

# TOWARDS SAFE AND SECURE COMPUTER BASED RAILWAY INTERLOCKING SYSTEMS

SÉBASTIEN MARTINEZ<sup>1</sup>, DALAY ISRAEL DE ALMEIDA PEREIRA<sup>1</sup>, PHILIPPE BON<sup>1</sup>, SIMON COLLART-DUTILLEUL<sup>1</sup> & MATTHIEU PERIN<sup>2</sup>

<sup>1</sup>Univ. Lille Nord de France, France

<sup>2</sup>Institut de Recherche Technologique Railenium, France

## ABSTRACT

Railway interlocking systems (RIS) are responsible for the control of trains' movements by allowing or denying their routing according to safety rules. Originally built as mechanical systems then as electrical mechanical systems, they are modelled as relay-based electrical circuit diagrams and checked manually. Even recent computer based RIS are still modelled as relay diagrams. Manual checking of safety in such a critical context is an issue that needs to be addressed by the railway domain. In this context, one of the objectives of the LCHIP project is the development of computer-based RIS based on the formal specification of the RIS behaviour, which allows the generation of formally checked computer binaries by refinement. However, the produced binaries may be accessed and tampered with by ill-intended persons as computer code is easier to analyse and understand than electrical mechanical systems. In the context of the LCHIP project, binaries are run on a Microchip PIC32 MCU embedding two independent 32-bit MIPS processors. The choice of such generic architecture allows easier production and maintainability, lower production costs and easier replacement in case of hardware failure. It also makes it easier to reverse engineer compiled software as the MIPS architecture is well documented and has a small instruction set. This paper presents how binaries can be obtained using the B-method formal language and it discusses the usage of software obfuscation as a way to ensure security of these binaries. Obfuscation transforms a piece of software to make it impossible to understand for potential attackers, while keeping its observable behaviour: obfuscated software give the same outputs for the same inputs as their unobfuscated counterparts. As the obfuscation transformations do not alter the behaviour of formally checked generated binaries in a way that would void the safety checking, it is possible to secure computer based RIS and prevent ill-intended persons from tampering with them.

*Keywords:* code obfuscation, formal methods, railway interlocking systems, safety, security.

## 1 INTRODUCTION

Railway interlocking systems (RIS) are responsible for controlling the trains' movements in a determined track by allowing or denying their movements as a way to avoid the occurrence of hazardous situations, like collisions, for instance. As the occurrence of failures may cause severe consequences, these systems are considered as safety-critical, which requires advanced approaches in order to guarantee their safety. The first built RIS was purely mechanical, then it evolved to use new technologies, becoming electrical mechanical systems, relay-based systems and, more recently, computer-controlled systems [1]. Relay-based RIS are still used by many railway infrastructure managers, like SNCF (the French National Railway Company). The logic of these systems are implemented as electrical circuits containing relays, an electrical component that may control the flux of electrical current inside the circuit. However, relay-based RIS are generally modelled by relay diagrams, whose informal error prone verification made by manually inspection may not guarantee the absence of errors [2]. Furthermore, as computer-based systems are easier to handle and maintain, cheaper and more flexible to extend functions [3], the industry has interest on the creation of approaches in order to transform the existing relay-base systems into computer-based systems.

In this context, the LCHIP Project<sup>1</sup> was founded with the objective of using formal methodologies of specification in order to support the analysis, verification and code generation of RIS. The formal specification language used in this project is B-method [4], which has been successfully used in many railway projects, like [5], [6] and [7], and whose mathematical background and supporting tools allow the automatic safety proof and refinement in order to generate safety-proved computer-controlled systems. The generated code may then be installed on industrial micro-controllers embedding two processors as a way to support software redundancy, allowing the code generated in two different refinement processes to run on both processors. If both executions give different results, these results are considered as errors and the system presents a different safe result instead. As micro-controllers are produced in industrial quantities, the production of a computer-based RIS tends to be cheaper than the production of its relay-based version.

However, using common technologies such as mass produced micro-controllers and computer software has an issue: as the system gets easier to be accessed and understood, ill-intended attackers may try to tamper with it, causing unexpected accidents or disrupting the traffic. Reducing the production costs generally also reduces the costs for attacking the system, causing security issues. Achieving safe and secure computer based systems requires raising the costs of attacks. Thus, by making programs more difficult to understand by human readers, code obfuscation could increase the costs of attacks on RIS. Deobfuscating a program is a costly task that must be achieved in order to understand its semantic and to tamper with it. Moreover, code obfuscation allows specific strategies for protecting a program from tampering, even when attackers achieve understanding its semantics.

