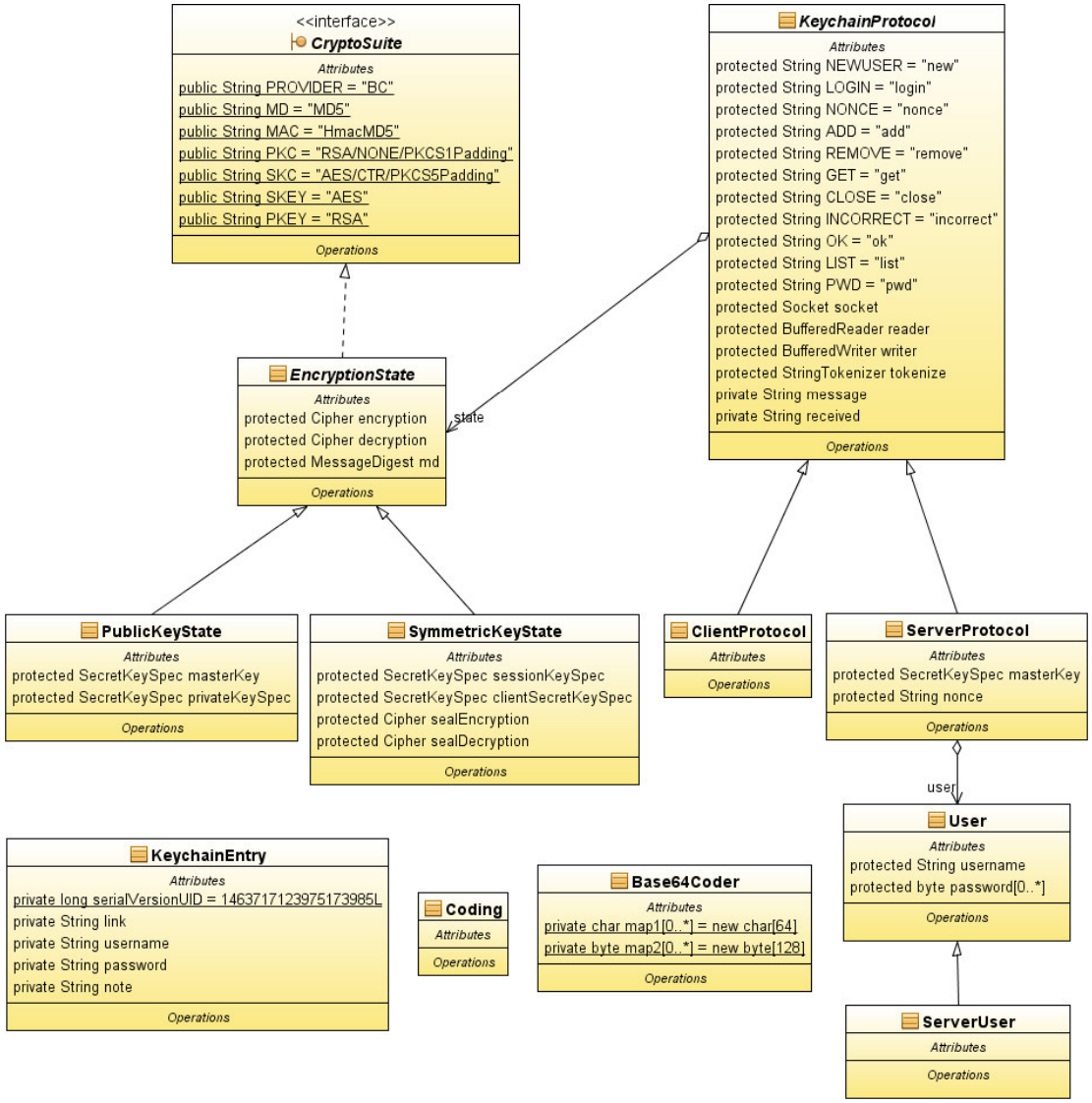
Password Management

- Manage all accounts with a single account.
- Operating systems, browsers, mail clients…
- Beware! All accounts depend on a single password.
- Local to the system.
- Remote Keychain (RKC)
  - Runs over network.
  - Easy access from everywhere.

