Modeling Cryptographic APIs using Clafer

Background. Many web applications and mobile apps are now gathering private user data. To properly protect this data, cryptographic algorithms have to be used. Cryptographic algorithms are typically offered as components in software libraries, e.g., Bouncy Castle [1] or FlexiProvider [2] with the aim of providing application developers a simple way to secure their data. However, correct use and composition of these library components is a challenging task for the average software developer. Usage mistakes that lead to severe security vulnerabilities have been shown in previous work [3, 4]. For example, Georgiev et al. [3] showed that many applications misuse SSL certificate validation leading to sensitive information (e.g., credit card data) being leaked. The main problem is that application developers lack the necessary expertise and specialized domain knowledge to understand many of the intricate details of cryptographic APIs. To overcome this, an automated support system is needed to guide developers through their cryptographic needs. For example, if the developer's task is to securely store a password, the system should automatically identify the relevant cryptographic algorithms and generate the correct code that uses these algorithms. Additionally, static analysis of the final application code is needed to ensure the correct integration of the generated code.

To achieve this, we need to have an accurate model of the whole cryptographic domain, and more specifically, we need to model cryptographic APIs. The code in Figure 1 shows an example of a model that describes parts of two of the encryption algorithms found in the cryptography domain. The model is written in *Clafer* which is a modeling language that combines concepts from feature modeling [5] with UML modeling. The model shown simply defines a concept called Algorithm which has attributes such as name as well as security, performance, and memory consumption levels. Additionally, AES and DES are two encryption algorithms of type SymmetricBlockCipher with concrete values for the relevant attributes. A model in Clafer can also contain dependencies and constraints between different concepts (not shown for simplicity).

```
abstract Algorithm
    Name -> string
    Security -> integer ?
    Performance -> integer ?
    Memory -> integer ?

abstract Cipher : Algorithm

abstract BlockCipher : Cipher

abstract SymmetricBlockCipher : BlockCipher

abstract AsymmetricBlockCipher : BlockCipher
```

```
AES : SymmetricBlockCipher
    [ Name = "AES" ]
    [ Performance = 7 ]
    [ Memory = 4 ]
    [ Security = 10 ]

DES: SymmetricBlockCipher
    [ Name = "DES"]
    [ Performance = 5 ]
    [ Memory = 2 ]
    [ Security = 10 ]
```

Figure 1: Example of modeling encryption APIs in Clafer

Thesis Goals. The goal of this thesis is to model the full set of APIs provided by one of the cryptographic libraries (e.g., Bouncy Castle or FlexiProvider) using the Clafer notation shown above. The information to be modeled can be identified by analyzing the library's Java implementation as well as reading the related documentation. The developed model can be later used to select the correct cryptographic components for a developer and to derive any usage protocols of the APIs.

Deliverables. The following is the set of expected deliverables:

- A full model including the following: (1) all classes, methods, etc. in the API, (2) all method parameters, (3) constraints on the method parameters (e.g., default values or allowed ranges for proper behavior), and (4) constraints on combinations of algorithms in the API (e.g., if the documentation or implementation suggests that two algorithms should not be combined together).
- A report on any limitations of the Clafer modeling language. This includes a description of any parts of the APIs that cannot be properly modeled in Clafer and the corresponding assumptions or workarounds used to overcome such limitations.

Please note that this can also be done as a paid HiWi project.

For more information, please contact Dr. Sarah Nadi (nadi@st.informatik.tu-darmstadt.de) from the STG group.

References

[1] "Bouncy Castle." `www.bouncycastle.org`.

[2] "FlexiProvider." `www.flexiprovider.de`.

[3] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: Validating SSL certificates in non-browser software," in *Proc. of the Conference on Computer and Communications Security (CCS)*, pp. 38–49, 2012.

[4] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why Eve and Mallory love Android: An analysis of android SSL (in)security," in *Proc. of the Conference on Computer and Communications Security (CCS)*, pp. 50–61, 2012.

[5] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," tech. rep., CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.