In this context, this paper presents some recent results in the LCHIP Project regarding how the Relay-based RIS may be specified, safety-proved and refined in order to generate computer-controlled systems. Furthermore, the protection of generated code from attacks using code obfuscation as a way to avoid external unwelcome interference is also discussed. Section 2 introduces how industry models the relay-based RIS, which may be formally specified and safety proved according to the methodology presented in Section 3. Then, Section 4 presents how models may be transformed into computer-based RIS through formal specification refinement. The protection of these generated programs are then discussed in Section 5 and the conclusion of this paper is presented in Section 6.

## 2 RELAY DIAGRAM MODELS

RIS are the part of the signalling systems responsible for the logic behind the control of the trains' movements. The track-side sensors information is processed inside the RIS, which responds by controlling the signals and turnouts according to safety requirements [8]. In industry, these systems are implemented using some different technologies, like Relay-based or computer-controlled, being this last one the most recent [1]. Despite the existence of new technologies, many railway infrastructure managers still use relay-based electrical circuits in order to implement RIS. This is the case of the SNCF, for instance.

A Relay-based RIS is the implementation of the interlocking logic by the use of electrical circuits containing relays. A Relay is an electromechanical component that can control the flux of electrical current inside the system by changing contacts positions. In this context, a component is activated if it is directly or indirectly connected to a positive and a negative energy source as a way to allow a flux of electrical current inside it.

---

<sup>1</sup> (Low Cost High Integrity Platform) <https://www.clearsy.com/en/4260-2/>

Table 1: Elements that may be used in a relay-based diagram.

		Energy sources.
		A lever and a button, respectively.
		Monostable and bistable relays, respectively.
		Blocks for timed activation and deactivation, respectively.
		A monostable and a bistable contact, respectively.

As electrical circuits, these systems are generally modelled as relay diagrams, which is a representation in a graph form of how the components are connected by electrical wires. As each railway company has its unique approach for modelling relay diagrams, this work focuses on the models used by SNCF. Table 1 presents some of the electrical components that may be used inside a relay diagram. A real industrial example is presented in Fig. 1. More details about the SNCF relay-based RIS. modelling approach may be found in [9].

A contact may be opened or closed as a way to block or allow, respectively, the flux of electrical current inside the wires. The contacts states are controlled by the relay states. A relay may be monostable or bistable. The former contains only one electromagnet that pulls or pushes the contacts against gravity when activated. In this case the contacts are placed horizontally in the physical system. A bistable relay contains two electromagnetic coils that pull the vertically positioned contacts to the left or right sides. The main difference between a monostable and a bistable relay is their contacts states when the electromagnetic coils are deactivated. When a monostable relay is deactivated, gravity makes the related monostable contacts to change their states. However, when both bistable relays coils are deactivated, gravity maintains the last state of the related bistable contacts.

The interaction between the system and the environment are made by the inputs and outputs. In this case, buttons and levers are the most known RIS interfaces. Similarly to contacts, they control the flux of the electrical current. However, their states are entirely controlled by the environment. In order to control the trains' movements, the relay-based RIS outputs are generally the commands for the signals and turnouts to change their states. Thus, in order to guarantee the safety of these systems, one must assure that the signals and turnouts do not lead the system to a hazardous situation, like a collision, for instance.

Some components may require some intending time to be activated or deactivated. This timed behaviour is implemented by complex structures which are abstracted inside relay diagrams as blocks. A relay connected to a block may have its activation or deactivation delayed according to the type of the block and the time depicted inside it.

