Table of Contents

```
class Provider<RQSTS,KSTR...> {
  assume OWNER->KSTR;
  KSTR LocalKeyStore<KEYID> keyStore; // (1)
  OWNER EngineWrapper<KSTR...> engine;

 Provider(KSTR LocalKeyStore<KEYID> store) {
   // Inject architectural violation
   this.keyStore = store; // (2)
   this.engine = new EngineWrapper(store);
 }
}
```

Figure 11: Injected architectural violation.

officers and the cryptographic engine" (p. 71). The key manager is the architectural agent that security officers use, hence we arrive at constraint 3.

Once we wrote these constraints, we formalized them using the Acme predicate language [16], as follows:

1. `forall c : Component in KeyManagement.MEMBERS | !connected(c, EngineWrapper)`
2. `!pointsTo(KeyVault, KeyManifest)`
3. `forall c : SyncCompT in self.COMPONENTS | pointsTo(c, KeyVault) -> c.label=="KeyManager" or c.label=="EngineWrapper"`

The full Acme specification of the target architecture, including the architectural style and the definition of the `pointsTo` predicate above, is in the appendix [9].

4.3 Evaluation summary

Scholia was able to successfully relate the security architecture and the implementation.

Renames. Because Scholia uses a structural comparison algorithm to compare the built and designed architectures, it can analyze conformance despite the naming discrepancies—e.g., `KeyManager` versus `KeyTool`.

Conformance findings. Overall, the top-level components in the target architecture (based on a Level-1 DFD) and the implementation were mostly consistent, as indicated by the large number of convergences (Fig. 10).

Drilling down into the representations of the some of the top-level components revealed more interesting differences. For example, a Level-2 DFD (in the appendix [9]) shows an Encoder component inside the Provider. However, the Encoder is implemented using a helper class `Utils`, which is never instantiated. Hence, the corresponding absence in the conformance view. We could resolve this absence by modifying the code to instantiate a singleton `Utils` object, without affecting the system's behavior.

In the process of modeling the target architecture, we confronted a number of architecture–implementation discrepancies of this nature. We ultimately dealt with them, in most cases, by modifying the target architecture to match the implementation. This was necessary because of the departures that the CryptoDB implementation made from the cryptographic database architecture. Had we not reconciled the differences at that stage, we would have had much more noise to sort through in the conformance operation. Naturally, distinguishing between deliberate departures from the architecture and genuine architecture violations requires careful judgment. However, we view it as a strength of our iterative approach that architects have the opportunity to exercise their judgment in this way to forestall uninteresting violation reports from the tool.

In other cases, we refined the annotations. For instance, we had initially modeled all instances of `CryptoReceipt` and `CompoundReceipt` in a `RECEIPTS` domain inside the `CustomerManager`. As a result, the analysis flagged the `ReceiptManager` inside the `CryptoProvider` as an absence. Then we looked more carefully at how the `Provider` and the `CustomerManager` exchanged these objects (Fig. 7). This led us to define a `RCPTMGR` domain inside `provider` for `CompoundReceipts`, and left the `CryptoReceipts` in the

RECEIPTS domain inside mgr (Fig. 9).

Constraint violations. Once we added the constraints to the target architecture, we used the AcmeStudio tool to verify them. Due to the traceability we established between the architecture and source code, we can have some confidence that the implementation meets these constraints.

To further validate our approach, we modified the CryptoDB code, injecting a manufactured architecture violation to confirm that our constraints would catch it. Specifically, we coupled the Provider and the LocalKeyStore as shown in Fig. 11. According to constraint 3 above, the Provider is not allowed to point to the LocalKeyStore in this way. In the architecture, access to the key vault is highly restricted due to the sensitivity of the contents.

When we modified the code in this way and ran our analysis, the predicate raised a warning about the architectural violation in the conformance view. It is true that enforcing predicates at the architectural level is not novel. But since our approach establishes traceability between the architecture and the code, enforcing constraints at the architectural level allows enforcing global constraints on the application structure in the code. In addition, the domain link checks alone would not have caught this violation. Both engine and provider are peers in the same PROVIDERS domain (Fig. 8). So, there must already be a domain link from PROVIDERS to KEYSTORAGE for engine to access the key vault.

5 Related Work

Architectural security analysis. Various architectural-level security analyses have been proposed [20, 21]. For example, UMLsec [22] extends UML with secrecy, integrity and authenticity, to allow analyzing security weaknesses at the design level. However, conformance between the architecture and the implementation is achieved using code generation, code analysis, and test-sequence generation. Code generation, while potentially guaranteeing the correct refinement of an architecture into an implementation, is often too restrictive to be fully adopted on a large scale and cannot account for legacy code. One could use the approach in this paper to analyze an existing system, after the fact, by adding annotations to the code.

Conformance analysis. There are many approaches to analyze conformance to a code architecture (see Knodel and Popescu for a comparative analysis [23]). However, the tool support for analyzing, *statically*, communication integrity in a runtime architecture is much less mature. SCHOLIA is modeled after, and complements, Reflexion Models [13], which handles the code architecture only.

Language-based solutions. Like ArchJava [6], SCHOLIA integrates architectural intent into source code, but instead of extending Java with architectural components and ports, SCHOLIA uses language support for annotations. The evaluation in this paper did not require re-engineering a system to follow ArchJava's rules [10]. In this paper, we only added annotations to the code and typechecked them using a tool.

Abi-Antoun et al. also added ownership domain annotations to several subject systems [11, 24]. The study in this paper has novel aspects. We added domain links (they were part of the formal model, but previously not supported by the tools) and reasoned about Strings instead of marking them shared. Moreover, the CryptoDB target architecture was drawn by a security expert instead of a professor [7], and has richer types, properties and constraints than the previous architectures that SCHOLIA analyzed, which increases the external validity of the result.

Code generation. SecureUML [25] recommends a model-driven approach in which security constraints are imposed on a model that is later elaborated into code. Of course, like all model-driven approaches, it is useful only for construction of new systems, not for analysis of existing implementations. Our approach is appropriate for use on existing code, requiring only annotations. Another difference is that SecureUML is based on a code architecture.

