Towards Model Driven Design of Crypto Primitives and Processes

Alberto Carelli, Giorgio Di Natale, Pascal Trotta, Tiziana Margaria

Towards Model Driven Design of Crypto Primitives and Processes

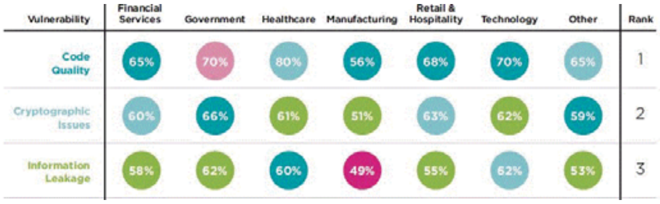
Alberto CARELLI*, Giorgio DI NATALE†, Pascal TROTTA‡, Tiziana MARGARIA‡
*CINI Cyber Security National Lab, Rome, Italy - alberto.carelli@Consorzio-CINI.it
†LIRMM, CNRS, Montpellier, France - giorgio.dinatale@lirmm.fr
‡University of Limerick Lero - The Irish Software Research Centre, Limerick, Ireland - tiziana.margaria@lero.ie

***Abstract*—To be understandable and reusable at large scale, also by non-experts in security, Crypto primitives must be implemented in a modular way, and come with well organized and well described processes to help understanding, foster adoption, and ensure a proper embedding in the applications they must protect. In this paper, we reap the benefits of the modular hardware and software architecture of the SEcube, and lift the issue of crypto-primitives management from the traditional code level to a model driven approach. On small examples, we illustrate the essential features of the approach concerning the modelling of cryptography primitives as SIBs and their organization in domain-specific SIB palettes. We also sketch how to use multifaceted taxonomies to provide compact yet expressive classifications, amounting to a semantic description of the security domain. We address in the issue of workflows by using models that ease the expression, analysis, control, and formal verification of inter- and intra-model control and data flow, though the adoption of the XMDD approach implemented in the DIME integrated modelling environment. A brief description of a home banking application sketches how in reality many of these security mechanisms need to work together in a safe and secure orchestration.**

I. Introduction

Cryptographic and security systems are the basis for guaranteeing properties like confidentiality, privacy, authentication and data integrity in several critical aspects of our society, like communications, banking, commerce, government, defence, and national security. These systems rely on the use of security primitives that allow the implementation of such properties.

Security primitives are low-level cryptographic hardware modules and algorithms used to guarantee security for computer systems. Even though security systems are widely distributed and used in all digital applications, their actual implementation remains still a challenging task, since the designer has to meet applications constrains (in terms of speed, throughput, costs) and at the same time to cope with the hectic market rules that do not give enough time for full validation and testing of such products.



Vulnerability	Financial Services	Government	Healthcare	Manufacturing	Retail & Hospitality	Technology	Other	Rank
Code Quality	65%	70%	80%	56%	68%	70%	65%	1
Cryptographic Issues	60%	66%	61%	51%	63%	62%	59%	2
Information Leakage	58%	62%	60%	49%	55%	62%	53%	3

Fig. 1. Top 3 vulnerability categories by industry vertical - from [1] p.11

Veracode an application security testing company, surveyed in 2015 the difficulties software developers declare in implementing cryptographic algorithms [1]. Analyzing the code Veracode customers submitted to its platform over an 18 month period along the OWASP[2] top10 vulnerability categories, cryptographic issues of application across all industries ranked second behind overall code quality, with information leakage as overall third. As shown in Fig. 1, these three categories were found to be the top 3 in 5 (Financial Services, Healthcare, Retail and Hospitality, Technology, and Others) of the 7 verticals surveyed, while in Government cryptographic issues ranked 5th with 51% prevalence and 4th in Manufacturing with 45% prevalence.[1]

Along this analysis, we see that the top 3 issues concern code in general, cryptography mastery, and information flows.