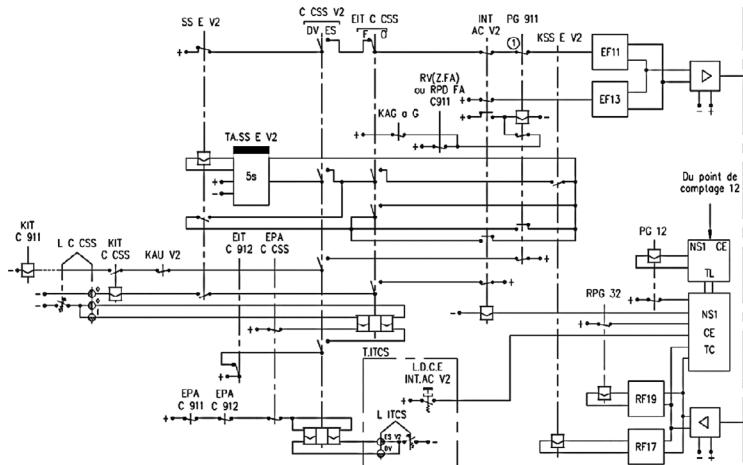


Figure 1: Partial relay diagram of a solution for the ITCS problem [10].

### 3 FORMAL SPECIFICATION OF RELAY-BASED RIS

As a way to allow the formal specification of relay-based RIS, the use of B-method as a formal language has been proposed in a previous work [10]. B-method is considered as one of the strongest approaches for the specification of railway systems [11] and its success has been proved by its use in many industrial projects ([5], [6], [7]).

In B-method, the specification is divided into smaller building blocks: the abstract machines [12]. These machines are subdivided into clauses, each one containing some details about the specified system. Some of these clauses are:

- SETS, containing sets of constant information;
- VARIABLES, which defines the system variables;
- INVARIANT, defining the variables types and properties that must be met during the system execution;
- INITIALISATION which presents the initial values of each variable;
- and OPERATIONS, describing the system state evolution.

Details about B-method syntax and semantics as well as its mathematical foundations may be found in [4].

In the context of this work, B-method supports the specification of the preconditions for each electrical component to be activated inside a relay-based RIS. Based on this behavioural specification, it is possible to analyse the impact of a component activation over the other components states and define safety properties that must be met during the system execution. Furthermore, the use of B-method supporting tools allows the animation and automatic safety proof of the system, as presented later in this section.

#### 3.1 Relay-diagram behavioural logic

The relay-based RIS logic is based on the precondition for each component to be activated. The activation of a relay, for instance, changes the state of contacts related to it, which may allow the electrical current to flow in other wires, provoking the activation of other components. The precondition for the activation of a component is presented in Definition 3.1.

**Definition 3.1.** (*Component Activation Precondition*) An electrical component is activated if both of its wires are connected to a different pole (positive and negative) of energy sources in a way that allows the flow of electrical current inside the component. This means that all contacts, buttons and levers between the component and both poles of energy sources must be closed.

A component activates once there is an electrical current flux inside it. Following the same logic, it is possible to define a precondition of the deactivation of a component, as presented in Definition 3.2.

**Definition 3.2.** (*Component Deactivation Precondition*) An electrical component is deactivated if its wires are not connected to different poles (positive and negative) of energy sources, preventing the flow of electrical current inside it. This means that at least one contact, button or lever between the component and one pole of energy source must be opened (considering that there is no other connection to the same type of pole).

The activated and deactivated states of a monostable relay may be represented by means of binary values, where TRUE is for the activated and FALSE for the deactivated state. The state of a monostable relay changes as soon as the preconditions for this state is no longer met. Nonetheless, as a bistable relay states depend on the activation and deactivation of two different coils, the representation of its states tends to be more complex. It is a known characteristic of relay-based systems that both coils of a bistable relay must never be activated at the same time. This component assumes a “right” state as soon as the right coil is activated and the left coil is deactivated. Following the same logic, this component assumes a “left” state when the left coil is activated and the right coil is deactivated. However, if both coils deactivate, the bistable relay has its last assumed state maintained by the gravity.

The contacts states depend exclusively on the relays states in a way that their behaviours may be completely abstract by their related relays states. Thus, the activation and deactivation of some components is a consequence of the relays states. Besides, levers and buttons may also impact the activation and the deactivation of components as they can also control the flux of electrical current inside the wires. Nevertheless, as these components are part of the system interface, they are controlled by the environment and they are treated as inputs for the definition of the system state.

The analysis of the RIS safety involves the study of the system outputs according to the given inputs. Output components may also be activated (TRUE) or deactivated (FALSE) and their states may indicate the permission for a train to continue its route. In the example presented in Fig. 1, the components *EF11* and *KIT C 911* are responsible for allowing two trains to enter in the same track in opposite directions. In this case, these outputs must be analysed in order to guarantee that they will never be activated at the same time, which could lead to a frontal collision between the trains. As a manual analysis may not be satisfactory in this safety-critical context, the specification of the relay-based RIS may provide a safety proof of the system based on mathematical foundations.

