Security-aware Modeling and Analysis for HW/SW Partitioning

Letitia W. Li[1,2], Florian Lugou[1] and Ludovic Apvrille[1]
[1]*Télécom ParisTech, Université Paris-Saclay, 450 route des Chappes, Biot, France*
[2]*Institut VEDECOM, 77 rue des Chantiers, Versailles, France*

Keywords: Embedded Systems, Partitioning, ProVerif, Formal Verification.

Abstract: The rising wave of attacks on communicating embedded systems has exposed their users to risks of information theft, monetary damage, and personal injury. Through improved modeling and analysis of security, we propose that these flaws could be mitigated. Since HW/SW partitioning, one of the first phases, impacts future integration of security into the system, this phase would benefit from supporting modeling security abstractions and security properties, providing designers with useful partitioning feedback obtained from a security formal analyzer.
In this paper, we present how our toolkit supports security modeling, automated security integration, and formal analysis during the HW/SW partitioning phase for secure communications in embedded systems. We introduce "Cryptographic Configurations", an abstract representation of security that allows us to verify security formally. Our toolkit further assists designers by automatically adding these security representations based on a mapping and security requirements.

1 INTRODUCTION

Many recent attacks targeted security vulnerabilities on embedded devices such as the Tesla hack (Constantin, 2016), mobile&smart phones (Maslennikov, 2010), avionics (Costin and Francillon, 2012), drones (Rodday, 2016), and smart objects such as the Fitbit (Apvrille, 2015). These security vulnerabilities leave users susceptible to personal injury, theft of personal information, or financial damage.

Finding security flaws in these systems is difficult, as their complexity is further increased by their heterogeneous nature; one must analyze both hardware and software components, and the consequent interactions. Moreover, correcting these security flaws might be difficult once the system has been released - and sometimes impossible - if the flaws cannot be corrected by software update only. As commonly stated, these kinds of issues should be discovered early in the development process.

The SysML-Sec methodology was introduced to handle the design of such complex systems, in terms of safety, performance, and security (Apvrille and Roudier, 2015). SysML-Sec extends UML for the design of embedded systems. One of its important early development phases is the hardware / software partitioning phase, which distributes functions to be realized by the system among candidate hardware architectures. This phase decides on the "best" architecture according to several criteria such as cost of the platform and its performance. To better address embedded system security, the choice of architecture should also be based on the ability of the architecture to support security features for secure communication.

Security properties of the components of the architecture itself should be modeled based on attacker capabilities. For example, communication buses and memories can be internal and secure against an external attacker, and functions mapped to the same processor using an on-die RAM can consider their message exchange secure. On the contrary, external buses can be spied on. The chosen architecture also affects where encryption algorithms need to be executed and where cryptographic materials need to be stored to fulfill security requirements.

Figure 1 shows the possible approaches to modeling security during partitioning, in terms of secure data exchange. Using the current HW/SW partitioning methodology of the Y-chart approach (Method A) (Kienhuis et al., 2002), manually modeling security only in terms of overhead or calculation complexity is insufficient due to the lack of support for security assessment. Method B involves manually modeling security in the application, and then performing map-

Li L., Lugou F. and Apvrille L.
Security-aware Modeling and Analysis for HW/SW Partitioning.
DOI: 10.5220/0006119603020311
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 302-311
ISBN: 978-989-758-210-3

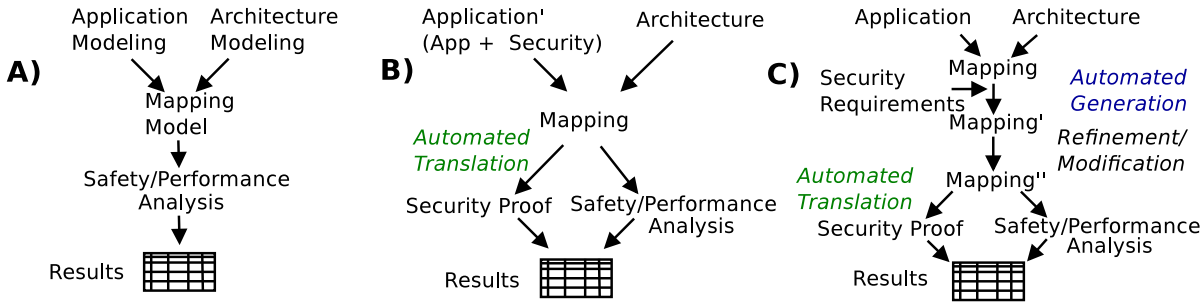

Figure 1: Methods for Security in Partitioning: Method A shows the Y-chart approach. Method B shows security-aware modeling with security analysis. Method C shows automated security generation.

ping and analysis. Method C improves upon B by automatically generating mappings with added security mechanisms based on provided data security requirements. Afterwards, a designer may refine or modify the automatically generated mapping (for example, with additional or altered security mechanisms), and then perform verification and analysis to ensure the final mapping fulfills safety, performance and security requirements.