Code-level analyses. Architectural analysis matches the way experts reason about security or privacy better than a purely code-based strategy. Our approach complements, and does not supplant, code-level analyses. Moreover, the traceability between a security architecture and the code that our approach derives can benefit other static analyses. Until now, due to the lack of traceability, much of the security design intent generated during threat modeling has not been easily accessible to other code quality tools. For instance,

a static analysis checking for buffer overruns [26] can use this traceability to assign to its warnings more appropriate priorities based on a more holistic view of the system.

Security testing. Analysis offers substantial benefits beyond those of testing alone. Perhaps most significantly, since our approach is based on static analysis, it can reveal information about all possible runs of a program, while testing is limited to a small number of runs. This difference is particularly important in the security domain. Similar to testing is dynamic conformance analysis, which instruments and monitors a system [27, 4].

Design enforcement. Many approaches can enforce local, modular, code-level constraints, e.g., [28]. Our approach is complementary, and can enforce structural constraints on the global runtime architectural structure.

6 Conclusion

We presented the first approach to relate, entirely statically, a security runtime architecture to a program written in a widely used object-oriented language, using annotations. Such an approach can increase the effectiveness of reasoning architecturally about the security of existing systems, because it ensures that the architecture is a faithful representation of the code, which is ultimately the most reliable and accurate description of the built system.

Acknowledgements

The authors would like to thank Jonathan Aldrich, David Garlan, Bradley Schmerl and Nenad Medvidovic for helpful feedback.

APPENDIX

A Documented Architectures

Fig. 12 is a Level-1 DFD.

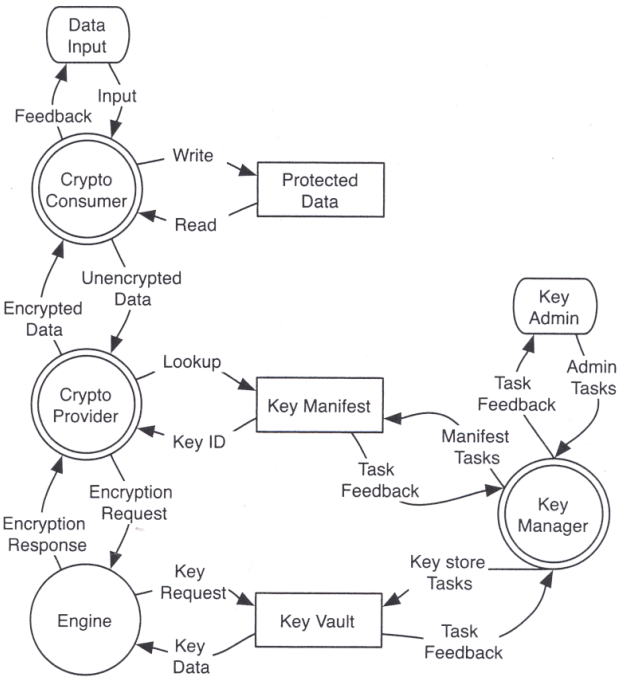

Figure 12: CryptoDB documented DFD (Level 1) [17, Fig. 9.1].

Fig. 13 is a Level-2 DFD which refines in place some of the components in the Level-1 DFD (Fig. 12).

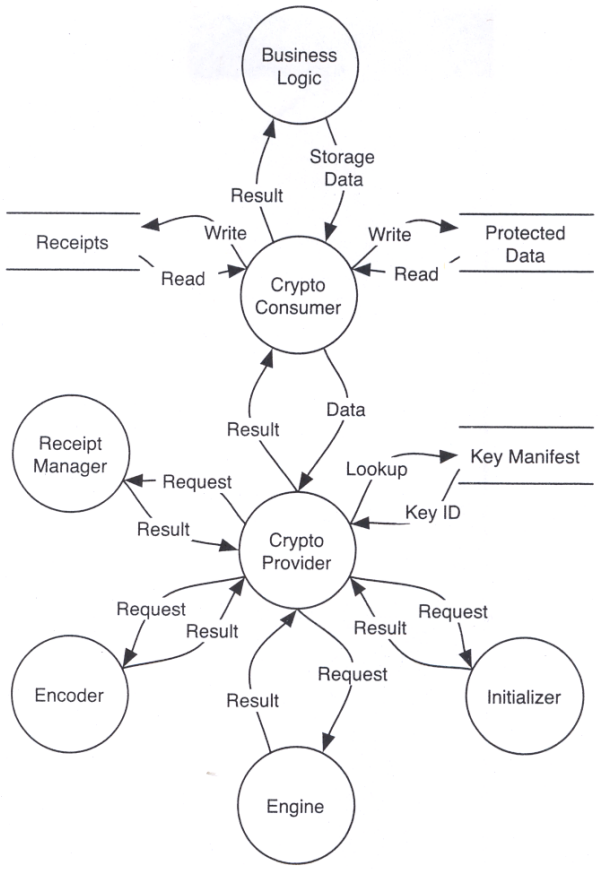

Figure 13: CryptoDB documented DFD (Level 2) [17, Fig. 6.1].

B Code Architecture

We used the Eclipse UML tool [29] to extract from the implementation various views of the code architecture. Fig. 14 shows the package structure. Fig. 15 shows the class diagram with a few selected core types.

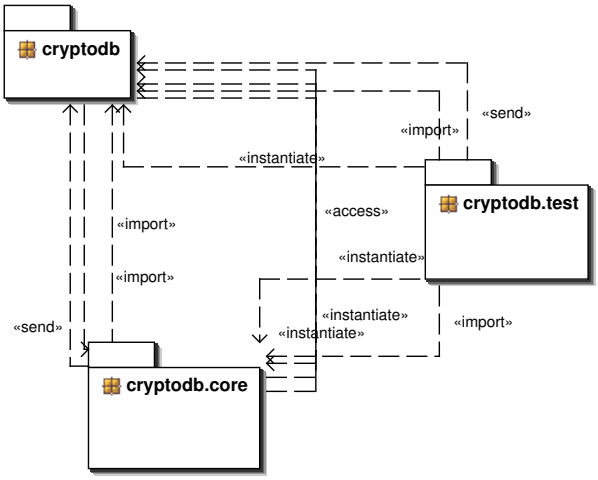

Figure 14: CryptoDB layer diagram.