Requirements Analysis

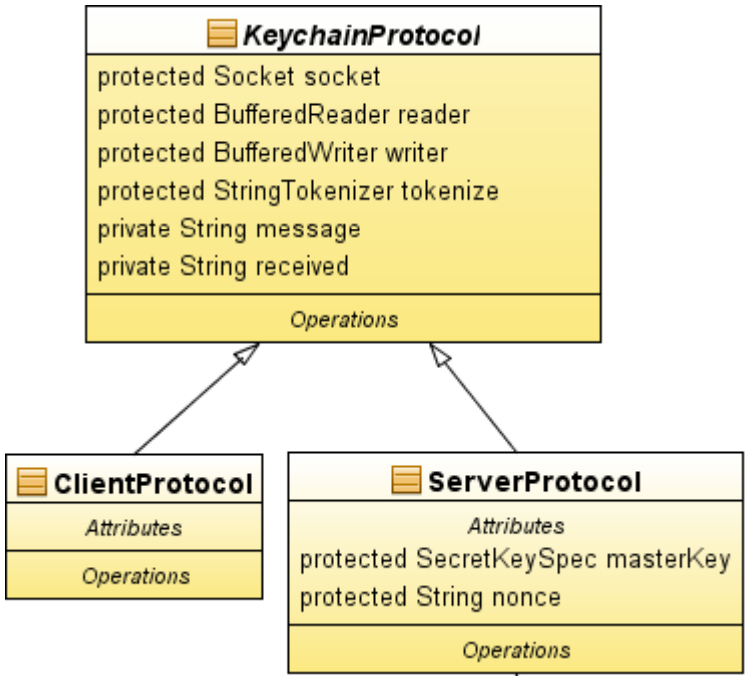
- Server supports multiple concurrent users.
- Authentication with master username and password.
- Add, delete, modify, retrieve account details.
  - Username, password, personal notes.
- Change master password.
- Master password is not recoverable!
  - Security!

