A SYSTEM ARCHITECTURE WITH A SECURE PROTOCOL FOR REMOTE SOFTWARE TAMPERING DETECTION IN EMBEDDED SYSTEMS

ABDO ALI ABDULLAH AL-WOSABI

UNIVERSITI KEBANGSAAN MALAYSIA

A SYSTEM ARCHITECTURE WITH A SECURE PROTOCOL FOR REMOTE SOFTWARE TAMPERING DETECTION IN EMBEDDED SYSTEMS

ABDO ALI ABDULLAH AL-WOSABI

THESIS SUBMITTED IN FULFILMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

FACULTY OF INFORMATION SCIENCE AND TECHNOLOGY
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2018

SENI BINA SISTEM BESERTA PROTOKOL SELAMAT UNTUK PENGESANAN PERUBAHAN PERISIAN DALAM SISTEM TERTANAM SECARA JAUH

ABDO ALI ABDULLAH AL-WOSABI

TESIS YANG DIKEMUKAKAN UNTUK MEMPEROLEHI
IJAZAH DOKTOR FALSAFAH

FAKULTI TEKNOLOGI DAN SAINS MAKLUMAT
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2018

DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries which have been duly acknowledged.

02 May 2018

ABDO ALI ABDULLAH
AL-WOSABI
P72745

ACKNOWLEDGEMENT

First and foremost, praise be to Almighty Allah for all his help and giving me patience and good health throughout all the stages of my life.

I am extremely thankful to Allah for guiding me in choosing the best possible supervisor, Prof. Dr. Zarina Shukur, who guided me with constant and wonderful suggestions and continuously motivated me throughout the duration of this PhD research. I am really honored and consider myself lucky to have worked under her direct supervision. I will forever be grateful, Prof.

I would like to thank Dr. Mohd Rosmadi Mokhtar, Dr. Rossilawati Sulaiman, Assoc. Prof. Dr. Ravie Chandren Muniyandi, and Assoc. Prof. Dr. Elankovan A. Sundararajan for giving me additional guidance and direction during Softam seminars and several meetings. I owe my thanks to my dear brother Muhammad Azwan Bin Ibrahim (from SIRIM Berhad Malaysia) for his commendable contribution to implementing the expert evaluation review, and also to those participants for evaluating the proposed framework and protocol. Special thanks to my dear brothers and sisters, including Nadeem Alherbawi, Nader Salameh, Ahmed Hussain, Sameer Abdulbaqi, Afifah Sabri, and Zaireeda Fauzee for their help, friendship, and creating a pleasant working environment throughout my years at UKM. In addition, I would like to express my thanks to Dr. Freya Martin for proofreading my thesis prior to submission.

I am very grateful to my dad, to whom this thesis is dedicated, and who devoted his entire life and died out of love for his children. And to my mom who made it a point of duty to pray for me every day. In addition, I would like to express my high appreciation to my brothers and sisters for their prayers, patience, constant encouragement, and unconditional support, which has helped me to complete my research.

I sincerely appreciate my beloved wife and our children for their understanding, motivation, and support in encouraging me to embark on this journey.

Last but not least, I must extend my thanks and gratitude to all the academic and staff at FTSM and UKM as a whole.