This paper presents abstract security modeling of data exchange during the HW/SW partitioning process, and how our toolkit TTool (Apvrille, 2003) extends support to Methods B and C. In particular, our contribution involves creation of *Cryptographic Configurations* for abstract modeling of security mechanisms and estimating performance costs for added security based on security requirements and a mapping. Also, the paper shows how to formally evaluate the placement and efficiency of security mechanisms with regards to security properties at stake in the partitioning phase. Future works will elaborate on other types of security.

Section 2 presents hardware/software partitioning in our toolkit. Section 3 presents our case study on which we demonstrate our methodology. Section 4 presents our approach to integrating security design elements into a partitioning environment. Next, we present the related work in Section 5. Finally, Section 6 proposes future work and concludes the paper.

2 CONTEXT

2.1 Partitioning

The HW/SW partitioning phase requires a description of the application functionalities and their communications, the selection of a system's hardware architecture and the mapping of functional tasks - and their communications - to different hardware and software components. Partitioning is commonly performed at a high-level of abstraction (also known as System-Level Design Space Exploration) where application functionalities are described mostly in terms of communications and execution times. Design Space Exploration intends to select an architecture according to criteria such as silicon area, power consumption, satisfaction of safety properties and performance. The performance includes the latencies between an input and an ouput, the schedulability of the tasks on each processor, the schedulability of bus transfers, and finally the peak and average load on processors and buses.

2.2 Modeling Partitioning in SysML-Sec

A SysML-Sec partitioning consists of three modeling stages: application modeling, architectural modeling, and mapping modeling.

The application model comprises of a set of communicating tasks built upon block diagrams. The behavior of tasks is described abstractly with activity diagrams. Functional abstraction allows us to ignore the exact calculations and data processing of algorithms, and consider only their relative execution time. Data abstraction allows us to consider only the size of data sent or received, and ignore details such as type, values, or names.

The architectural model displays the underlying architecture as a network of abstract execution nodes, communication nodes, and storage nodes. Execution nodes consist of CPUs and Hardware Accelerators. All execution nodes can be parametrized with data size, instruction execution time, and clock ratio. CPUs can further be customized with scheduling policy, task switching time, cache-miss percentage, etc. Communication nodes include bridges and buses. Buses connect execution and storage nodes for task communication and data storage or exchange, and bridges connect buses. Buses are characterized by their arbitration policy, data size, clock ratio, etc., and

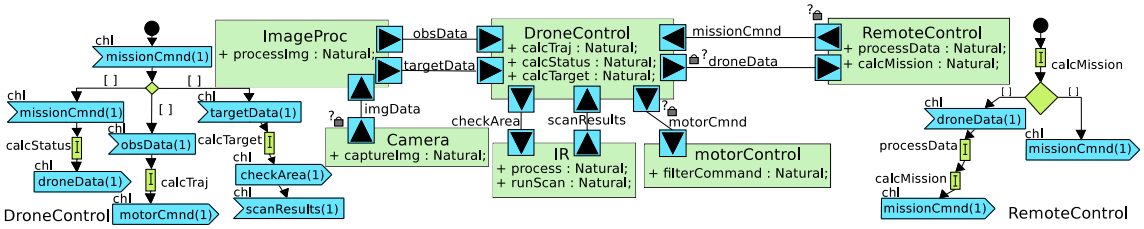

Figure 2: Drone Application Design.

bridges are characterized by data size and clock ratio. Storage nodes are memory units, which are defined by data size and clock ratio.

Mapping partitions the application into software and hardware as well as specifying the location of their implementation on the architectural model. A task mapped onto a processor will be implemented in software, and a task mapped onto a hardware accelerator will be implemented in hardware. The exact physical path of a data/event write may also be specified by mapping channels to buses, bridges and memories.

2.3 Formal Verification During Partitioning

Once mapping is complete, formal verification evalutates safety properties (reachability, liveness, deadlock), the scheduling of tasks and transactions on buses, and the worst case execution time and CPU/bus loads. Simulation is used for step by step execution of the application, and for evaluation of precise timing for a given execution trace.

Before the present contribution, security could not be explicitly modeled in the mapping stage. The additional computation power induced by the introduction of security mechanism could be evaluated by indicating the execution complexity of a security mechanism, but its impact on security could not be modeled or verified. For example, in (Schweppe et al., 2011), it was shown how to evaluate the impact of security computation on an automotive system. The present contribution intends to perform architecture exploration that takes into account both the fulfillment of security requirements and their impact on the performance of the system. This extension should help in finding an architecture that can enforce the security requirements while still respecting its safety and performance properties.

3 CASE STUDY

Autonomous Drones offer many avenues of support for future diaster relief efforts. Their manuverabilty allows them to access remote locations even with damaged roads, or access damaged structures for reconnaisance instead of risking relief workers. Combined with detection systems, these drones may process captured images and broadcast conditions of discovered victims to relief headquarters.