Object Oriented Design



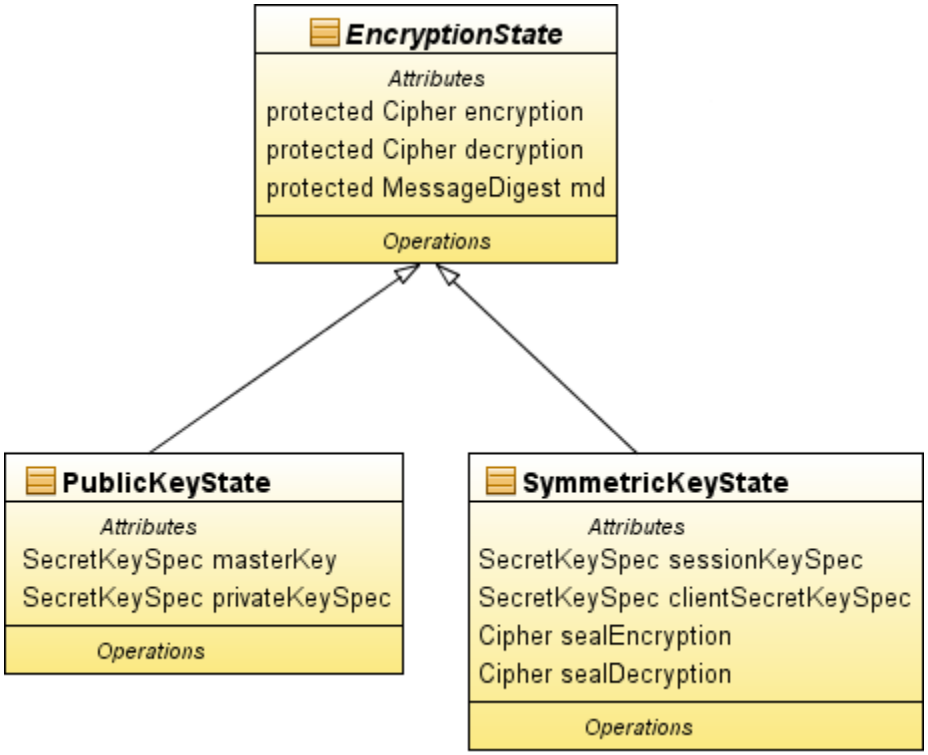
Patterns

- Strategy



Patterns

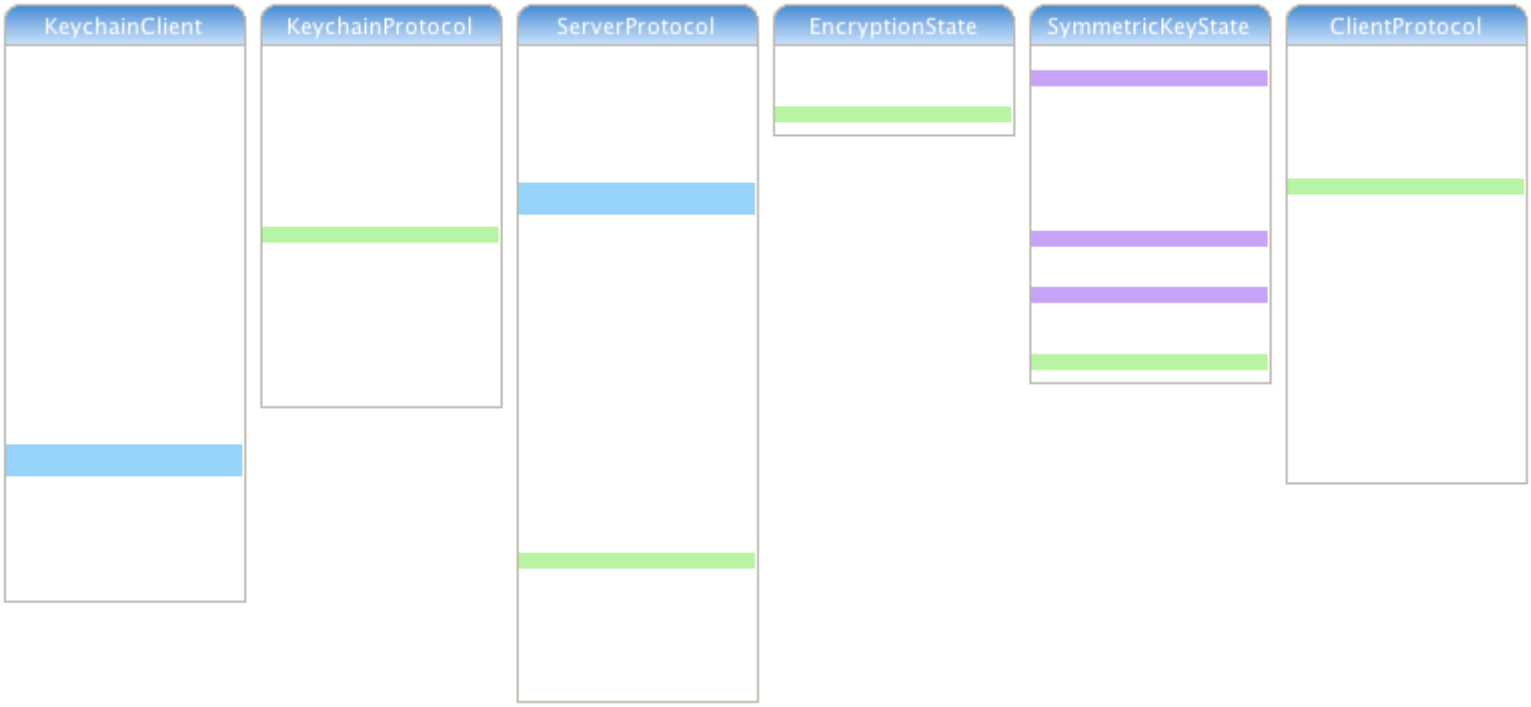
- State



Problem Statement

- OO design patterns solves scattering and tangling.
  - But not enough.
- Crosscutting concerns
  - Authentication
  - Decryption
  - Message Authentication
- Enforcement of
  - Crypto Responsibility
  - Network Stream Access

Crosscutting Concerns



Authentication Aspect

- Crosscuts ServerProtocol and ClientProtocol
- In case a modification, change both.