In our SEcube^{TM} platform, we propose to

- reap the benefits of model driven design, and lift the issue of application design and management mastery from the exclusive code level to a model driven approach, as described in [3] and applied in [4],
- provide a set of predefined, high cryptography primitives, as described exemplarily in Section II, making them available for use within the modelling environment as a domain specific palette of primitives, as described in Section III, and subsequently as a library of generated code,
- profile these functionalities according to relevant description facets and make them available in a semantically accessible model driven fashion, as described in Section IV,
- address in the issue of flows by using models that ease the expression, analysis, control, and formal verification of inter- and intra-model control and data flow, though the adoption of the XMDD approach [5] implemented in the DIME integrated modelling environment [3], [6],
- support the domain-specific and (security) aspect analysis via properties that express the guarantees needed to enforce well behaved, secure execution. These properties, as discussed in [4], can often be enforced at design time,

[1] As shown in the same report, with an overall flaw density of 352 flaws/MB and a very high or high severity density of 54 flaws/MB, Manufacturing appears to be the sector in most need of improvement, in comparison to Technology (83 resp. 29 flaws/MB) and Government (62 resp. 7 flaws/MB)

on the models, this way helping to reduce the amount of costly and slow code-level analysis or testing otherwise unavoidable.

In the following, we concentrate on a few exemplary security primitives (Section II), then sketch how a security primitives palette looks like in DIME (Section III), and in its semantic description by means of layered taxonomies (Section IV). We then describe how model driven design of Security Processes is made possible in DIME thanks to its support of variability, aspect orientation, and loose programming (Section V), and lightly analyze the interplay of different techniques for the security of a home banking example (Section VI).

II. Security Primitives

Security primitives are low-level functions, modules, and algorithms that allow adding security capabilities to digital devices, protocols, and applications. In the following we briefly describe the most important primitives: encryption algorithms (both symmetric and asymmetric), Physically Unclonable Functions, and True Random Number Generators. For a systematic introduction to applied cryptography refer to [7].

A. Encryption Algorithms

Encryption algorithms are used when is needed to preserve the confidentiality and the privacy of data when this is stored or transferred. Security is ensured altering the information message to be exchanged or saved. The information to be processed, the *plaintext*, passes through a series of mathematical operations (e.g., transposition, substitutions, etc) depending on the encryption algorithm, in order to be encrypted. The result obtained is an encoded information, the *cyphertext*, where the data is no longer clear and only authorized parties are able to read it. During both encoding and decoding processes, the cryptographic key is used as a parameter to specify the transformations of the input data. Moreover, the length of key has to be long "enough" so that exhaustive malicious attacks become unfeasible or too costly.

Depending on the key, encryption algorithms can be classified in symmetric and asymmetric. In symmetric encryption algorithms the key is common to both encryption and decryption stage. This requirements is one of the main drawbacks. Examples of symmetric encryption algorithms are: Blowfish, DES (Data Encryption Standard), 3DES (Triple DES), AES (Advanced Encryption Standard, also known as Rijndael), IDEA (International Data Encryption Algorithm), RC5. Conversely, for asymmetric cryptography (or public-key cryptography) a pair of keys is used to encrypt/decrypt data. Every user owns a pair of key, one public and the other one private. The message is encrypted with the public key, but decrypted with the private one. Public-key algorithms include RSA (Rivest - Shamir - Adleman), DSA (Digital Signature Algorithm), Diffie-Hellman, and ECC (Elliptic Curves Cryptography).

B. Physically Unclonable Functions

Physically Unclonable Functions (PUFs) exploit intrinsic manufacturing variability existing during the fabrication process of integrated circuits in order to generate a signature, unique to each single device. PUFs are a replacement of existing solutions based on Read-Only Memories (programmed at manufacturing time) or non-volatile One-Time Programmable memories. These existing solutions are shown to be vulnerable to reverse-engineering attacks and thus they cannot guarantee high security.

The produced signature must be unique from device to device, unclonable, and, for a same device, it must be robust with respect to ageing and environmental variations (reproducible). The adopted underlying mechanism is a challenge-response generator. A PUF needs an input, i.e., the challenge, to produce an output, i.e., the response. The challenge-response pairs (CRPs) set must be unique for a single device.

PUFs can be used either to generate secret keys used in encryption algorithms, or to generate a set of CRPs used to authenticate the physical device itself. As for the authentication, the PUF is first queried in order to obtain a significant subset of CRPs to save in a secure server. Once deployed, the PUF will be queried with that set of challenges. If the signatures generated by the device are equal to those stored in the server, the device is authenticated. PUFs with a significant amount of CRPs, such that is unfeasible for an attacker to exhaustively stimulate the PUF with all the allowed challenges, are classified as *strong* PUFs. On the other side, PUFs with small amount of CRPs are defined *weak*. Furthermore, some PUFs are designed to retrieve only one response, as a single signature. Typically they are adopted for key generation and storing.