ABSTRACT

No doubt, a person of modern society relying on Embedded Systems (ESs) has increased rapidly and the era of digital machines is gaining popularity among users and also systems providers. At the same time, such instruments face substantial security challenges because they usually operate in a physically unprotected environment, and thus attract the attackers to gain unauthorized access for utilizing the system functions. Accordingly, system integrity is important and hence there is a need to propose a technique/tool to verify that the original/pure systems codes have been used in those devices. In this research, our main objective is to design a system architecture with a secure communication for code integrity verification of an ES. Indeed, the study presents the proposed system architecture for ESs integrity verification which includes two main phases: fetching an ES code at a server site and examining the ES at a remote site (using a designed user application). The integrity of that ES has been verified by comparing the computed hash value, at the user site, to the digest value of the previously saved code; the result could show whether that system has been altered or tampered with. Essentially, the hash function (SHA-2) with a random key to calculate a unique digest value for a targeted system have been utilized. Also, the study used timestamps and nonce values, two secure keys, and public key algorithm to design a secure protocol in-order to prevent potential attacks during data and the associated values transfer between the server and the remote user application. As many researchers state that the formal methods are very precise and accurate for presenting system specifications; this study modeled and analyzed the proposed verification protocol using the Communicating Sequential Processes (CSP) formal method approach. Thus, the study has represented the proposed protocol and its secrecy and authentication specifications using the CSP approach. Besides, the Compiler for the Analysis of Security Protocols (Casper) has been used to translate the protocol description into the corresponding process algebra CSP model. Then, the researcher used the Failures Divergences Refinement (FDR) in-order to evaluate the proposed protocol. Those formal method tools are considered as a reliable verification measurement in-order to figure-out potential flaws and correct them. Overall, the final output of checking all the defined secrecy and authentication assertions using FDR 4.2.0, and thus all the secrecy and authentication specifications defined in the developed Casper script are passed; in other words, the FDR fails to trace any potential attack upon the proposed protocol. Additionally, the entire framework has been evaluated, and thus five experts were reviewed to determine the positive features, suggested enhancements, and obstacles and weaknesses of the proposed framework. The results obtained from expert evaluations indicate their satisfactions and they considered the proposed framework would be useful.

ABSTRAK

Tanpa diragui, kebergantungan seseorang di dalam masyarakat moden kepada sistem tertanam meningkat begitu ketara dan juga meningkat populariti era mesin digital terhadap pengguna dan pembekal sistem. Pada masa yang sama, instrument ini menarik minat penyerang untuk mengeksploitasi potensi kelemahan di dalam sistem perisian untuk memperolehi akses tidah sah bagi menggunakan fungsi sistem atau pun memperoleh data secara haram. Oleh yang demikian, integriti sistem adalah penting dan oleh itu, adalah merupakan satu keperluan untuk mencadangkan satu teknik/perkakasan berkesan untuk mengesahkan keaslian ES digunakan dalam peralatan tersebut. Dalam kajian ini, objektif utama adalah untuk membina satu seni bina sistem yang praktikal dengan komunikasi selamat untuk pengesahan kod integrity ES. Oleh yang demikian, kami membentangkan seni bina sistem cadangan kami untuk pengesahan ESs integriti yang merangkumi dua fasa utama: semasa pengambilan kod ES di tapak server (sebagai contoh pusat data) dan pemeriksaan ES di tapak terpencil (menggunakan aplikasi pengguna yang direka). Integriti ES yang telah disahkan dengan membandingkan nilai hash dikira, di tapak pengguna, kepada nilai kod cerna yang disimpan sebelum ini; hasil dapat menunjukkan sama ada sistem yang telah diubah atau diganggu. Pada dasarnya, fungsi hash (SHA-2) dengan kunci rawak untuk pengiraan nilai yang unik tercerna untuk sistem sasaran telah digunakan. Kami juga menggunakan setem masa dan nilai naunsa, dua kunci keselamatan, dan algoritma kunci awam untuk mereka bentuk satu protokol keselamatan bagi mengelakkan potensi serangan pada ketika penghantaran data atau pun nilai berkaitan antara server dan juga aplikasi pengguna terpencil. Sebagaimana kebanyakan penyelidik menyatakan bahawa, kaedah formal adalah sangat tepat dan jitu dalam mempersembahkan spesifikasi sistem, kajian ini mengguna pakai perkakas kaedah formal Proses Komunikasi Berturutan (CSP) sebagai penilaian. Oleh itu, kami membentangkan cadangan protokol dan kerahsiaannya beserta spesifikasi pengesahan menggunakan notasi CSP. Disamping itu, Pengkompil untuk Analisis Protokol Keselamatan (Casper) telah digunakan untuk menterjemah keterangan protokol kepada model proses algebra CSP yang berkenaan. Kemudian, kami menggunakan Pembaikan Kegagalan Penyimpangan (FDR) bagi menilai protokol yang dicadangkan. Perkakasan kaedah formal itu adalah di kenalpasti sebagai kaedah verifikasi yang dipercayai bagi mengenal pasti ruang kecacatan seterusnya diperbaiki. Secara keseluruhan, hasil akhir semakan semua rahsia dan pengesahan tegas menggunakan FDR 4.2.0, dan demikian, kesemua kerahsiaan dan spesifikasi kerahsiaan yang dinyatakan dalam Casper code dihantar; dalam kata lain, FDR gagal untuk mengesan sebarang potensi serangan keatas protokol yang dicadangkan. Sebagai tambahan, keseluruhan rangka kerja telah dinilai, dan untuk itu lima pakar telah melaksanakan penilaian untuk mengenalpasti ciri-ciri positif, cadangan penambahbaikan, dan halangan serta kelemahan ke atas rangka kerja yang dicadangkan. Hasil yang didapati dari penilaian pakar menunjukkan mereka mereka berpuas hati dan memberikan pandangan bahawa rangka kerja yang dicadangkan adalah bermanfaat.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\square$	external choice
(+)	(= $\oplus$) the bit-wise exclusive-or
$\forall x : A \bullet P$	for all x in set A such that P
$\neg P$	not P (P is not true)
$A \setminus S$	A without S
b ? x	on (channel) b input to x
Com	the commit event
dbServ	the database server agent
$Dec_{privKey}(C)$	denotes the decryption of cipher text C under the corresponding private key privKey
$Enc_{pubKey}(M)$	denotes the encryption of message M under the public key pubKey
env	the environment (the undefined sender) of the designed system
esCode	Code of an embedded system
esHMAC	(= ES_HMAC) represents the calculated digest value of the system's code
esId	(=ES_ID) the identifier of an embedded system
hf(x)	the hash function returns the digest value of x
hk	(=Hk) hash key
Init	the initiator agent who wills to run the protocol
integVer	result of the system integrity verification
Km	the session key which could be used by Mallory (i.e., the attacker) agent
lastVerDate	the last verification date for an embedded system
m % v	the recipient of the message should not attempt to decrypt the message m, but instead store it in the variable v
NAtc	a nonce value (i.e., random number) generated by the attacker agent
Nm	similar to NAtc