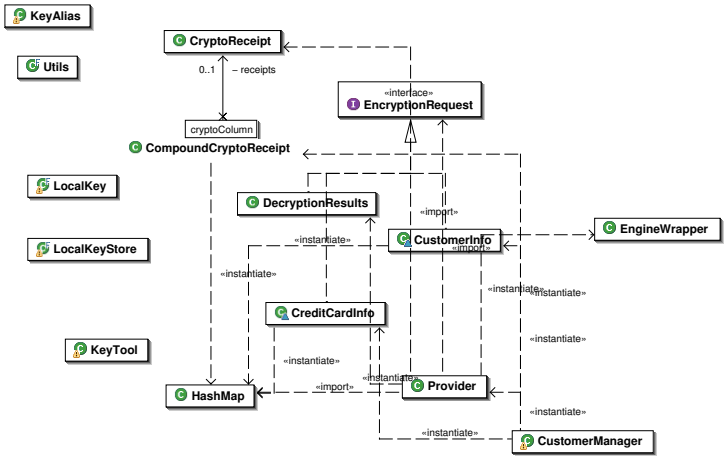

Figure 15: CryptoDB class diagram.

C Flat Object Graphs

Fig. 16 is a flat object graph obtained statically using PANGAEA [30].

Figure 16: CryptoDB flat object graph extracted using PANGAEA.

Figs. 17, 18 are flat object graphs obtained statically using WOMBLE [31].

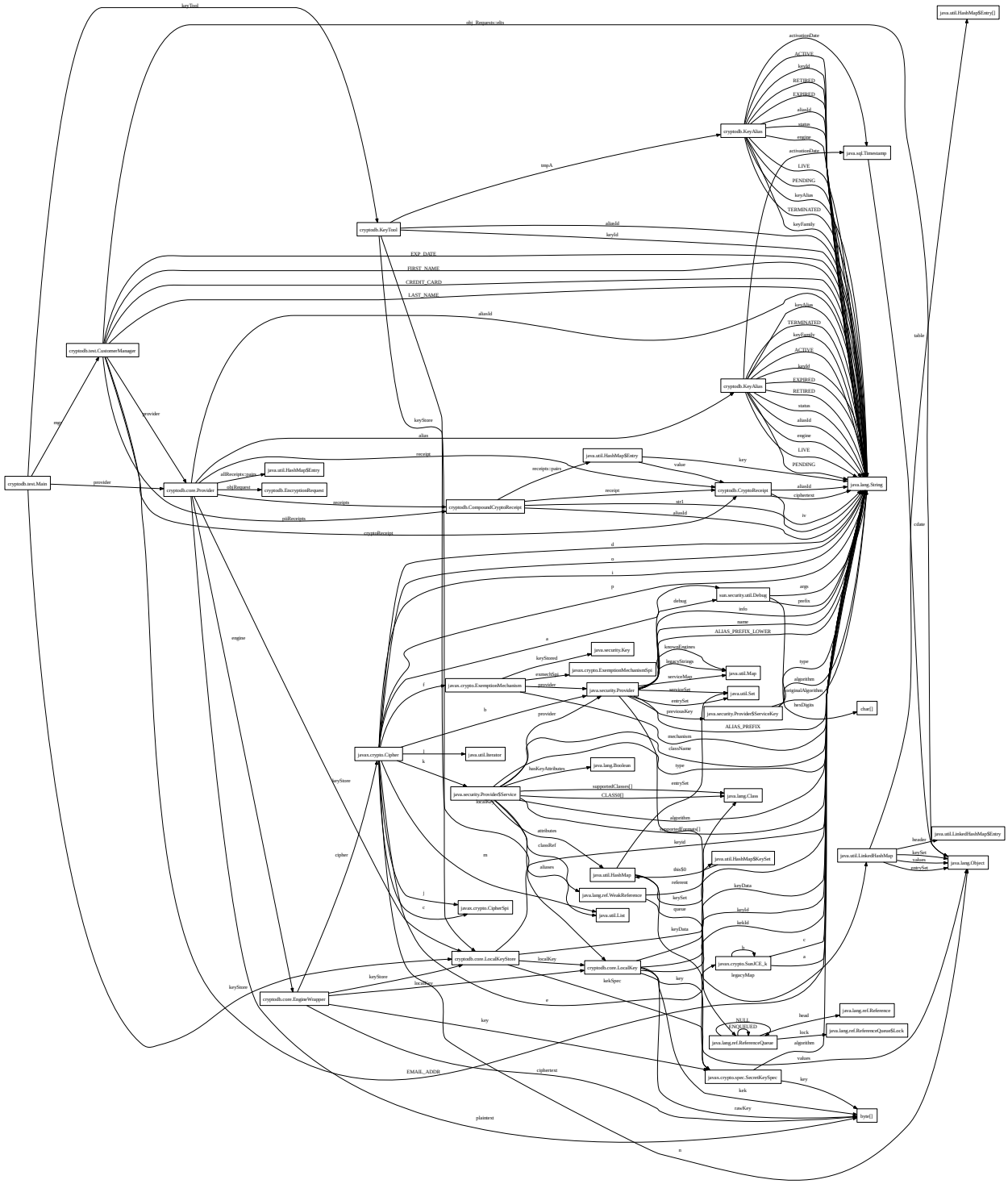

Figure 17: CryptoDB flat object graph extracted using Womble (rendered using GraphViz dot).