One of the most investigated solutions uses SRAMs, since they provide high security (i.e., high inter-chip variation) and high stability (i.e., low intra-chip variation). Commercial devices and state-of-the-art studies exist for current SRAM CMOS technologies.

In the context of our work, we want to provide SEcube users with an easy way to access the PUF, without the need of understanding the underlying electrical and intrinsic physical mechanisms exploited for the PUF to work. A weak PUF will be seen as a constant from the programmer, whose value will be always the same for a same device, and always different for different devices. On the other side, a strong PUF will be seen as an array of weak PUF, one for each challenge. The parameters of the PUF will be the size of the generate response and the number of possible challenges.

C. True Random Number Generators

True Random Number Generators (TRNGs) are used to generate random numbers from a physical process, rather than a fixed algorithm of a predictable computer program. They are implemented by taking advantage from a physical process, like thermal noise or any other quantum phenomena and are expected to generate random bits with very high entropy and zero correlation. An on-chip TRNG design should occupy small area, give high bit rate, and have low power consumption, while assuring un-biased bit streams with high entropy per bit and low (no) correlation among them. In our work, a TRNG will be seen as a function generating a random number each time it is called. The parameters of the TRNG will be the size of the generate number and the maximum obtainable throughput.

III. Security Primitives Palette in DIME

Security primitives like those described in the previous section could be employed in different domains of interest at many levels of abstraction.

Due to the large quantity of primitives available, an efficient organization is required in order to easily locate and use them in the most appropriate way. As over time such collections grow large and more diverse, their organization must be easily understandable to different stakeholders: those how use them, manage them, and maintain them. Also here, the coarse granular model driven approach adopted in the DIME [] integrated modeling environment helps maintain the essential information well represented at the surface, while hiding the internal organization and more detailed traits of its description and implementation, that can be accessed at need on demand.

At the level of individual primitives, every security primitive becomes an atomic domain specific SIB (*Service Independent Building Block*) of the `Security` domain. The SIBs for a given domain offer what the domain experts consider to be n appropriate and useful basic service, able to satisfy a specific function.

As briefly introduced in [8] and illustrated in [3], DIME has a number of model types and views that collectively form a multifaceted yet coherent description of the system under design. The DIME Diagram Editor provides the canvas to draw the various graphical models, and additionally it provides also the palettes with basic model components that are in use for the of the model currently open and under design. These palettes can be

- **specific to the subject matter domain relevant for the specific case study**, like diabetic outpatient treatment as in [9], bioinformatics as in [10], [11], geo-information systems like in [12], but also home banking operations as in the case study of Sect.VI, or computer vision and robotics as in [13], but also
- other **cross-sectoral palettes** of functionalities needed, e.g., for security or communication, that are themselves domains, but find their use largely in combination with and embedded within (any) application that, like the above mentioned ones, would be primarily classified in another domain.

Each SIB can be dragged on to the canvas and linked with others in order to *"draw"* the workflow implementing a larger service or process. Therefore, for each SIB, the essential information at the model level concerns its correct use and embedding in (potentially any) context. At a minimum, as for APIs and Services in a service-oriented paradigm, this spans their correct embedding inside

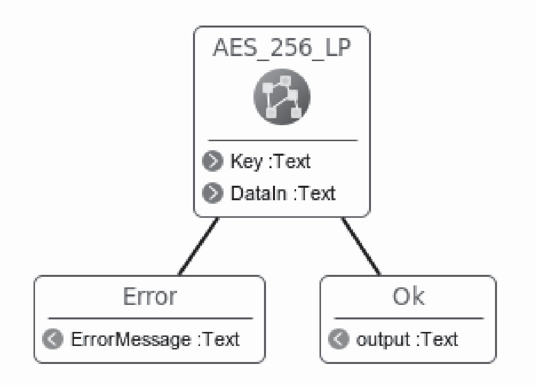


Fig. 2. SIB `AES-256` in DIME: control flow and data flow

- **Data models**, that cover the design of domain models based on common concepts like classes, attributes, and uni- or bi-directional relations between elements, and
- **Process models**, whose types in DIME span the core business logic, data retrieval (search queries) as well as dynamic access control (security guards, particularly interesting for the SEcube^{TM} platform).