nr	(= nRec = Nrec) a nonce value (i.e., random number) generated by the receiver agent
ns	(= nSen = Nsend) a nonce value (i.e., random number) generated by the sender agent
nServSys	(= NServerSys = servNonce) a nonce value (i.e., random number) generated by the server system agent
nUsr	(= NUser = usrNonce) a nonce value (i.e., random number) generated by the user agent
P ||| Q	P interleave Q
P ∧ Q	P and Q (both true)
P ⇒ Q	if P then Q
PkMallory	the public key of Mallory (i.e., the attacker) agent
pkServSys	(= PkServerSys) the public key of the server system agent
pkUsr	(= PkUser) the public key of the user agent
PubK(X)	the public key function retrurns the public key of X
PubKeyAtc	the public key of an attacker
pubKeyR	the public key of the receiver agent
pubKeyS	the public key of the sender agent
PubKfun(X)	similar to PubK(X)
R	(= Rec) the receiver agent
regExpDate	the expiry date of the user registration
Resp	the responder agent
Run	the running event
S	(= Sen) the sender agent
SecK(X)	the secrter key function retrurns the secret (i.e., private) key of X
SecKeyAtc	the corresponding private key of PubKeyAtc
secKeyR	the corresponding private key of pubKeyR
secKeyS	the corresponding private key of pubKeyS
SecKfun(X)	similar to SecK(X)
secVal	the secret value

servSys	the server system agent
sk	(= Sk) session key used by honest agents
Skey(X)	The secure key function that returns the symmetric server’s secret key (i.e., SkeyServSys)
SkeyServSys	(= SKeyServerSys) the symmetric secure key that the server agent shares with the database server
SkMallory	the corresponding private key of PkMallory
skServSys	(= SkServerSys) the corresponding private keys of pkServSys/ PkServerSys
skUsr	(= SkUser) the corresponding private keys of pkUsr/PkUser
ts	(= TS) timestamp value
usr	the user agent
Usr_ID	the identifier of user agent
x != e	x is not equat to e
$x : A \rightarrow P$	(choice of) x from A then P
x := e	x becomes (value of) e
x = = e	equality operator evaluates to true if two values are equal
$x \in N$	x is a member of N
XOR	Exclusive-or operation