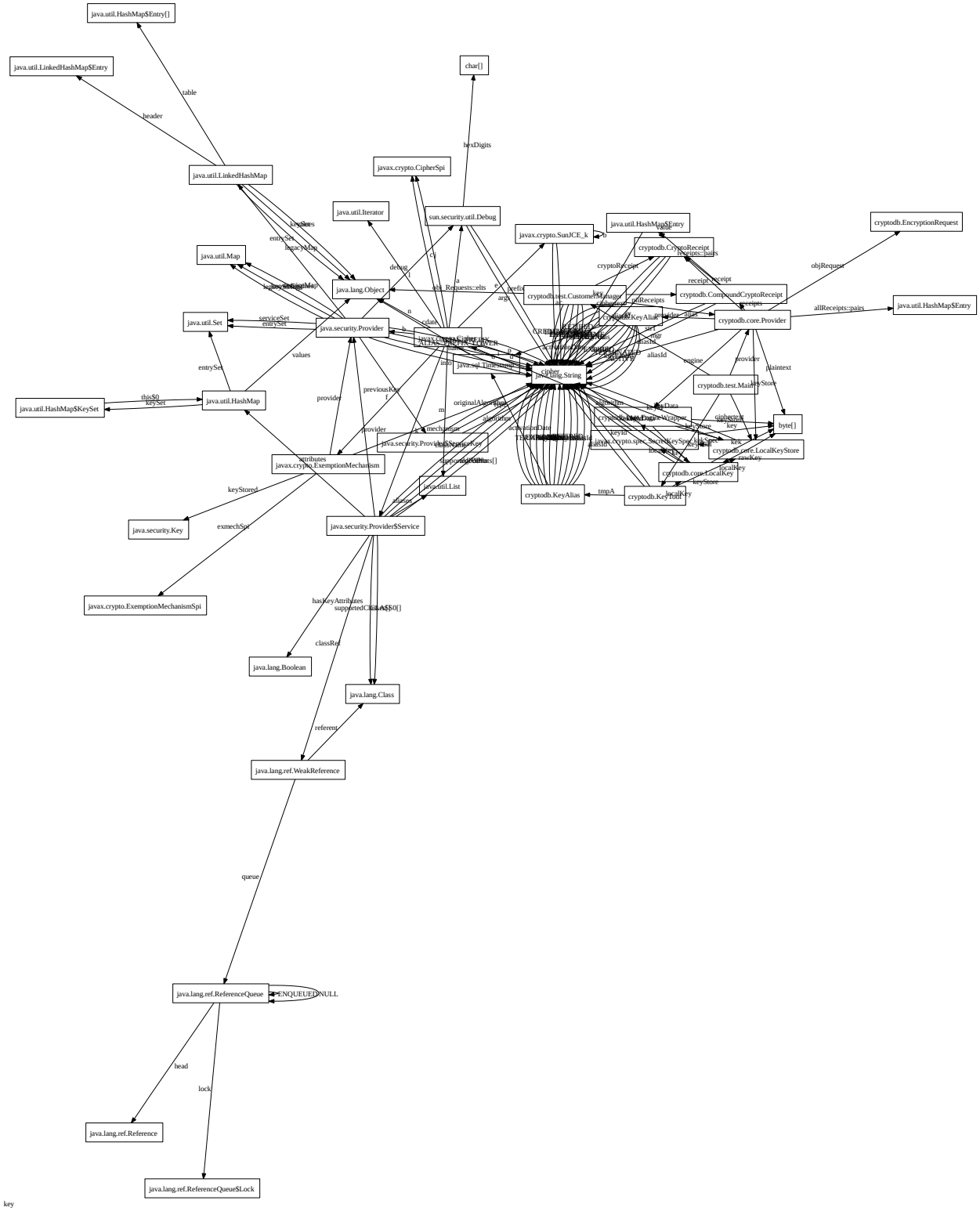

Figure 18: CryptoDB flat object graph extracted using Womble (rendered using GraphViz fdp).

D Acme Source Code for Designed Architecture

Here, we reproduce the entire architectural model, in Acme [15]. We provide both the family file, SyncFamily.acme, which defines the architectural family that supports SCHOLIA, and the target architecture itself, CryptoDBTarget.acme.

D.1 SyncFamily.acme

This file defines the architectural family SyncFamily. The properties defined here are used by SCHOLIA for conformance analysis.

```
import $AS_GLOBAL_PATH/families/TieredFam.acme;

Family SyncFamily extends TieredFam with {

    analysis isSrcComponent(d1 : SyncCompT, conn : SyncConnT) : boolean =
        connected(conn, d1) and
        exists src : SyncUserT in conn.ROLES | exists put : SyncUseT in d1.PORTS |
            declaresType(src, SyncUserT) and declaresType(put, SyncUseT)
            and attached(src, put);

    analysis isDstComponent(d2 : SyncCompT, conn : SyncConnT) : boolean =
        connected(conn, d2) and
        exists dst : SyncProviderT in conn.ROLES | exists get : SyncProvideT in d2.PORTS |
            declaresType(dst, SyncProviderT) and declaresType(get, SyncProvideT)
            and attached(dst, get);

    analysis pointsTo(d1 : SyncCompT, d2 : SyncCompT) : boolean =
        exists conn : SyncConnT in self.CONNECTORS |
              isSrcComponent(d1, conn) and isDstComponent(d2, conn);

    Role Type SyncUserT extends userT with {
        Property syncStatus : int;
    }
    Component Type SyncCompT extends TierNodeT with {
        Property syncStatus : int;
        Property label : string;
        Property hasDetail : boolean;
        Property detailStatus : int;
        Property traceability : string;
    }
    Connector Type SyncConnT extends CallReturnConnT with {
        Property syncStatus : int;
        Property label : string;
        Property traceability : string;
        Property summary : int;
    }
    Port Type SyncUseT extends useT with {
        Property syncStatus : int;
    }
    Port Type SyncProvideT extends provideT with {
        Property syncStatus : int;
```

```
    }
    Role Type SyncProviderT extends providerT with {
        Property syncStatus : int;
    }
}
```