However, use of drones during these precarious situations induces problems of their own. Previous work (Rodday, 2016) has demonstrated how an insecure remote connection to the drone allows an attacker to hijack control. Captured images of victims are sensitive and, for the sake of privacy, they should not be allowed to be intercepted and posted online or published (Tanzi et al., 2015). Also, drones capable of generating a 3D mapping of a building carry a detailed architectural plan of an area, which could be valuable to criminals targeting this location. Even if an attacker physically steals a drone, the on-board sensitive information should remain confidential.

In this case study, we will consider a drone which communicates with a Remote Control Center for mission directives using Wifi to navigate to a target area and search for victims. The drone itself consists of a Main Controller, Image Processing Unit, Camera, Infrared Sensor, and Motor Controller. Activity diagrams of the Main Controller and Remote Control Center are shown in Figure 2. The Main Controller calculates a trajectory based on mission commands and image data, and uses IR data to check for victims. The drone constantly updates its status to Remote Control Center. Channels through which sensitive data are sent and that we wish to secure are marked with a grey lock and question mark. Remote Control Center is mapped to an external computer, and the drone itself must contain the physical

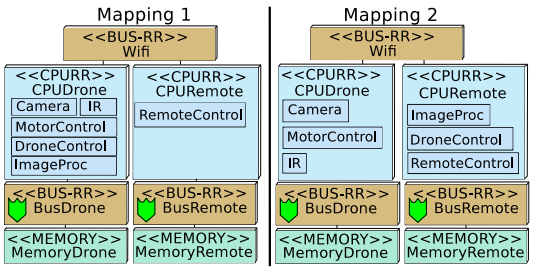

Figure 3: Drone Sample Architecture and mapping.

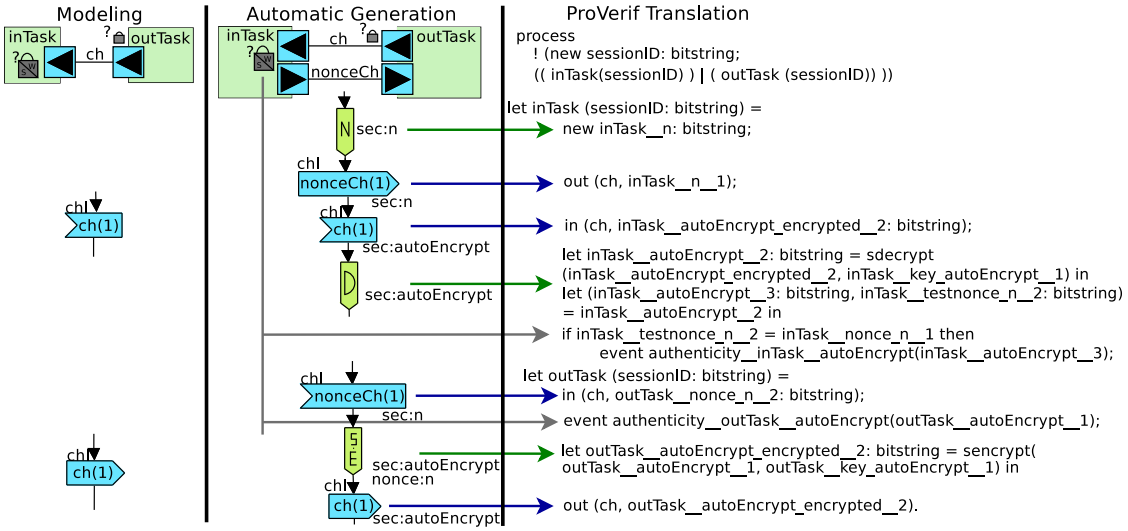

Figure 4: Overview of Modeling and Analysis.

sensors and drivers: Camera, IR, and Motor Controller. Also, the drone and remote computer must be modeled as communicating over an insecure bus, but the remainder of the architecture and mapping is yet to be determined. For example, moving task Image Processing Unit to the remote computer decreases onboard processing and saves battery power, but broadcast of unencrypted image data is undesired. While our toolkit can perform rapid exploration of multiple architectures, in the interest of space we present two mappings: one with all processing on-board the drone, and another leaving only the minimal tasks on the drone. Figure 3 shows the two proposed mappings. We model that the remote CPU offers more processing power, so it executes tasks faster. In the following section, we present how our toolkit automatically generates a secure model for each mapping and subsequent analysis.

4 SECURE PARTITIONING

In this section, we present our contribution: extensions to the HW/SW partitioning phase with secure data exchange concepts to support Methods B and C of Figure 1. Adding security semantics — and even more, proving security properties — during the partitioning phase is a difficult challenge. On the one hand, the ideal security analysis would take into account every single detail of the system. On the other hand, the designer needs to quickly describe his/her system for efficient design space exploration, which requires providing an abstract description of the system. We provide a security semantic that does not require the designer to deal with implementation details that are irrelevant to partitioning. For example, we are not interested during partitioning in the implementation of encryption algorithms; we only need to consider parameters that will affect the partitioning choice (satisfaction of security properties, execution time, data size). Our solution uses *Cryptographic Configurations*, a tag indicating various security functions and their overhead.