Considering for example the SIB implementing the encryption function of AES shown in Figure 2, we see that

- its **Icon** denotes that it is itself a hierarchic model. Accordingly, one finds it listed in the DIME Model viewer, one could open the corresponding model and inspect the Service Logic Graph of its logical internal flow,
- its **API** expects as inputs a `Key` of type `Text` and `DataIn`, input data of type `Text` as well, and its outputs are an `Output` and an `ErrorMessage`, both of type `Text`.

In terms of a normal API description, as found in Architectural DSLs, in WSDL description of services, in the SCA SOA standard, and in the widely practiced component based design in software engineering, this would be all the information available on this component. It is sufficient from an architectural point of view to describe its execution-independent I/O potential, i.e. its static ”pluggability”, but it does not describe its behaviour, essential to use it properly.

In DIME, however, we model also the contro flow:

- the **control flow** foresees two outgoing branches: if no issue arose upon execution, the `Output` data is produced and the execution continues with the SIB that is connected to the outgoing branch labelled `Ok`. If some error occurred, an `ErrorMessage` is prepared and sent to the (exception handling) SIB connected with the `Error` branch.
- all these **labels** are viewed by the checking algorithms, that check both completeness (e.g, no dangling branches in a Service Logic Graph), (type) correctness when composing SIBs, and the correctness of the logical flow.

A large set of the properties expressed in Sect. 4 of [4] in fact

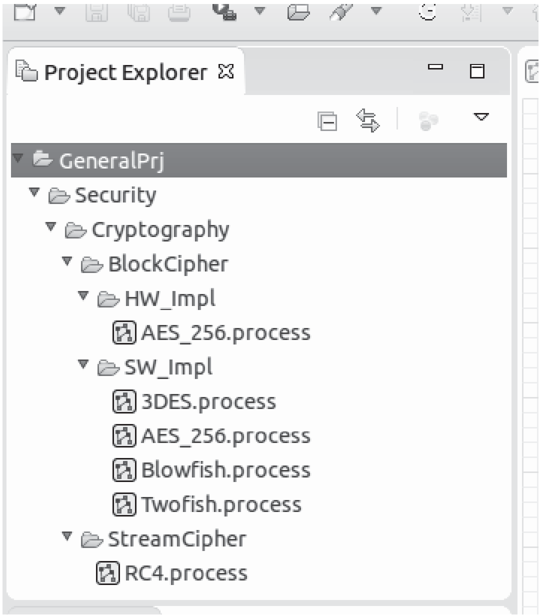


Fig. 3. Excerpt of the Security SIB Services in DIME

concerns morphologic and control flow matters, and needs the information contained in these labels.

IV. Semantic description of the Security DSL

The AES algorithm belongs to the `Cryptography` subdomain of the `Security` domain and it is a `Block Cypher`. In DIME, a user can find it under these same headers browsing the directory path as shown in Figure 3. To find it easily, one needs either a consolidated knowledge of security, or a search mechanism. However, we see that `AES_256` appears twice: once as a hardware and once as software implementation. All the security primitives seen in the previous section, and all the SEcube and the libraries relative to application domains, can be organized in taxonomies, i.e., concept based classifications with domain specific categories, whereby categories lower in the taxonomy are specializations of the higher ones (formally this is an **is_a** relation). The SIBs are leaves of such a taxonomy. The taxonomies we use are multifaceted: a tree might not be flexible enough, since some SIBs might belong simultaneously to multiple categories on different branches. In our an example, the encryption primitive for `AES_256` might be available in both hardware and software implementations, thus belonging to both `HW_Impl` and `SW_Impl` categories.

While specialists can leverage their knowledge of the domain and its terminology to find what they are looking for, casual user are easily lost. For them, not knowing the domain vocabulary, it is important to support a declarative, query-like approach. Given a taxonomy, these users can navigate through the linked concepts, and query this structure with questions like

Which AES implementations are not Hardware based?

Referring to the strongly simplified taxonomy in Figure 4, this query translates in a logical expression

$$\texttt{256} - \texttt{bit} \wedge \neg \texttt{Hardware}$$

that return the sets of SIBs that satisfy that property.