D.2 CryptoTargetDB.acme

This file defines the target architecture itself, including the constraints we discussed in the paper.

```
import families/SyncFamily.acme;

System CryptoDBTarget : SyncFamily = new SyncFamily extended with {

    Component KeyVault : SyncCompT = new SyncCompT extended with {
        Port KeyVault : SyncProvideT = new SyncProvideT;
        Port KeyManager : SyncUseT = new SyncUseT;
        Port EngineWrapper : SyncUseT = new SyncUseT;

        Property label = "KeyVault";
    }
    Component CryptoProvider : SyncCompT = new SyncCompT extended with {
        Port KeyManifest : SyncUseT = new SyncUseT;
        Port CryptoProvider : SyncProvideT = new SyncProvideT;
        Port CustomerManager : SyncUseT = new SyncUseT;
        Port EngineWrapper : SyncUseT = new SyncUseT;

        Property label = "CryptoProvider";

        Representation CryptoProvider_Rep = {
            System CryptoProvider_Rep : SyncFamily = new SyncFamily extended with {
                Component ReceiptManager : SyncCompT = new SyncCompT extended with {
                    Port ReceiptManager : SyncProvideT = new SyncProvideT;
                    Port CryptoProvider : SyncUseT = new SyncUseT;

                    Property label = "ReceiptManager";
                }
                Component Encoder : SyncCompT = new SyncCompT extended with {
                    Port CryptoProvider : SyncUseT = new SyncUseT;
                    Port Encoder : SyncProvideT = new SyncProvideT;

                    Property label = "Encoder";
                }
            }
            Bindings {
                CustomerManager to ReceiptManager.CryptoProvider;
                EngineWrapper to Encoder.CryptoProvider;
            }
        }
    }
    Component KeyManager : SyncCompT = new SyncCompT extended with {
        Port KeyManifest : SyncUseT = new SyncUseT;
```

```
    Port KeyVault : SyncUseT = new SyncUseT;
    Port KeyManager : SyncProvideT = new SyncProvideT;

    Property label = "KeyManager";
}
Component KeyManifest : SyncCompT = new SyncCompT extended with {
    Port KeyManifest : SyncProvideT = new SyncProvideT;
    Port KeyManager : SyncUseT = new SyncUseT;
    Port CryptoProvider : SyncUseT = new SyncUseT;

    Property label = "KeyManifest";
}
Component EngineWrapper : SyncCompT = new SyncCompT extended with {
    Port EngineWrapper : SyncProvideT = new SyncProvideT;
    Port CryptoProvider : SyncUseT = new SyncUseT;
    Port KeyVault : SyncUseT = new SyncUseT;

    Property label = "EngineWrapper";

    Representation EngineWrapper_Rep = {
        System EngineWrapper_Rep : SyncFamily = new SyncFamily extended with {
            Component Engine : SyncCompT = new SyncCompT extended with {
                Port Engine : SyncProvideT = new SyncProvideT;
                Port EngineWrapper : SyncUseT = new SyncUseT;

                Property label = "Engine";
            }
        }
        Bindings {
            EngineWrapper to Engine.Engine;
            CryptoProvider to Engine.EngineWrapper;
        }
    }
}
Component CustomerManager : SyncCompT = new SyncCompT extended with {
    Port CustomerManager : SyncProvideT = new SyncProvideT;
    Port CryptoProvider : SyncUseT = new SyncUseT;
    Port CustomerInfo : SyncUseT = new SyncUseT;

    Property label = "CustomerManager";

    Representation CustomerManager_Rep = {
        System CustomerManager_Rep : SyncFamily = new SyncFamily extended with {
            Component Receipts : SyncCompT = new SyncCompT extended with {
                Port Receipts : SyncProvideT = new SyncProvideT;
                Port CustomerManager : SyncUseT = new SyncUseT;

                Property label = "Receipts";
            }
        }
        Bindings {
```

```
            CustomerManager to Receipts.Receipts;
            CryptoProvider to Receipts.CustomerManager;
        }
    }
}
Component CustomerInfo : SyncCompT = new SyncCompT extended with {
    Port CustomerManager : SyncUseT = new SyncUseT;
    Port CustomerInfo : SyncProvideT = new SyncProvideT;

    Property label = "CustomerInfo";
}
Connector CustomerInfo_CustomerManager : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector CustomerManager_CustomerInfo : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector CustomerManager_CryptoProvider : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector CryptoProvider_CustomerManager : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector EngineWrapper_CryptoProvider : SyncConnT = new SyncConnT extended with {
    Role user : SyncUserT = new SyncUserT;
    Role provider : SyncProviderT = new SyncProviderT;
}
Connector CryptoProvider_EngineWrapper : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector KeyVault_KeyManager : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector KeyManager_KeyVault : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector KeyManifest_KeyManager : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector KeyManager_KeyManifest : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
```

```
Connector KeyManifest_CryptoProvider : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector CryptoProvider_KeyManifest : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector KeyVault_EngineWrapper : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Connector EngineWrapper_KeyVault : SyncConnT = new SyncConnT extended with {
    Role provider : SyncProviderT = new SyncProviderT;
    Role user : SyncUserT = new SyncUserT;
}
Attachment CryptoProvider.CustomerManager to CryptoProvider_CustomerManager.user;
Attachment CustomerManager.CustomerManager to CryptoProvider_CustomerManager.provider;
Attachment CustomerManager.CustomerManager to CustomerInfo_CustomerManager.provider;
Attachment CustomerInfo.CustomerInfo to CustomerManager_CustomerInfo.provider;
Attachment CustomerInfo.CustomerManager to CustomerInfo_CustomerManager.user;
Attachment KeyManifest.CryptoProvider to KeyManifest_CryptoProvider.user;
Attachment KeyVault.EngineWrapper to KeyVault_EngineWrapper.user;
Attachment EngineWrapper.EngineWrapper to CryptoProvider_EngineWrapper.provider;
Attachment EngineWrapper.CryptoProvider to EngineWrapper_CryptoProvider.user;
Attachment EngineWrapper.EngineWrapper to KeyVault_EngineWrapper.provider;
Attachment EngineWrapper.KeyVault to EngineWrapper_KeyVault.user;
Attachment CryptoProvider.EngineWrapper to CryptoProvider_EngineWrapper.user;
Attachment CryptoProvider.KeyManifest to CryptoProvider_KeyManifest.user;
Attachment KeyVault.KeyManager to KeyVault_KeyManager.user;
Attachment KeyManager.KeyVault to KeyManager_KeyVault.user;
Attachment KeyManifest.KeyManager to KeyManifest_KeyManager.user;
Attachment KeyManager.KeyManifest to KeyManager_KeyManifest.user;
Attachment CryptoProvider.CryptoProvider to CustomerManager_CryptoProvider.provider;
Attachment CryptoProvider.CryptoProvider to KeyManifest_CryptoProvider.provider;
Attachment CryptoProvider.CryptoProvider to EngineWrapper_CryptoProvider.provider;
Attachment KeyManifest.KeyManifest to CryptoProvider_KeyManifest.provider;
Attachment KeyManifest.KeyManifest to KeyManager_KeyManifest.provider;
Attachment KeyManager.KeyManager to KeyManifest_KeyManager.provider;
Attachment KeyManager.KeyManager to KeyVault_KeyManager.provider;
Attachment KeyVault.KeyVault to EngineWrapper_KeyVault.provider;
Attachment CustomerManager.CryptoProvider to CustomerManager_CryptoProvider.user;
Attachment CustomerManager.CustomerInfo to CustomerManager_CustomerInfo.user;
Attachment KeyVault.KeyVault to KeyManager_KeyVault.provider;
Group KeyManagement = {
    Members {KeyManager}
}
Group CryptoConsumption = {
    Members {CustomerManager, CustomerInfo,
        CustomerManager_CustomerInfo, CustomerInfo_CustomerManager}
}
```

```
    Group CryptoProvision = {
        Members {CryptoProvider, EngineWrapper,
            CryptoProvider_EngineWrapper, EngineWrapper_CryptoProvider}
    }
    Group KeyStorage = {
        Members {KeyManifest, KeyVault}
    }
    rule noVaultToManifest = invariant !pointsTo(KeyVault, KeyManifest);
    rule keyManagementAndEngineDisconnected = invariant
         forall c : Component in KeyManagement.MEMBERS | !connected(c, EngineWrapper);
    rule limitedVaultAccess = invariant forall c : SyncCompT in self.COMPONENTS |
        pointsTo(c, KeyVault) -> c.label == "KeyManager" OR c.label == "EngineWrapper";
}
```