### 3.2 Relay-based logic specification in B-method

The relay-based RIS behaviour may be specified in B-method by defining the content of each clause of the B-machine. In the *SETS* clause one may define some special components states for levers or bistable relays. In our running example, for instance, it is defined the states *POS\_O* and *POS\_F* for the “right” and “left” states of the bistable relay *EIT\_C\_CSS*. The *SETS* clause of our running example is presented in Fig. 2.

```

MACHINE
    itcs
SETS
    O_OU_F = {POS_O, POS_F};
    DV_OU_ES = {POS_ES, POS_DV}
VARIABLES
    KIT_C_CSS,
    SS_E_V2,
    TA_SS_E_V2,
    EIT_C_CSS,
    C_CSS_V2,
    PG_911,
    EF11,
    KIT_C_911

```

Figure 2: B-method MACHINE, SET and VARIABLES clauses.

```

INVARIANT
    KIT_C_CSS : BOOL &
    SS_E_V2 : BOOL &
    TA_SS_E_V2 : BOOL &
    EIT_C_CSS : O_OU_F &
    C_CSS_V2 : DV_OU_ES &
    PG_911 : BOOL &
    EF11 : BOOL &
    KIT_C_911 : BOOL &

    not (KIT_C_911 = TRUE &
         EF11 = TRUE)

INITIALISATION
    KIT_C_CSS := FALSE ||
    SS_E_V2 := FALSE ||
    EIT_C_CSS := POS_F ||
    C_CSS_V2 := POS_DV ||
    PG_911 := TRUE ||
    TA_SS_E_V2 := FALSE ||
    EF11 := FALSE ||
    KIT_C_911 := FALSE

```

Figure 3: B-Machine INVARIANT and INITIALISATION clauses.

The state of relay-based RIS is defined by the specific state of each component. In a B-machine, the system state is defined by the state of each variable. Thus, in our methodology, each component of the relay-based RIS is defined as a variable of the B-machine. Nonetheless, as a way to simplify the specification without impacting on the system logic, it is possible to abstract the contacts, since their states may be easily deduced from their related relays during the system execution. Furthermore, as B-method supports the definition of inputs in the system execution description, the RIS inputs are not treated as variables. These specification decisions have no negative impact on the system safety verification, animation and analysis. In fact, the less amount of variables decrease the number of states and allows a faster and lighter verification.

The variables typing must be specified inside the *INVARIANT* clause. These types represent the possible states that the components may assume during the system execution. Relays and outputs, for instance, may be typed as Boolean, since they may assume the activated (TRUE) and deactivated (FALSE) states. Bistable relays must be typed according to the special types created in the *SETS* clause. Furthermore, it is also possible to define safety properties that must be met during the system execution inside the *INVARIANT* clause. For instance, as the components KIT\_C\_911 and EF11 may never be activated at the same time in our running example, this property may be defined as presented in Fig. 3.

The initial state of a relay-based RIS is the functional state depicted in the relay diagram itself. So, it is possible to define the initial state of the specified system in the *INITIALISATION* clause according to the interpretation of the relay diagram. The specification of the initial state of our running example is presented in Fig. 3.

The system execution logic is specified in a B-machine inside the *OPERATIONS* clause. So, the state evolution of a RIS may be specified as a single operation that changes the components states based on the inputs values given by the environment. A B-operation requires

```
KIT_C_CSS = bool(SS_E_V2 = TRUE & EIT_C_CSS = POS_O &
INT_AC_V2 = TRUE & L_C_CSS = POS_O)
```

Figure 4: Example of a state evolution.

the definition of a precondition where all the inputs must be typed. The logic for the complete system state evolution may be specified using the following notation:

```
<<variables>>:(<<variables typing>> & <<variables information>>)
```

This expression allows the evolution of the system variables (`<<variables>>`) by informing their types (`<<variables typing>>`) and the conditions that the system must meet after the execution of this expression (`<<variables information>>`). In this approach, these conditions are the precondition for each component to be activated or deactivated. In our running example, for instance, the precondition for the component `KIT_C_CSS` to be activated is represented inside Fig. 4.

### 3.3 Animation and verification

The ProB tool [13], allows not only the animation of the machines but also their specification and model-checking. During the animation, the tool allows operations to be called and it always verifies if the machine state is valid according to the invariant. In this case, the ProB tool contains a model-checker providing a counter example when an invalid state is found.