While the implementation details ultimately determine the security of a system, taking security into account in early development phases, without constructing the entire software/system design, would prevent costly late-stage reworking of designs that present vulnerabilities due to early partitioning choices. Even if actual security algorithms will be more complex, basic security primitives and relative computation complexities should be accurate and provide useful feedback to designers when selecting a mapping.

An overview of the aspects of security modeling and analysis is shown in Figure 4. Starting with a base partitioning model, our toolkit automatically generates the secured models (which the designer can modify), and then finally translates a final model into a specification handled by a formal verifier — ProVerif (Blanchet, 2001).

4.1 Modeling of Security Mechanisms

We introduce *Cryptographic Configurations* to allow the security verifier to track data encryption elements, used in Methods B and C of Figure 1. Within activity diagrams, they appear as an upside-down pentagon marked with their type, as shown in Figure 4. Cryptographic Configurations can be typed as follows. *Sym-*

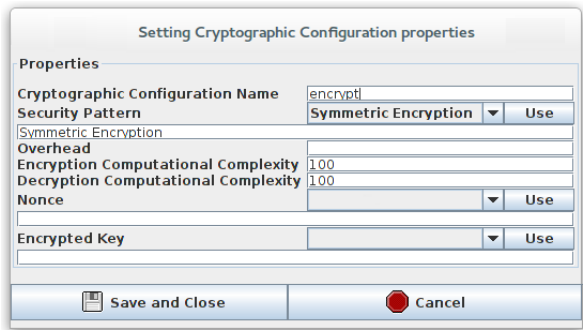

Figure 5: Window for specification of Cryptographic Configuration.

metric Encryption, *Asymmetric Encryption*, and *MAC* patterns encrypt data along with a key/keys specific to the pattern. *Hash* calculates a hash of the data. *Nonces* can be concatenated to messages before encryption to verify authenticity. *Advanced* allows the user to indicate their own encryption scheme, such as combinations of Cryptographic operations.

Figure 5 shows the specification of a Cryptographic Configuration. The designer can choose the type, and then the corresponding performance properties. For example, encryption operations may be characterized by an Encryption and Decryption Computational Complexity (a measure of the relative execution cycles depending on the processor), and Overhead (additional bits added to the message due to encryption). Within the encryption configurations, a specified nonce may also be concatenated onto messages to prevent replay attacks. Cryptographic material such as nonces are characterized by a size in number of bits. These parameters allow us to model the impact of security mechanisms on performance when evaluating candidate mapping.

Cryptographic Configurations can also secure other cryptographic material such as keys. For example, we can model key distribution, where one task forges a key for Symmetric Encryption, and sends it encrypted with another task's public key. The receiving task can then decrypt the message with its own private key to recover the shared key.

The security of the hardware architecture itself is modeled by specifying if a bus is insecure or secure (marked with a green shield) depending on the assumed attacker capabilities. For example, on Figure 3, the Wifi connection is modeled insecure and all data broadcast is available to an attacker, where both internal buses on the Remote Computer and Drone are modeled as secure. In our case study, we first limit our protection to data broadcast wirelessly and to external attackers.

To send encrypted data along a channel, the designer must create a Cryptographic Configuration, tag all channels along which encrypted data is sent, and finally recover the original data with the Decryption operator. All the keys must be mapped to memories before security verification, and the toolkit warns a designer if access to a key implies transmission across a non-private channel.

4.2 Modeling of Security Properties

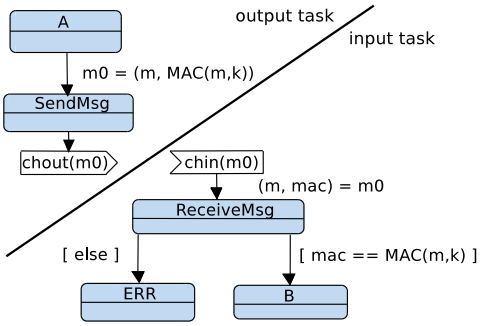

Figure 6: Translation result of a *MAC* cryptographic configuration.

The purpose of our work is to provide feedback to the designer about various security-related properties of the partitioning model. These properties are defined to match their counterparts in the prover (ProVerif) specification. A more formal definition of these properties can thus be found in the ProVerif user manual or other publications (Blanchet, 2001) for instance.

4.2.1 Reachability

An element of a state machine of a task is said to be reachable if there exists an execution trace of the model in which this element is reached.

4.2.2 Confidentiality

We say that an exchange on a channel, identified by its cryptographic configuration, is **not** confidential if the attacker is able to get the content of a message exchanged using the cryptographic configuration by listening and sending messages, and performing computations.

4.2.3 Integrity

Integrity – also called *weak authenticity* – is a property of an exchange, characterized by an *input* operation and its associated *output* operation. This property implies that each message received at the *input* operation was necessarily sent in the exact same form by the associated *output* operation.