E Mapping between Architectural Components and Code Elements

The names of the components in the target architecture do not always match up exactly to the names of code elements in the Java implementation. For example, some of the Java class names are implementation-specific (LocalKeyStore instead of KeyVault). Table 1 provides a mapping between the components in the target architecture and the corresponding Java classes.

Architectural Component	Java Class	Note
CustomerManager	cryptodb.test.CustomerManager	AKA "crypto consumer"
CustomerManager.Receipts	cryptodb.CryptoReceipt	Receipts the consumer holds onto
CustomerInfo	cryptodb.test.CustomerInfo	AKA "protected data"
CryptoProvider	cryptodb.core.Provider	
CryptoProvider.ReceiptManager	cryptodb.CompoundCryptoReceipt	Used by the provider to produce receipts
CryptoProvider.Encoder	cryptodb.Utils	
EngineWrapper	cryptodb.core.EngineWrapper	
EngineWrapper.Engine	javax.crypto.Cipher	
KeyManifest	cryptodb.KeyAlias	The key manifest contains key aliases
KeyVault	cryptodb.core.LocalKeyStore	The key vault contains keys (LocalKeys)
KeyManager	cryptodb.KeyTool	

Table 1: Mapping between architectural components and code elements.

F Additional diagrams

F.1 Target architecture

The CryptoDB target architecture is in Fig. 19.

F.2 Built architecture

The C&C view obtained from the abstracted object graph is in Fig. 20.

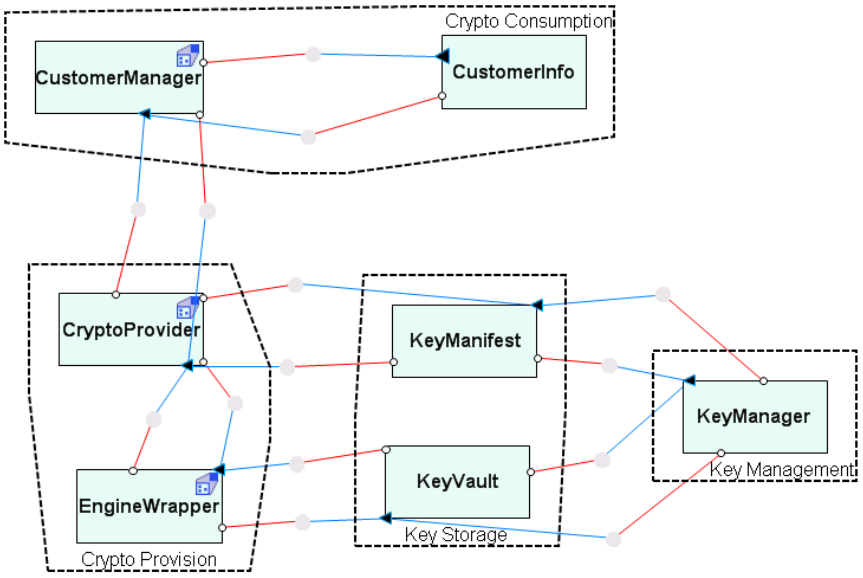

Figure 19: CryptoDB target architecture in Acme.

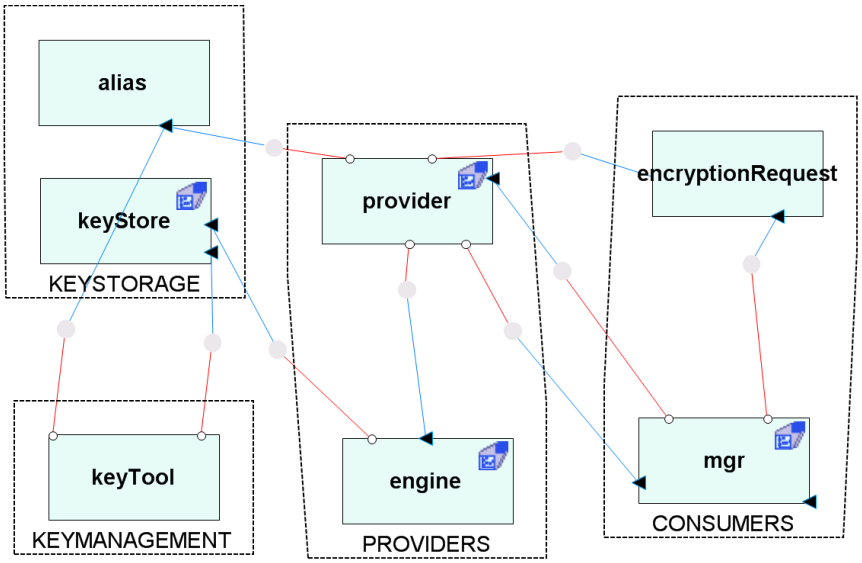

Figure 20: CryptoDB built architecture in Acme.

F.3 Extracted object graphs

An object graph without abstraction by types shows separate `CustomerInfo` and `CreditCardInfo` (Fig. 21). With abstraction by types, these two are merged, because they both implement `EncryptionRequest`.