## 4 BUILDING A COMPUTER-BASED RIS

The B-method allows the specification refinement and disposes of a set of logical conditions that can be verified in order to guarantee that the refined machine presents indeed the same behaviour as the more abstract one. Thus, this methodology allows the refinement of the specification in order to implement safety-proved systems. Furthermore, the automatic code generation is a feature supported by the Atelier B tool , which allows the safe transformation from a B-method implementation to software code. The refinement of B-method specifications have been discussed for years in the literature and many different approaches have been already presented [15]. Besides, the automatic refinement methodologies have also been discussed [16].

In the LCHIP Project, the generated source code is compiled and deployed on a Microchip PIC32 MCU embedding two independent 32-bit MIPS processors. Each processor runs a different version of the program generated by Atelier B using a different generation chain. In normal behaviour of the RIS, both versions of the program give the same outputs for the same inputs. The outputs of the RIS are the outputs of the programs. If the programs give different outputs (for example, because of a hardware default), the RIS system is considered as faulty and the global system, of which the RIS is part of, falls back to a safety position.

Such safety rule ensures a higher resilience to errors and hardware faults but does little for protection against attacks. The methodology for the specification and implementation of Relay-based RIS as safety-proved computer-controlled systems may increase the safety of these systems, but it may also induce a vulnerability to attacks, which is a security problem. To address that issue, the use of obfuscation techniques is discussed.

## 5 DISCUSSION ON SOFTWARE OBFUSCATION

Computer based RIS are vulnerable to reverse engineering and tampering by ill-intended people. Binaries could be downloaded through physical access and disassembled, allowing access to their source code. An ill-intended person could analyse the source code, modify it

and install that modified version in the system, at best to disrupt the traffic, at worst to cause accidents. As the LCHIP project uses the well documented MIPS architecture for running binaries, such task would not require much effort and resources. In order to prevent such attacks, we consider code obfuscation: the transformation of programs into more complex versions, harder to understand for possible attackers.

According to the definition of obfuscation by Collberg et al. [17, 18], obfuscating a program does not change its *observable behaviour*. Observing an obfuscated program as a black box does not allow its distinction from its non-obfuscated version. Such approach allows the usage of obfuscation as it does not invalidate formal proofs on the non-obfuscated program.

**Definition 5.1 (Obfuscating transformation).** Let  $\mathcal{P} \xrightarrow{\tau} \mathcal{P}'$  be a transformation of a source program  $\mathcal{P}$  into a target program  $\mathcal{P}'$ .

$\mathcal{P} \xrightarrow{\tau} \mathcal{P}'$  is an obfuscating transformation if, given the same inputs:

- If  $\mathcal{P}$  fails to terminate or terminates with an error, then  $\mathcal{P}'$  may or may not terminate
- Otherwise,  $\mathcal{P}'$  terminates and produces the same output as  $\mathcal{P}$

In [17], Collberg et al. classified obfuscating transformations in three categories. *Layout transformations* affect the source code of the program and make its understanding more difficult for human readers. A common layout transformation is variable scrambling: renaming

