Building a Configurator for Cryptograhic Components (HiWi Position)

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], often leading to sensitive data (e.g., credit card information) 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 a small example of a model. The model is written in *Clafer* which is a modeling language that combines concepts from feature modeling [5] with class 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.

```
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

Job Description. The goal of this project is to implement an Eclipse plugin that provides a configurator which interfaces with an existing Clafer model. The configurator reads the model and presents to the user the different tasks they can perform (e.g., store password, encrypt text, etc.). The user will select their required task as well as specify any additional constraints they have. For example, they might want an algorithm with the lowest memory consumption. The user's selections need to be translated into a query that Clafer can understand. After querying the model, the results (i.e., the suitable cryptographic components) would be presented to the user. The particular tasks of this job are:

- Building the GUI for the configurator as an Eclipse plugin
- Developing the interface between the configurator and the clafer model
- Identifying (and implementing) the mapping between the user's choices and the query presented to Clafer
- Reading Clafer query results and presenting them to the user

Skills. Applicants are expected to have the following set of skills:

- Excellent Java programming skills

- Previous GUI design experience
- Experience implementing Eclipse plugins (a plus)
- Cryptography background knowledge (a plus)
- Background about feature models (a plus)
- Knowledge of Haskell (a plus – Clafer is implemented in Haskell. However, understanding Clafer internals should not be necessary for this 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.