An object graph showing explicit top-level domains for the different kinds of `String`s is in Fig. 22.

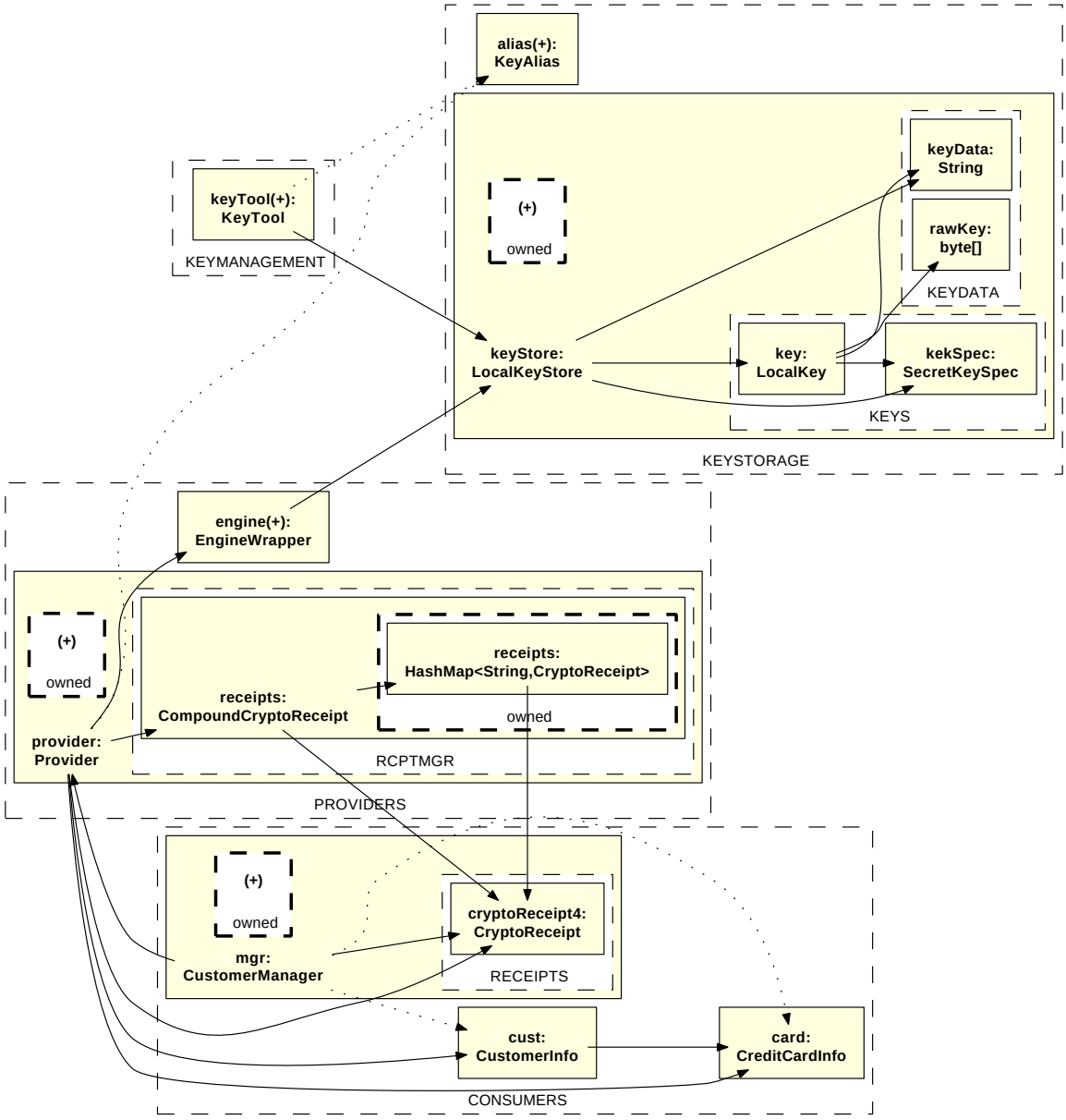

Figure 21: CryptoDB OOG, binding top-level domains for `String` to `shared`.