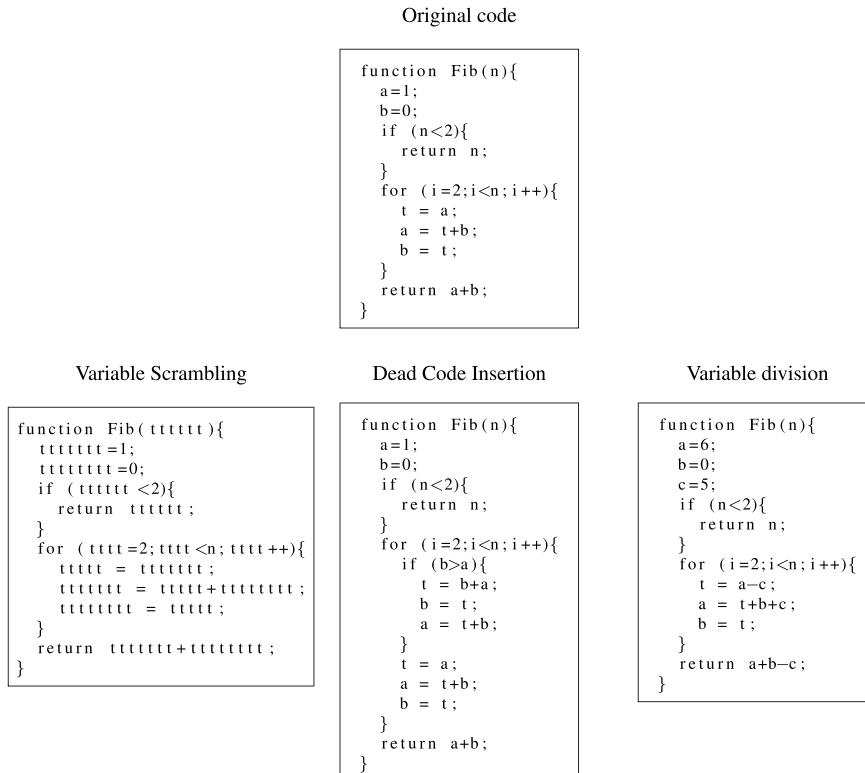


Figure 5: Examples of obfuscation transformations.

variables to meaningless sequences of characters, easily confused with each other. *Control transformations* affect the control flow of the program, making its behaviour more complex. A common control transformation is the insertion of *dead code*: useless code is inserted in the program but never executed. *Data transformations* affect data structures used by the program, introducing complexity. A common data transformation is the division of a variable into two apparently independent variables. Figure 5 presents examples of applications of variable scrambling, dead code insertion and variable division to an iterative function calculating the  $n^{\text{th}}$  element of the Fibonacci sequence.

Quantifying the quality of an obfuscating transformation is a complex issue as it implies quantifying the semantic complexity (*i.e.* how hard it is to understand) of a program, which is subjective. Indeed, two different people may find a same program easy or difficult to understand. Several metrics for semantic complexity are presented in the literature. For example, Harrison and al. [19] proposed measuring the nesting level of programs to measure their complexity. Collberg et al. defined three measures on obfuscating transformation in order to quantify their quality: *Potency*, *Resilience* and *Cost*. Each of these measures are calculated on the application of a given transformation on a given program. Potency measures the gain of semantic complexity when applying the transformation on the program. Resilience measures the difficulty of reverse engineering the obfuscated program to retrieve its original version. Cost measures the overhead introduced on the execution time of the program after applying the transformation. Ideally, a good obfuscating transformation has a high resilience, a high potency and a low cost. The quality of an obfuscating transformations cannot be measured as such. The quality of an obfuscating transformation depends on the program it is applied to. As a consequence, obfuscating a given program requires finding a sequence of obfuscating transformations suited to that program: a sequence of transformations of high quality when applied to the given program.

In the context of the LCHIP project, programs must comply with safety regulations that restrain programming techniques. As a consequence, some obfuscating techniques cannot be used for computer based RIS. For example, the SIL4 regulation applies on the LCHIP project and forbids the usage of pointers in programs. As a consequence, obfuscating transformations using pointer aliasing to obfuscate the control flow or the data flow of the program cannot be used. Pertinence of obfuscation in the context of the LCHIP project would depend on the finding of high quality obfuscating transformations complying with safety regulations.

The step of the software production chain in which the program is obfuscated may have an impact on the quality of the obfuscation. Some steps like the transformation of the B model into source code or the compilation of the source code into an executable binary are based on transformations. Common compilers not only translate source code into binaries, they also apply transformations optimizing the program by, for example, reducing its execution time or reducing its resources consumption. Such transformations alter the structure of the program and may degrade the obfuscation of the program. While applying them early in the software production chain is usually easier, obfuscating transformations should be applied as late as possible to avoid interference by other transformations.

### 5.1 Tamper-proofing computer-based RIS

Obfuscation in itself does not protect programs from tampering. It makes the analysis of the program more difficult and therefore may make the program more difficult to tamper with. In the context of computer based RIS, expected outputs are specified and documented.

Attackers managing to access the RIS documentation could use it to generate a rogue program and cause accidents without needing to deobfuscate the original program. Even without accessing the documentation, attackers may manage to deobfuscate the program with enough time and resource. Indeed, obfuscation is never a definitive answer because deobfuscation is always possible [20], although it may be highly time and resource consuming. To protect computer based RIS, we discuss using obfuscation as a tool to tamper-proof programs.