For instance, an exchange tagged with a *MAC* cryptographic configuration would be used to model the fact that the message is concatenated with its MAC. An implicit verification is also considered before fully admitting the message on the other end of the channel. If the key used for computing this MAC is only used at the *output* and *input* operations, integrity will be proved true since each time a message m is accepted at the *input* operation (state B in Figure 6), this means that the message was received with $MAC(m)$, which can only be computed if m was sent at the corresponding *output* operation (state A in Figure 6).

Note that encryption and decryption are modeled with ProVerif by using a linked constructor and destructor function. This means that the decryption of a received message only completes if this message was previously forged by calling the encryption function with the same key that is used to decrypt it. Thus, if a symmetric key is shared only by the sender and the receiver, integrity will be maintained for any message encrypted by the sender using this key.

4.2.4 Strong Authenticity

Like integrity, strong authenticity is a property attached to an exchange. Strong authenticity implies weak authenticity, but also requires that, given multiple executions of the system that would happen in parallel (we call each of these executions a *session*), a message received at this *input* operation is sent by the associated *output* operation of the **same** session.

If a *MAC* cryptographic configuration is used as earlier and a message m is received at the *input* operation, then the message was necessarily sent by the associated *output* operation. However, this *output* operation could generate the message m in one session, and m could be received in two different sessions. *MAC* is indeed not sufficient to guarantee strong authenticity. This failure to provide strong authenticity makes the system vulnerable to replay attacks. Thus, strong authenticity can be ensured combining MAC and nonces cryptographic configurations.

4.3 Automated Security Generation

While Cryptographic Configurations can be manually handled by the designer, it is also possible to automatically generate these security elements, as shown in Method C of Figure 1. Based on security requirements provided by the designer, our toolkit automatically generates Cryptographic Configurations for each channel whose data must be secure. Currently, we support automatic preservation of confidentiality (protecting against leakage of sensitive data) and strong authenticity (protecting against tampering and replay). When both need to be guaranteed, Cryptographic Configurations are generated so that sensitive data will be concatenated with a nonce and then encrypted before being transmitted across an insecure channel. This automated encryption adds a basic estimation of security, which the designer can later modify. This transformation is shown in Figure 4 and described in Algorithm 1.

In our illustrated example, we present two tasks, where *outTask* sends data along a channel *ch* to *inTask*. We assume that they have pre-shared keys. To ensure strong authenticity of the exchange, first *inTask* forges a nonce *n* and sends it to *outTask*. A new channel *nonceChannel* is generated for this purpose. Upon receiving the nonce, *outTask* concatenates it to the data to be sent, and then in this case uses symmetric encryption to encrypt the combined message. The encrypted message is sent to *inTask*, which then decrypts it. The decrypted message is split to separate out the nonce, and if the received nonce matches the sent one, then *inTask* accepts the sent data.

```
foreach channel ch do
    foreach bus in path do
        if bus = public then
            ch.security ← insecure;
        end
    end
    if ch.security = insecure then
        inTask.addChannel(nonceChannel);
        outTask.addChannel(nonceChannel);
        New: n [CryptoConf]
        ch.inTask.forgeNonce(n);
        ch.inTask.send(n);
        New: autoEncrypt [CryptoConf]
        autoEncrypt ← ch.outTask.encrypt(ch.data |
        n);
        ch.outTask.send(autoEncrypt);
        recData ← ch.inTask.receive(autoEncrypt);
        decData | n2 ← ch.inTask.decrypt(recData);
        if n = n2 then
            symData ← decData;
        end
    end
end
```

Algorithm 1: Pseudocode of secure model generation. Note that "|" means concatenate.

4.4 Formal Verification of Security Properties

Security analysis is performed with ProVerif, a verification tool operating on pi-calculus specifications (Blanchet, 2001). A ProVerif specification consists of a set of processes communicating on public and private channels. Processes can split to cre-

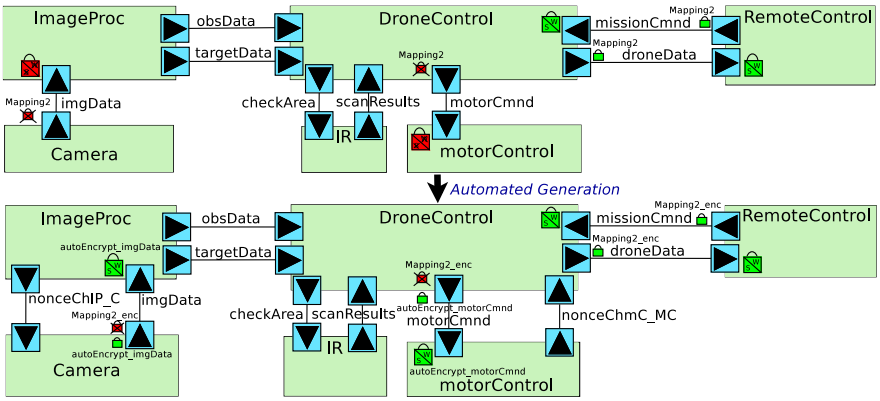

Figure 7: Security Verification results for Mapping 2 and auto-generated secured Mapping 2.