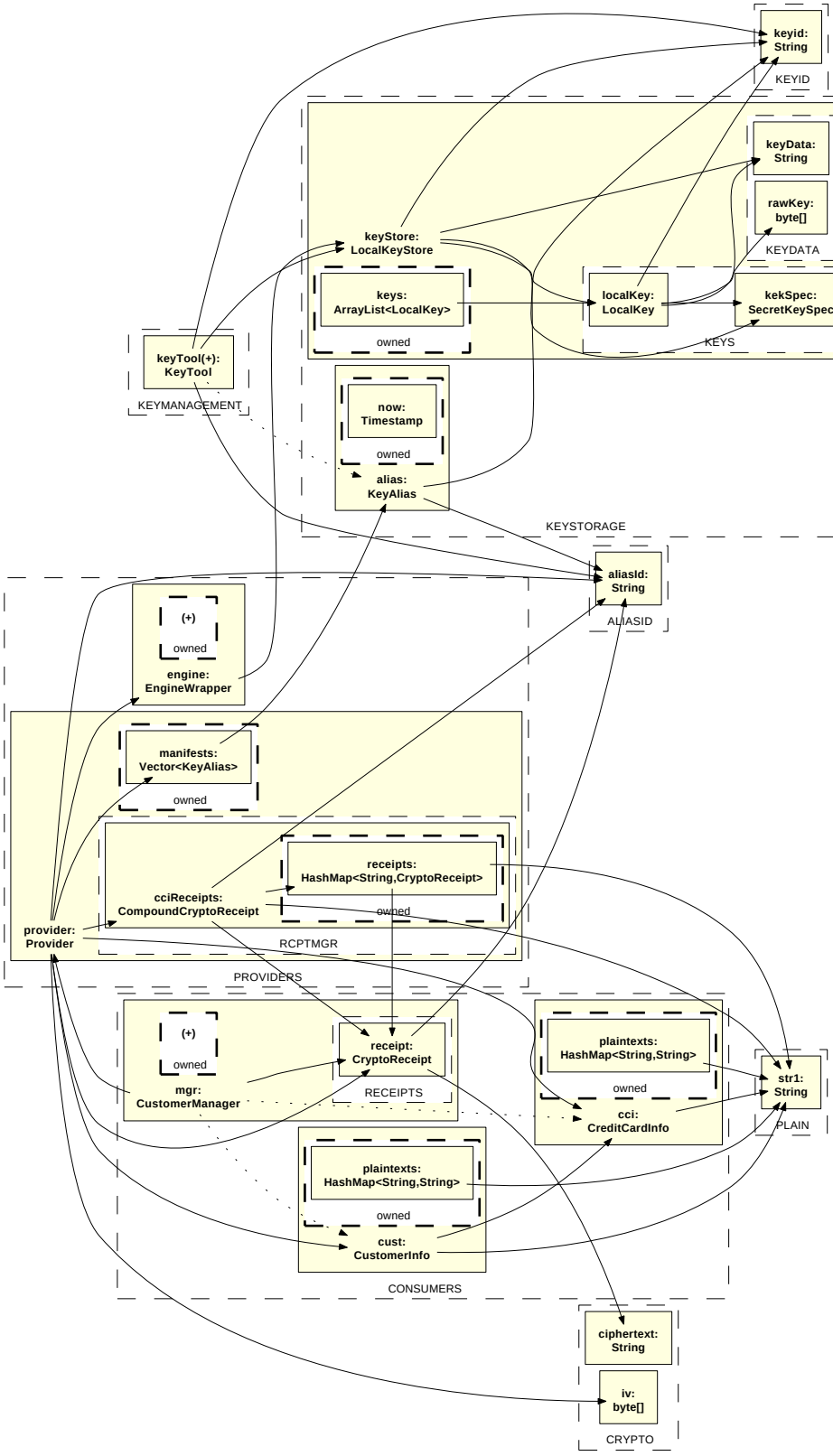

Figure 22: CryptoDB OOG with `Strings`.

References

[1] P. Torr, "Demystifying the threat-modeling process," *IEEE Secur. & Priv.*, vol. 3, no. 5, 2005.

[2] M. Abi-Antoun, D. Wang, and P. Torr, "Checking threat modeling data flow diagrams for implementation conformance and security," in *ASE*, 2007.

[3] P. Clements, F. Bachman, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2003.

[4] B. Schmerl, J. Aldrich, D. Garlan, R. Kazman, and H. Yan, "Discovering architectures from running systems," *IEEE TSE*, vol. 32, no. 7, 2006.

[5] D. C. Luckham and J. Vera, "An event-based architecture definition language," *IEEE TSE*, vol. 21, no. 9, 1995.

[6] J. Aldrich, "Using types to enforce architectural structure," in *WICSA*, 2008.

[7] M. Abi-Antoun and J. Aldrich, "Static conformance checking of runtime architectural structure," Carnegie Mellon Univ., Tech. Rep. CMU-ISRI-08-132, 2008.

[8] P. Tonella and A. Potrich, *Reverse Engineering of Object Oriented Code*. Springer, 2004.

[9] M. Abi-Antoun and J. M. Barnes. (2009) Addendum. [Online]. Available: http://www.cs.cmu.edu/~mabianto/cryptodb/

[10] M. Abi-Antoun and W. Coelho, "A case study in incremental architecture-based re-engineering of a legacy application," in *WICSA*, 2005.

[11] M. Abi-Antoun and J. Aldrich, "Ownership domains in the real world," in *IWACO*, 2007.

[12] J. Aldrich and C. Chambers, "Ownership domains: Separating aliasing policy from mechanism," in *ECOOP*, 2004.

[13] G. C. Murphy, D. Notkin, and K. J. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," *IEEE TSE*, vol. 27, no. 4, 2001.

[14] D. Kirk, M. Roper, and M. Wood, "Identifying and addressing problems in object-oriented framework reuse," *Empir. Softw. Eng.*, vol. 12, no. 3, 2006.

[15] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," in *Foundations of Component-Based Systems*. Cambridge Univ. Press, 2000.

[16] R. Monroe, "Capturing software architecture design expertise with Armani," Carnegie Mellon Univ., Tech. Rep., 2001.

[17] K. Kenan, *Cryptography in the Database*. Addison-Wesley, 2006, accompanying code at http://kevinkenan.blogs.com/downloads/cryptodb_code.zip.

[18] M. Abi-Antoun and J. Aldrich, "Static extraction of sound hierarchical runtime object graphs," in *TLDI*, 2009.

[19] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl, and D. Garlan, "Differencing and merging of architectural views," *Autom. Softw. Eng.*, vol. 15, no. 1, 2008.

[20] M. Moriconi, X. Qian, R. A. Riemenschneider, and L. Gong, "Secure software architectures," in *IEEE Secur. & Priv.*, 1997.

[21] Y. Deng, J. Wang, J. J. P. Tsai, and K. Beznosov, "An approach for modeling and analysis of security system architectures," *IEEE T. Knowl. & Data En.*, vol. 15, no. 5, 2003.

[22] J. Jürjens, *Secure Systems Development with UML*. Springer, 2004.

[23] J. Knodel and D. Popescu, "A comparison of static architecture compliance checking approaches," in *WICSA*, 2007.

[24] M. Abi-Antoun, J. Aldrich, and W. Coelho, "A case study in re-engineering to enforce architectural control flow and data sharing," *J. Syst. & Softw.*, vol. 80, no. 2, 2007.

[25] T. Lodderstedt, D. A. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," in *UML*, 2002.

[26] B. Hackett, M. Das, D. Wang, and Z. Yang, "Modular checking for buffer overflows in the large," in *ICSE*, 2006.

[27] M. Sefika, A. Sane, and R. H. Campbell, "Monitoring compliance of a software system with its high-level design models," in *ICSE*, 1996.

[28] H. J. Hoover and D. Hou, "Using SCL to specify and check design intent in source code," *IEEE TSE*, vol. 32, no. 6, 2006.

[29] Omondo, "EclipseUML," http://www.omondo.com/, 2009.

[30] A. Spiegel, "Automatic distribution of object-oriented programs," Ph.D. dissertation, FU Berlin, 2002.

[31] D. Jackson and A. Waingold, "Lightweight extraction of object models from bytecode," *IEEE TSE*, vol. 27, no. 2, 2001.