Obfuscating transformations can be used to watermark and tamper-proof programs [21]. When watermarking a program, obfuscation is used to hide information inside the program that can be retrieved later using a secret key. This technique can be used to prove intellectual ownership of a program in the case of legal conflicts. One method for watermarking programs using obfuscation is to encode the watermark as a sequence of obfuscating transformations applied to the program. The secret key would be a function that extracts that sequence of transformations from the obfuscated program. In the following example,  $\tau_0, \dots, \tau_n$  is the watermark and  $\mathcal{K}$  is the secret key.

$$\mathcal{P} \xrightarrow{\overbrace{\tau_0 \dots \tau_n}} \mathcal{P}' \quad \mathcal{K}(\mathcal{P}') = \tau_0, \dots, \tau_n$$

Several methods for tamper-proofing programs can be found in the literature. A common method, used in most Linux distributions, checks a hash value or a RSA signature of the binary. Another method presented by Aucsmith [22] encrypts the binary and decrypts it, piece by piece, at execution time. Unfortunately, we could not find tamper-proofing methods that could be directly applied in the context of the LCHIP project. The execution platform would not support a complex encryption/decryption process and hash value checking cannot be done without potential attackers noticing the checking operations.

We discuss a tamper-proofing method using program watermarking. A watermark in the form of a sequence of obfuscating techniques applied to the program is used as a proof of authenticity. An external control system, connected to the deployed RIS, would regularly apply the key function to the deployed binaries, verifying the presence of the watermark. If the watermark is not detected in one program, the control system would request the global system to fallback to a safety position, according to classical safety principles of intrinsic safety. The chosen method for detecting the presence of the watermark is more difficult to detect than regular hash checking. As a consequence the watermarking detection algorithm could be embedded in each RIS for redundancy. Successful attacks on such systems need to target the watermark checking system. By provoking false positive results, attackers could install rogue programs in deployed RIS. This can be achieved either by hacking the control system, either by producing rogue programs in which the watermark would be detected. Such attack is called *collision attack* [23]. Embedding the checking algorithm inside each deployed RIS would make attacks on the checking system difficult as it would require hacking the external control system and the RIS on which the rogue program would be deployed. Collision attacks on the key function should be difficult. Provided the watermark is made of high quality obfuscating transformations, collisions should be rare. As previously stated in this section, deobfuscation is always possible, which means our watermarking strategy is not absolutely safe from potential attacks, although it increases the cost of these attacks. With enough time and resources, attackers may be able to identify and reproduce the watermark in order to generate a rogue program. To address that issue, the watermark and its checking algorithm need to be regularly updated, as any critical software system should be. Another

possible attack would be hacking the external control system and provoke false negatives for each deployed RIS. Discussing the protection of the external control system is out of the scope of this paper.

## 6 CONCLUSION

As a way to alleviate the costs of production and maintenance of relay-based RIS and improve their safety, a methodology for the formal specification, verification, refinement and implementation of safety-proved computer-based RIS have been proposed by the LCHIP Project. This approach uses B-method and its supporting tools as a way to transform the behavioural logic of existing relay-based RIS into computer-based systems that may be executed in cheaper microchips. However, despite the safety improvements presented by this work, the security of these systems may be a problem as they run in generic hardware configurations.

This paper presents the last results published by the LCHIP Project and a discussion about its possible security problems. In order to prevent attackers from tampering with the RIS, the paper also discusses the use of code obfuscation techniques. By definition, obfuscating transformations increase the complexity of the program without invalidating the safety proof achieved using formal methods. In this context, it is possible to deploy a program difficult to understand by attackers and insert a watermark inside each generated program while preventing attackers from detecting it. The paper proposes a decentralized architecture checking the presence of that watermark to detect potentially tampered RIS that do not have the watermark. Efficiency of that method depends on the possibility to find efficient obfuscating transformations compliant with industrial safety standards.

As a future work, it is possible to implement the techniques discussed in this paper as well as to make a deep analysis of the impact of using software obfuscation on the system efficiency and security. Furthermore, it is also possible to use this work as basis in order to develop a new refinement approach for B-method that includes software obfuscation strategies as a way to develop safe and secure computer-based systems. The inclusion of software obfuscation as a way to improve the security of formal specified RIS is a new study field and literature still lacks any contribution about the subject.