ate concurrently executing processes, and replicate to model multiple executions (sessions) of a given protocol. Cryptographic primitives such as symmetric and asymmetric encryption or hash can be modeled through constructor and destructor functions. ProVerif assumes a Dolev-Yao attacker, which is a threat model in which anyone can read or write on any public channel, create new messages or apply known primitives. Other works used ProVerif to analyze security in real-world systems such as EVITA (Pedroza et al., 2011) and Direct Anonymouse Attestation (Smyth et al., 2015).

Once a complete application modeling has been mapped to an architecture, a ProVerif specification can be generated from the partitioning model as shown in Figure 4. Infinite parallel processes are generated for each task, and each task sends and receives data or performs cryptographic operations as shown. In (Lugou et al., 2016), we presented a translation from a software model consisting of states, actions, and cryptographic operations. We leverage this translation by first converting our partitioning model into an equivalent software model representation, and use the existing translation process to generate a ProVerif model.

The main ideas of the translation are to convert each functional task into a software-model block, and to transform cryptographic configurations into their corresponding cryptographic primitives. Since our software model will be used only for security verification, we only concern ourselves with aspects of the architecture if they impact the security of the system. We therefore preserve the existance of each task, its behavior including security operations, and all inter-task communications. Furthermore, to preserve a task's accessible keys, we add the key as an attribute to each software block if its corresponding task could access a memory storing that key through a secure path. On the other hand, we disregard simulation parameters such as cache miss or pipeline size. We also do not distinguish between tasks mapped to CPUs and Hardware Accelerators since, while the tasks mapped to Hardware Accelerators would ultimately not be present in a software model, we must nonetheless maintain their existance to examine inter-task communications at this stage.

Also, each communication path between tasks – mapped to a set of buses, bridges, memories – is translated into an abstract software-model channel. Any of the communication nodes to which a path is mapped can be tagged by the user as *secure* – which corresponds in our context to the impossibility for an attacker to access it. If a path is mapped to a set of communication nodes that are all tagged as *secure*, the resulting software-model channel will be *private*. Otherwise, a *public* channel is used.

4.5 Verification Results

ProVerif verification is performed on the translated specification, and results are displayed on the graphical diagrams for user convenience so they are visible as the user continues to refine the design. Figure 7 shows the security verification results for the unsecured mapping 2 with remote processing, and then the verification results for the automatically generated secured model.

Data exchanged across channel vulnerable to an attacker is indicated by a crossed-out red lock. However, data proved confidential is indicated by a green lock. The name of the mapping is displayed next to verification results. If a channel sends data encrypted with a Cryptographic Configuration, then we display the security verification result for both unencrypted and encrypted data. For example, *imgData* is sent across a channel that is mapped to an insecure path in architecture for mapping *Mapping2_enc*, but data sent across that channel encrypted with Crypto-

Table 1: Performance Results over Mappings.

Mapping	Drone CPU Usage (%)	Drone Bus Usage (%)	Execution Time (cycles)
Mapping 1 (On-board processing)	95.8	1.7	2933
Mapping 1 Secured	99.8	0.4	5748
Mapping 2 (Remote processing)	55.8	0.1	1855
Mapping 2 Secured	77.3	0.1	3419

graphic Configuration *autoEncrypt_imgData* is confirmed secure. Weak Authenticity and Strong Authenticity are indicated by the split lock with the half-locks colored green for verified true and half-locks colored red and crossed out for verified false. For example, on the unsecured model, *missionCmnd* is verified regarding strong and weak authenticity while *motorCmnd* is not. When the authenticity results refer to data secured by a Cryptographic Configuration, the lock is annotated with the name of that configuration.

As shown, initially image data and motor commands can be guaranteed neither confidential nor authentic, but our automatically generated secured model ensures that data encrypted with Cryptographic Configurations are secure.

4.6 Impact of Security on Safety and Performance

Security mechanisms add overhead, affecting safety properties, schedulability, and performance. Forging a nonce or performing encryption occupy computation cycles, and encrypted data may include extra bits compared with original data. In addition, sending an encrypted message sometimes means sending more bits, therefore message transfer time is increased, and more memory is used to store the extra data. A chosen 'best' mapping may later be demonstrated non-optimal with the addition of necessary security mechanisms.

We evaluated the performance of the mappings for our disaster relief drone. Table 1 shows simulation results over 10,000 runs for Mapping 1 with all tasks mapped to the drone (On-board processing), and for Mapping 2 with minimal tasks mapped to the drone (Remote processing), and both their automatically generated secured models. As expected, Mapping 1's onboard CPU has much higher usage than Mapping 2's, and since computationally intensive tasks are performed on a slower drone CPU, execution time is increased as well. Upon securing all data broadcast wirelessly, there is a significant increase in on-board CPU usage and execution time. We could explore other architectures, such as addition of extra CPUS or Cryptographic Accelerators dedicated to performing encryption, assuming the drone could support additional components.