For our applications in the DIME based models, a suitably expressive classification language is achieved through taxonomies that are DAGs (directed acyclic graphs). A taxonomy is a special case of an ontology, more precisely a tree, like in the phylogenetic trees of species in biology, or if multifaceted, like in our case, a DAG. Fig.4 illustrates small fragments of the `Security` and `Implementation` facets.

More generally, a domain ontology provides a shared vocabulary, which can be used to model a particular domain, i.e., the type of objects and/or concepts that exist, and their properties and relations. The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.

The creation of ontologies is a large collective effort: ontologies are a precious good that codifies the knowledge and the language of a community of practice. The knowledge expresses there is typically layered: upper ontologies define general terms of the domain. In our simplified Fig. 4, the top level terms are those likely to belong to an upper ontology. The bulk of the general domain specific knowledge is defined in middle ontologies, in the hands of *domain experts* in the particular field of interest, who define its stable vocabulary. Although formally defined ontologies and automatic reasoning are still not widely adopted, a body of de-facto classifications, mostly in tree or matrix form has crystallized in most communities. For example, Fig. 5 reproduces an informal classification tree for System properties that distinguishes among Functional and Non-functional ones, listing some at the bottom. We see here one of the weaknesses of informally expressed domain knowledge: where exactly is each of the 4 lists connected to? some have titles (seeming like a lower ontology connection, eg, the ”-ilities” header), some not, and certainly this depiction is not machine readable, thus cannot be queried.

While upper ontologies should in the long term change rarely, middle and lower ontologies are frequently updated by concept and relation refinement. In our case, the middle ontology includes concepts like `Block Cypher` and `Stream Cypher`, that are likely ignored outside the security community. Lower ontologies are closer to the instances, that evolve rapidly and are more dynamic, following the evolution of the technology, platforms, and needs.

Verification and synthesis methods like those described in the next Section depend on the knowledge of properties of the things they deal with, thus they need to refer to machine readable descriptions of such things, codifications of the domain they belong to in terms of concepts and relations among concepts and with the things, in order to navigate and query these concept/relation networks.