## ACKNOWLEDGMENTS

This work is supported by the LCHIP Project and the results presented in this paper are a product of the studies made in this project. We thank Clearsy LCHIP team for sharing their studies with us and we also thank SNCF for providing and allowing us to publish the relay schema in this paper.

## REFERENCES

- [1] Hansen, K.M., Formalising railway interlocking systems. *Nordic Seminar on Dependable Computing Systems*, Citeseer, pp. 83–94, 1998.
- [2] Haxthausen, A.E., Le Bliguet, M. & Kjær, A.A., Modelling and verification of relay interlocking systems. *Monterey Workshop*, Springer, pp. 141–153, 2008.
- [3] Akita, K., Watanabe, T., Nakamura, H. & Okumura, I., Computerized interlocking system for railway signaling control: smile. *IEEE Transactions on Industry Applications*, (3), pp. 826–834, 1985.

- [4] Abrial, J.R., *The B-Book: Assigning Programs to Meanings*. Cambridge University Press: New York and NY and USA, 1996.
- [5] Behm, P., Benoit, P., Faivre, A. & Meynadier, J.M., Meteor: a successful application of b in a large project. *International Symposium on Formal Methods*, Springer, pp. 369–387, 1999.
- [6] Lecomte, T., Servat, T., Pouzancré, G. et al., Formal methods in safety-critical railway systems. *10th Brasilian Symposium on Formal Methods*, pp. 29–31, 2007.
- [7] Guiho, G. & Hennebert, C., Sacem software validation. *Software Engineering, 1990. Proceedings., 12th International Conference on*, IEEE, pp. 186–191, 1990.
- [8] Theeg, G., *Railway signalling & interlocking international compendium*. PMC Media House GmbH: BingenHamburg, 2017.
- [9] Rétiveau, R., *La signalisation ferroviaire*. Presse de l'école nationale des Ponts et Chaussées, 1987.
- [10] de Almeida Pereira, D.I., Deharbe, D., Perin, M. & Bon, P., B-specification of relay-based railway interlocking systems based on the propositional logic of the system state evolution. *International Conference on Reliability, Safety, and Security of Railway Systems*, Springer, pp. 242–258, 2019.
- [11] Fantechi, A., Fokkink, W. & Morzenti, A., B-specification of relay-based railway interlocking systems based on the propositional logic of the system state evolution. *Formal Methods for Industrial Critical Systems: A Survey of Applications*, pp. 61–84, 2013.
- [12] Schneider, S., *The B-Method: An Introduction*. Palgrave, 2001.
- [13] Leuschel, M. & Butler, M., Prob: a model checker for b. *International Symposium of Formal Methods Europe*, Springer, pp. 855–874, 2003.
- [14] ClearSy, *Atelier B User Manual, version 4.0*. ClearSy System Engineering, Parc de la Duranne – 320 av. Archimède – Les Pléïades III Bat A – 13857 AIX EN PROVENCE CEDEX 3 – FRANCE.
- [15] Sekerinski, E. & Sere, K., *Program Development by Refinement: Case Studies Using the B Method*. Springer Science & Business Media, 2012.
- [16] Burdy, L. & Meynadier, J.M., Automatic refinement. *Proceedings of BUGM at FM*, **99**, 1999.
- [17] Collberg, C., Thomborson, C. & Low, D., A taxonomy of obfuscating transformations. <http://wwwcsaucklandacnz/staff-cgi-bin/mjd/cstRcgipl?serial>, 1997.
- [18] Collberg, C. & Nagra, J., *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional, 1st edition, 2009.
- [19] Harrison, W. & Kenneth, M., A complexity measure based on nesting level. *ACM SIGPLAN Notices*, **16**, pp. 63–74, 1981.
- [20] Appel, A., Deobfuscation is in np, 2002.
- [21] Collberg, C.S. & Thomborson, C., Watermarking, tamper-proofing, and obfuscation – tools for software protection. *IEEE Transactions on Software Engineering*, **28(8)**, pp. 735–746, 2002.
- [22] Aucsmith, D., Tamper resistant software: an implementation. *Information Hiding*, ed. R. Anderson, Springer Berlin Heidelberg: Berlin, Heidelberg, pp. 317–333, 1996.
- [23] Wang, X. & Yu, H., How to break md5 and other hash functions. *Advances in Cryptology – EUROCRYPT 2005*, ed. R. Cramer, Springer Berlin Heidelberg: Berlin, Heidelberg, pp. 19–35, 2005.