Listing 1. Authentication Aspect Pointcuts

```
1  public aspect Authentication {
2    pointcut clientSide( ClientProtocol p,
3          String username, String password ) :
4      (call( * ClientProtocol.login( String, String ) ) ||
5       call( * ClientProtocol.newUser( String, String ) ))
6      && args( username, password ) && target( p );
7
8    pointcut serverSide( ServerProtocol p,
9          String username, String password ) :
10     (call( * ServerProtocol.login( String, String ) ) ||
11      call( * ServerProtocol.newUser( String, String ) ))
12     && args( username, password ) && target( p );
13 }
```

Authentication Aspect

Listing 2. Authentication Aspect Advises

```
1  public aspect Authentication {
2    User around( ClientProtocol p, String username,
3      String password ) : clientSide(p, username, password)
4    { // client side authentication logic }
5
6    User around( ServerProtocol p, String username,
7      String password ) : serverSide(p, username, password)
8    { // server side authentication logic }
9  }
```

Decryption Aspect

- Initialization using initialization vectors (IV).
- Decryption requires decoding.

Listing 3. Decryption Aspect Introductions

```
1  public aspect Decryption
2  {
3    public IvParameterSpec.new( String s )
4    { // decode string and return IV spec }
5
6    public byte [] Cipher.doFinal( String s )
7    { // decode string and decrypt }
8  }
```

Message Authentication

- Message Authentication Code (MAC)
  - Sent along the message.
  - Verifies sender.
- Originally not implemented.
- Assume we cannot modify or extend the code.
- Implement as an aspect.

Message Authentication

Listing 4. Message Authentication Aspect

```
1  public aspect MessageAuthentication
2  {
3    private Mac mac;
4
5    /* Pointcuts */
6
7    pointcut appendMac( String message ) :
8      execution(* SymmetricKeyState+.prepareMessage(..))
9      && args( message );
10
11   pointcut checkMac( BufferedReader reader ) :
12     execution(* SymmetricKeyState+.receive(..))
13     && args( reader );
14
15   pointcut initMac( byte [] sessionKey ) :
16     execution(SymmetricKeyState.new(..))
17     && args( sessionKey );
18
19   /* Advises */
20
21   String around( String message ) : appendMac( message )
22   { // append mac to prepared message }
23
24   String around( BufferedReader reader ) :
25     checkMac( reader )
26   { // check integrity of received message }
27
28   after( byte [] sessionKey ) : initMac( sessionKey )
29   { initMac( sessionKey ); }
30
```

Enforcement Aspects

- Crypto Responsibility

Listing 5. Crypto Responsibility Enforcement Aspect

```
1  public aspect CryptoResponsibility
2  {
3    pointcut cipherInstance() :
4      call(* Cipher.getInstance(..)) &&
5      within( rkc.** ) && !within( EncryptionState+ );
6
7    pointcut messageDigestInstance() :
8      call(* MessageDigest.getInstance(..)) &&
9      within( rkc.** ) && !within( EncryptionState+ );
10
11   declare error :
12     cipherInstance() || messageDigestInstance() :
13       "Crypto related tasks should be performed in a
14         subclass of EncryptionState.";
```

Enforcement Aspects

- Network Stream Access

Listing 6. Stream Access Enforcement Aspect

```
1  public aspect StreamAccessEnforcement
2  {
3    pointcut ReaderStreamAccessEnforcement() :
4      get(BufferedReader KeychainProtocol.reader) &&
5      !( within( KeychainProtocol ) ||
6         within( Authentication ) ||
7         withincode(* ClientProtocol+.login(..) )
8         ) );
9
10   pointcut WriterStreamAccessEnforcement() :
11     get(BufferedWriter KeychainProtocol.writer) &&
12     !( within( KeychainProtocol ) ||
13        within( Authentication ) ||
14        withincode(* ServerProtocol+.login(..) ) ||
15        withincode(* ServerProtocol+.newUser(..) )
16       ) );
17
18   declare error : ReaderStreamAccessEnforcement() :
19     "Reader stream should not be accessed from here.";
20
21   declare error : WriterStreamAccessEnforcement() :
22     "Writer stream should not be accessed from here.";
23 }
```

Discussion

- OOD Patterns are useful.
  - Resolves some of scattering and tangling problems.
  - But limited.
- AOSD
  - Captures aspects as first class entities.
  - Inter-type declarations are helpful.
  - Implement new features without modifying code.
- AJDT
  - Sometimes problematic?

Conclusion

- RKC, remote password management utility
- OOD and patterns
  - Strategy and Strategy
- Crosscutting concerns
  - Authentication, Decryption, Message Authentication
- Solution using aspects.

Demo

Q&A

Thanks…