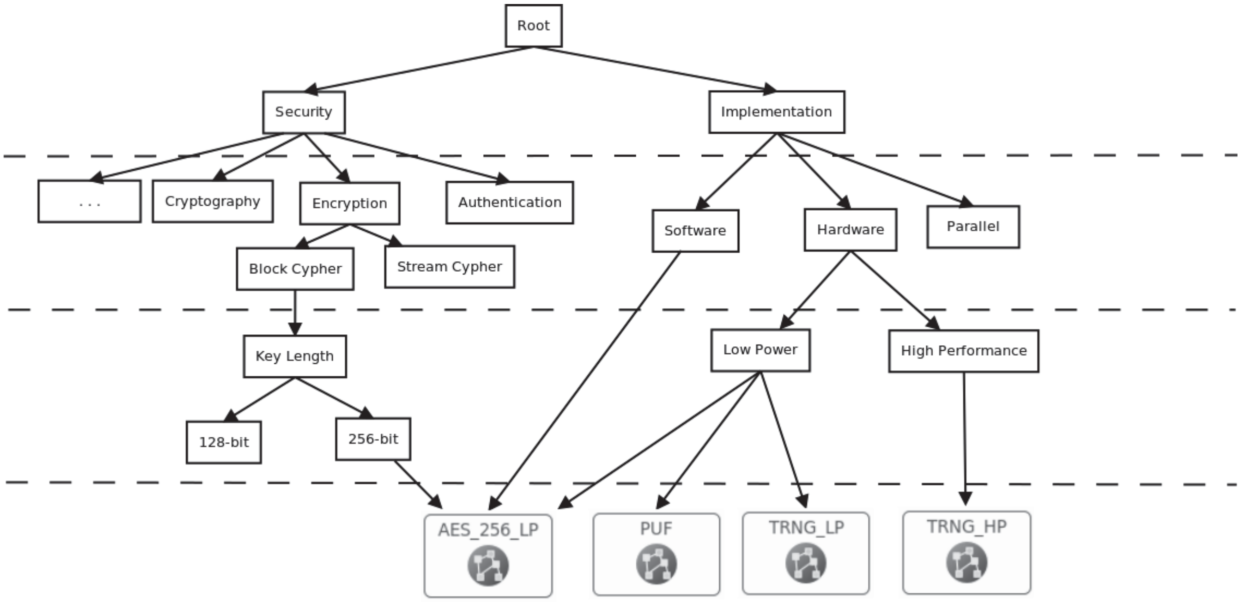


Fig. 4. Fragment of a simplified taxonomy in the security domain

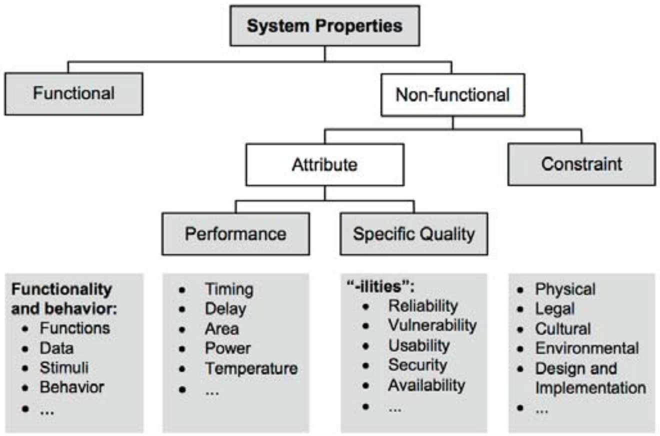


Fig. 5. Taxonomy of System Properties, from [14]

V. Towards Model Driven Design of Security Processes in DIME

As described in Section II, encryption algorithms can be classified as symmetrical and asymmetrical according to the encryption key nature. There are several algorithms created on the base of mathematical properties (e.g. big numbers multiplication, exponentiation, etc.) and entropy generation techniques (e.g. mix column, permutation tables, etc.). In spite of these significant internal differences, it is possible to define a common execution process and an abstract interface for all the algorithms in the same class.

For example, symmetrical algorithms can be managed through the same programming interface, summarized with the following functions: `Initialize`, `SetIV` (meaning Set Initial Vector), `Update`, `Finalize`.

- The `Initialize` function specifies the algorithm direction (e.g. encryption mode, decryption mode), the feedback mode (e.g. ECB, CBC, CFB, etc.) and starts the key expansion procedure.
- The `SetIV` function is only used in a feedback mode and it sets the initial vector. In principle the SetIV function may be included in the Initialize phase. Nevertheless, since the initial vector could be changed at any time and the key expansion procedure (which is usually time consuming) is not required, the SetIV functions can be isolated for a more effective usage, and the `Initialize` function becomes itself a process, that internally uses `SetIV` (cf the Usex discussion in [8]).
- The `Update` function just processes a data buffer (n blocks in case of block-cipher algorithms or n-bytes in case of stream-cipher algorithms) according to the algorithm configuration (direction, feedback mode, etc.).
- Last, the `Finalize` function frees the internal implementation structures, especially in implementations able to manage multiple concurrent sessions.

This is a case of set up and usage process that is *parameterized* in the concrete algorithm i.e. independent of which one among the set of symmetrical algorithms currently available is chosen. This is interesting, in that it allows to express in our platform the *shape of this interaction* independently of the instance of behaviour (i.e. which concrete run is executed) and of architecture (i.e. which concrete components are plugged in). In terms of the philosophy explained in [15], [16], this is a case of *horizontal and vertical looseness*.

Vertically, we employ hierarchical modeling as an aspect-oriented mechanism for specifying variability. In this case, we will use placeholder SIBs at all the variation points, that specify the characterization of the suitable instances in terms of the currently valid taxonomy concepts. This way, any SIB (elementary or not) that satisfies the properties of the variation point is an eligible instantiation, and thus a correct variant. This needs to be complemented by *constraint-guarded variability modeling*: Model checking needs to be applied in order to establish the global consistency of the product variants, which are typically built by manual specification of variations points. In this case, even if any of the symmetrical algorithms is eligible, once we have instantiated our choice concrete algorithm, it needs to remain the same along the entire protocol. This ”sameness” can be expressed as an additional constraint and checked efficiently.