To minimize the monetary and performance costs, security mechanisms and dedicated cryptographic accelerators should be added only where necessary. As shown, our toolkit determines the location of insecure data exchange and suggests a basic placement of security mechanisms for an architecture. A designer can then adapt and customize the model or architecture accordingly with provided performance results. For any provided architecture and mapping, our toolkit formally verifies the confidentiality and weak/strong authenticity of sensitive data.

5 RELATED WORK

Many design environments focus on the design process of embedded systems (Balarin et al., 2003; Rosales et al., 2014; Kangas et al., 2006), and many on the verification of security, but few study both in one framework. (Hansson et al., 2010) relies on Architecture Analysis and Design Language (AADL) models to consider architectural mapping during security verification. The authors note that a system must be secure on multiple levels: software applications must exchange data in a secure manner, and also execute on a secure memory space and communicate over a secure channel. Our approach, however, considers security regarding protection against external attackers instead of access control.

Another approach performs Design Space Exploration on a vehicular network protecting against replay and masquerade attacks (Lin et al., 2015). The project evaluates possible security mechanisms, their effects on message sizes, and candidate architectures during the mapping phase. While their work targets automotive systems and network communications, our analysis may be applied more broadly for any embedded system. Also, we focus on the integrity and confidentiality of data itself instead of on specific attacks.

5.1 Security Modeling

Many works address the modeling of security during software design only. SecureUML enabled the design and analysis of secure systems by adding mechanisms to model role-based access control (Lodderstedt et al., 2002). Authorization constraints are expressed in Object Constraint Language (OCL) for formal verification. Our security model focuses on protecting against an external attacker instead of access control. In contrast to formula-based constraints or queries, our approach to security analysis relies on graphically annotating the security properties to query within the model.

Another work (Vasilevskaya and Nadjm-Tehrani, 2015) proposed modeling security in embedded systems with attack graphs to determine the probability that data assets could be compromised. While their approach is also UML-based, they focus on estimating probabilities of success for attacks, while ours focuses on verifying adequate placement of encryption.

UMLSec (Jürjens, 2002) is a UML profile for expressing security concepts, such as encryption mechanisms and attack scenarios. It provides a modeling framework to define security properties of software components and of their composition within a UML framework. It also features a rather complete framework addressing various stages of model-driven secure software engineering from the specification of security requirements to tests, including logic-based formal verification regarding the composition of software components. However, UMLSec does not take into account the HW/SW Partitioning phase necessary for the design of IoTs.

5.2 Formal Verification of Security

Assessing security properties when designing software components relies on formal approaches. Heitmeyer et al. verified data separation between partitions with possibly different security levels for embedded devices. (Drouineaud et al., 2004) introduces a first order Linear Temporal Logic (LTL) into the Isabelle/HOL theorem prover, allowing the modeling of both a system and its security properties, but unfortunately leading to non-easily reusable specific models.

Many other projects formally verify security protocols using a variety of modeling and programming languages. (Avalle et al., 2014) provides a recent survey of various works. From F#, C, and Java-coded protocols, the surveyed works generate models for evaluation in a variety of abstract modeling languages such as Horn clauses, ASPIER, and First-order logic. Our toolkit however, translates graphical models.

The Software Architecture Modeling (SAM) framework (Ali et al., 2009) aims to bridge the gap between informal security requirements and their formal representation and verification. SAM uses formal and informal security techniques to accomplish defined goals and mitigate flaws. SAM relies on a well established toolkit - SMV - and considers a threat model, but the ”security properties to proof” process is not yet automated. In contrast, our work focuses on automatic formal verification from an abstract partitioning model.

6 FUTURE WORK AND CONCLUSION

Designing safe and secure embedded systems requires us to explicitly take into account these two constraints early during the development process. Since safety and security can strongly impact the HW/SW partitioning of these systems, the paper proposes a new approach to capture both security mechanisms and security properties, and evaluate them with regards to security and performance properties.

Our contribution enhances partitioning models with annotations for secure data exchange (including cryptographic configurations), offers a utility to automatically generate secured models, and proposes a model-to-ProVerif transformation, all implemented in our toolkit. We have tested our toolkit on partitioning and securing larger systems, leading us to conclude that our method can be used to model, secure, and analyze real world industrial embedded systems in a variety of disciplines (such as telecom and automotive systems, up-to-date cryptographic protocols, etc.).

Our future work will extend our security modeling to access control, timestamps, self-updating nonces (sequence numbers), and other considerations. Proof of accuracy and soundness of the transformation is another important future work. We also plan to facilitate the capture of security constraints and mechanisms.

We envision that our contribution to modeling of security during partitioning offers designers increased support in the design of secure embedded systems.