Horizontally, this process is actually spread at several locations inside the larger process of the application, which is the context (or host) of the embedded security process. Therefore, its description and modelling in DIME is de facto a template of the shape, whereby most of the control flow will not be contiguous, but be interrupted by intermittent portions of the application process. For example, it is to be expected that the application will have a prologue before reaching the `Initialize` SIB, then there will be one or more occurrences of the `Update` SIB, either directly after `Initialize` or intercalated in the application workflow, and that even the `Finalize` SIB may not be the last SIB executed by the application. Here, the loose programming paradigm of [] can help. Equivalent to declarative properties, these templates can be described in terms of ”must-”successors that are in all instances the concrete next SIBs, and ”eventually” successors, that can occur along any given path after a number of other SIBs. For this kind of templates it is possible to do manual refinement followed by checks, but also to resort to constraint-driven variability modeling using (LTL) synthesis technology [17], [18], [16], to fully automatically generate workflows that satisfy all given shape constraints.

VI. Case Study: Home Banking

Nowadays, home banking is one of the most widely used web-based secure services. Several security mechanisms protect both the customer and the bank during the various processes. Although mechanisms and strategies may vary according to the banking institutions, the security primitives provided by the SEcube platform and their correspondent modelling can be combined to cover any occurring security scenario concerning the bank, the user, and the operations to be carried on with the bank accounts.

The right Bank. For example, when the customers initiate a web connection to the bank website to start a home banking session, the first security procedure is implemented through the HTTPS protocol: it aims to authenticate the bank website and create an encrypted communication channel.

This process is usually implemented by digital certificates: the bank website exposes its identity and public key through a certificate delivered and signed by a certification authority recognized by the customer web browser. This process guarantees that the user is not connected to a fake bank web site.

In terms of security primitives, the certificates management usually requires algorithms like RSA and DSA, which are fully supported and modeled in the SEcube platform.

Once the web bank authentication is successfully performed, a secure HTTPS channel is established using symmetrical encryption algorithms, such as AES256, and the user must authenticate itself.

The right User. There are many ways for the users to be authenticated. However, all the methodologies provided by the banks are based on a multi-factor authentication process, which requires more than one factor (e.g. username, password, one-time password, token authentication, etc.) to prove the user’s identity and authenticity. Again, in this case the low level security primitives provided by our platform can be combined to implement the higher level mechanisms. For example, the one-time password can be implemented as an encryption algorithm evolution started from a basic key and plaintext (also called seed) which are in common between the user token and the bank server.

Once both the parties are authenticated, there are several ways to create symmetrical session keys to protect the communication channel. In most of the cases the symmetrical keys are derived by the random challenges used in the mutual-authentication process. Sometimes the session keys are created and exchanged using specific asymmetrical algorithms like DH (Diffie Hellman). A very few times, the session keys are generated from a pre-shared master key. In any case, all the techniques described above (and many others) can be implemented by a combination of the security primitives provided by the platform.

The right Operation. After the mutual authentication is performed and the secure channel is established, the secure service is in place and the sensitive information can be encrypted and signed. In order to prevent reply attacks, the communication protocols usually provide counters which are automatically incremented at any packet transmission.

According to the banking policies, the signature process can be performed at any transmitted packet or just for specific operations, such as money transactions, authorizations, etc. In any case the algorithms used for signature belong to the asymmetrical class: RSA, DSA, Elliptic Curves, etc.

More security behind the scenes. Sometimes, within the same working session, the cryptographic keys can be renewed in a transparent way (without any action on the user side). In any case the security primitives are still the same independently of the mechanisms and strategies implemented by the specific home banking services.

Wrapping up properly. Finally, when the working session is terminated, the secure channel is closed, the session keys are deleted and all the encrypted/signed packets used in the previous session are invalidated against possible reply attacks attempts.

VII. Conclusion

In this paper we have shown how the consequently modular hardware and software architecture of the SEcube platform paired with the DIME integrated modelling environment and semantic domain description techniques can render security more easily understandable and reusable at large scale, also by non-experts in security, With an adequate formal representation of the sematic domain and its properties and relations, and with better support for property expression and enforcement, we are confident that the SEcube environment as a whole can significantly contribute to a wider adoption and higher quality implementation of security also in vertical domains where so far it is still difficult to master. Referring again to the Veracrypt report [1], their reporting that the prevalence of Cryptographic issues ranged from 80% of the uploaded applications analyzed provenient from the Healthcare sector, to 63% in Retail and Hospitality 62% in Technology, 60% in the Financial Sector, 51% in Government, down to 45% in Manufacturing, is discomforting. Healthcare and Financial Sector, as highly regulated industries (like Transportation, that was not surveyed as a category) are mandated to achieve high rates of security. One of the observations of the report concerns the programming language of the analyzed over 200.000 projects. The vast majority of the application was written in .NET or Java, and the choice maes a difference. "*Where some languages and programming models completely eliminate some security issues (for instance, buffer management issues common in C/C++ are completely eliminated in Java or .NET), often the choice of programming language is influenced by factors other than security*. This indicates that there are benefits in raising the level of abstraction ant which programmers work, providing a language discipline, infrastructure, and tools that take care in a preventive way that certain issued do not arise. Moving to models instead of code for secure application design can be the next generation of abstraction that helps to scale pervasively.

Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

We also thank Antonio Varriale (Blu5 Labs) for his valuable input to this work.

References

[1] Chris Wysopal. State of software security - focus on application development (2015). https://www.veracode.com/.

[2] The Open Web Application Security Project. https://www.owasp.org/index.php/Main_Page.

[3] Steve Boßelmann, Johannes Neubauer, Stefan Naujokat, and Bernhard Steffen. Model driven design of secure high assurance systems: an introduction to the open platform from the user perspective. In *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), July 2016, in press.

[4] Giuseppe Air Farulla, Marco Indaco, Axel Legay, and Tiziana Margaria. Model driven design of secure properties for vision-based applications: A case study. In *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), July 2016, in press.

[5] Tiziana Margaria and Bernhard Steffen. Agile IT: Thinking in User-Centric Models. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 490–502. Springer Berlin / Heidelberg, 2009.

[6] Stefan Naujokat, Michael Lybecait, Dawid Kopetzki, and Bernhard Steffen. Cinco: A simplicity-driven approach to full generation of domain-specific graphical modeling tools. *Int. Journal on Software Tools for Technology Transfer (STTT), Springer Verlag, (to appear)*, 2016.

[7] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.

[8] Antonio Varriale, Giorgio Di Natale, Paolo Prinetto, Bernhard Steffen, and Tiziana Margaria. Secube^{TM}: An open security platform: General approach and strategies. In *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), July 2016, in press.

[9] Tiziana Margaria, Steve Boßelmann, and Bertold Kujath. Simple modeling of executable role-based workflows: An application in the healthcare domain. *J. Integrated Design & Process Science*, 17(3):25–45, 2013.

[10] Anna-Lena Lamprecht and Tiziana Margaria. Scientific Workflows and XMDD. In *Process Design for Natural Scientists: An Agile Model-Driven Approach*, volume 500 of *CCIS*. Springer Berlin Heidelberg, 2014.

[11] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. Seven variations of an alignment workflow - an illustration of agile process design and management in bio-jeti. In *Bioinformatics Research and Applications, Fourth International Symposium, ISBRA 2008, Atlanta, GA, USA, May 6-9, 2008. Proceedings*, pages 445–456, 2008.

[12] Samih Al-Areqi, Steffen Kriewald, Anna-Lena Lamprecht, Dominik Reusser, Markus Wrobel, and Tiziana Margaria. Towards a flexible assessment of climate impacts: The example of agile workflows for the ci: grasp platform. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II*, pages 420–435, 2014.

[13] Giorgio Di Natale, Alberto Carelli, Pascal Trotta, and Tiziana Margaria. Model driven design of crypto primitives and processes. In *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), July 2016, in press.

[14] Nadereh Hatami Mazinani. Multi-level analysis of non-functional properties. PhD Thesis, Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart, 2014.

[15] Anna-Lena Lamprecht, Stefan Naujokat, and Ina Schaefer. Variability management beyond feature models. *IEEE Computer*, 46(11):48–54, 2013.

[16] Sven Jörges, Anna-Lena Lamprecht, Tiziana Margaria, Ina Schaefer, and Bernhard Steffen. A constraint-based variability modeling framework. *International Journal on Software Tools for Technology Transfer*, 14(5):511–530, 2012.

[17] Bernhard Steffen, Tiziana Margaria, and Burkhard Freitag. Module configuration by minimal model construction. In *Technical Report - MIP Universitaet Passau, Faku—taet fuer Mathematik und Informatik*. Citeseer, 1993.

[18] Stefan Naujokat, Anna-Lena Lamprecht, and Bernhard Steffen. Loose programming with PROPHETS. In *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 94–98, 2012.