REFERENCES

Ali, Y., El-Kassas, S., and Mahmoud, M. (2009). A rigorous methodology for security architecture modeling and verification. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, volume 978-0-7695-3450-3/09. IEEE.

Apvrille, A. (2015). Geek usages for your Fitbit Flex tracker Hack.lu, Luxemburg, October 2015. Slides at framadrive.org/index.php/s/Wk6nxAKMpVTdQl4.

Apvrille, L. (2003). TTool. ttool.telecom-paristech.fr.

Apvrille, L. and Roudier, Y. (2015). SysML-Sec: A Model Driven Approach for Designing Safe and Secure Systems. In *3rd International Conference on Model-Driven Engineering and Software Development, Special session on Security and Privacy in Model Based Engineering*, France. SCITEPRESS Digital Library.

Avalle, M., Pironti, A., and Sisto, R. (2014). Formal verification of security protocol implementations: a survey. *Formal Aspects of Computing*, 26(1):99–123.

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., and Sangiovanni-Vincentelli, A. (2003). Metropolis: An Integrated Electronic System Design Environment. *Computer*, 36(4):45–52.

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 82–, Washington, DC, USA. IEEE Computer Society.

Constantin, L. (2016). Researchers hack Tesla Model S with remote attack. http://www.pcworld.com/article/3121999/security/researchers-demonstrate-remote-attack-against-tesla-model-s.html.

Costin, A. and Francillon, A. (2012). Ghost in the Air(Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices. In *BLACKHAT 2012, July 21-26, 2012, Las Vegas, NV, USA*, Las Vegas, USA.

Drouineaud, M., Bortin, M., Torrini, P., and Sohr, K. (2004). A first step towards formal verification of security policy properties for RBAC. In *QSIC'04*, pages 60–67, Washington, DC, USA.

Hansson, J., Wrage, L., Feiler, P. H., Morley, J., Lewis, B., and Hugues, J. (2010). Architectural Modeling to Verify Security and Nonfunctional Behavior. *IEEE Security Privacy*, 8(1):43–49.

Jürjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK, UK. Springer-Verlag.

Kangas, T., Kukkala, P., Orsila, H., Salminen, E., Hännikäinen, M., Hämäläinen, T. D., Riihimäki, J., and Kuusilinna, K. (2006). UML-based Multiprocessor SoC Design Framework. *ACM Trans. Embed. Comput. Syst.*, 5(2):281–320.

Kienhuis, B., Deprettere, E. F., Wolf, P. v. d., and Vissers, K. A. (2002). A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, pages 18–37, London, UK, UK. Springer-Verlag.

Lin, C.-W., Zheng, B., Zhu, Q., and Sangiovanni-Vincentelli, A. (2015). Security-Aware Design Methodology and Optimization for Automotive Systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 21(1):18.

Lodderstedt, T., Basin, D. A., and Doser, J. (2002). SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML'02, pages 426–441, London, UK, UK. Springer-Verlag.

Lugou, F., Li, L. W., Apvrille, L., and Ameur-Boulifa, R. (2016). SysML Models and Model Transformation for Security. In *Conferénce on Model-Driven Engineering and Software Development (Modelsward'2016)*, Rome, Italy.

Maslennikov, D. (2010). Russian cybercriminals on the move: profiting from mobile malware. In *The 20th Virus Bulletin Internation Conference*, pages 84–89, Vancouver, Canada.

Pedroza, G., Knorreck, D., and Apvrille, L. (2011). AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties. In *The 11th IEEE Conference on Distributed Systems and New Technologies (NOTERE'2011)*, Paris, France.

Rodday, N. (2016). Hacking a Professional Drone. Slides at www.blackhat.com/docs/asia-16/materials/asia-16-Rodday-Hacking-A-Professional-Drone.pdf.

Rosales, R., Glass, M., Teich, J., Wang, B., Xu, Y., and Hasholzner, R. (2014). MAESTRO— Holistic Actor-Oriented Modeling of Nonfunctional Properties and Firmware Behavior for MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.*, 19(3):23:1–23:26.

Schweppe, H., Roudier, Y., Weyl, B., Apvrille, L., and Scheuermann, D. (2011). C2x communication: Securing the last meter. In *The 4th IEEE International Symposium on Wireless Vehicular Communications: WIVEC2011*, San Francisco, USA.

Smyth, B., Ryan, M. D., and Chen, L. (2015). Formal analysis of privacy in Direct Anonymous Attestation schemes. *Science of Computer Programming*, 111(2).

Tanzi, T. J., Sebastien, O., and Rizza, C. (2015). Designing Autonomous Crawling Equipment to Detect Personal Connected Devices and Support Rescue Operations: Technical and Societal Concerns. *The Radio Science Bulletin*, 355(355):35–44.

Vasilevskaya, M. and Nadjm-Tehrani, S. (2015). *Quantifying Risks to Data Assets Using Formal Metrics in Embedded System Design*, pages 347–361. Springer International Publishing, Cham.