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
CA	Certificate Authority
Casper	Compiler for the Analysis of Security Protocols
CSP	Communication Sequential Processes
DoS	Denial of Service attack
DSRM	Design Science Research Methodology
ES	Embedded System
EST	Embedded System Tampering
etc	etcetera; and so forth
FDR	Failures Divergences Refinement
FPGA	Field Programmable Gate Array
FTSM	Faculty of Information Science and Technology
HMAC	Hash-Based Message Authentication Code
Id	Identifier
i.e.	that is/in other words
IS	Information System
MAC	Message Authentication Code
OTP	one-time pad
QR	Quick Response
RSA	Rivest-Shamir-Adleman algorithm
SHA	Secure Hash Algorithm
UKM	Universiti Kebangsaan Malaysia

CHAPTER I

INTRODUCTION

1.1 OVERVIEW

Embedded Systems (ESs) are now available anywhere and anytime and are an established part of daily routines. Their usage in sensing, storing, processing, and transferring personal and private data in devices such as ATM cards, modern car systems, and mobile phones has become widespread and their utility irreplaceable. However, developers of those systems face significant challenges in relation to the issue of code integrity and information security. Software tampering is one of those challenges, and software integrity verification is one of the main approaches used to defeat it.

In fact, checking code integrity achieves tamper-proofing by identifying unauthorized alterations and recognizing whether any tampered code is being executed or any tampered data are being used. Therefore, protecting the software stored in an Embedded System (ES) is of paramount importance. Obviously, this research emphasizes that there is no complete solution to this problem, and the major aim of conducting this research is to contribute positively (even if in a modest way) to fighting against software tampering in such systems.

1.2 RESEARCH BACKGROUND AND MOTIVATION

The reliance of individuals in modern society on ESs has increased rapidly, and the era of digital machines is being embraced by users and by device/machine providers alike. The advancements in ES applications are leading to the provision of user-friendly services. Embedded Systems are used to facilitate people's daily activities, and can be found in items such as digital scale machines, digital cameras, modern cars, mobiles, and ATM cards, which are used extensively most of the time every day. Thus, the security of such systems has become the key challenge and an interesting research area since human lives are affected directly by the digital revolution (Al-Sharafi et al. 2016; Singh, Dilawari & Manimegalai 2014; Subashini, & Kavitha 2012), and this will likely remain the case for the foreseeable future (Brasser, Rasmussen & Tsudik 2016).

In fact, such systems face substantial security challenges because they usually operate in physically unprotected environments (Nimgaonkar, Gomathisankaran & Mohanty 2013a, 2013b; Sridhar, Hahn & Govindarasu 2012). Embedded System Tampering (EST) is a latent threat that can be implemented by using certain tools to inject tampered/malicious codes into systems. Attackers often exploit potential vulnerabilities in a targeted system in order to gain unauthorized access and then utilize the system or obtain the system's data illegally. Basically, those potential vulnerabilities can be exploited in any ES that is not well designed and does not include anti-tamper techniques/algorithms. Mostly, attackers aim to acquire control over some features of the software through an illegal alteration of the executable code and behaviour (Dalai, Panigrahy & Jena 2012; Stamp 2006).

1.3 PROBLEM STATEMENT

Software tampering is not limited to certain countries; cases have occurred in Asia and in Europe too. One real-world example is the malpractice that was detected in a retail fuel outlet in Malaysia. A petrol station in Silibin, Ipoh was sealed after law enforcers found that the administrator had manipulated all of its 26 fuel pumps to make more profit illegally. The metrology expert found that the petrol gauges at the pumps had been adjusted with dubious readings (Ibrahim et al. 2015; Kaur 2013). Similar cases have been reported in India (Anand 2013). Another real-world example comes from Germany, where the authorities found a number of illegal manipulation cases had been conducted against the recorded data on modern cash registers (Reckendorf et al. 2010).

Undoubtedly, authorized persons and organizations are aware of such illegal manipulations, so they need an applicable technique and procedure to combat such acts. Hence, tampering detection is gaining more attention and being given higher priority by ES designers and developers (Babar et al. 2011; Santucci 2010). Checking code integrity achieves tamper-proofing by identifying unauthorized alterations and recognizing whether any tampered code is being executed or any tampered data are being used. Such techniques/tools do not prevent theft but instead discourage software tampering. Thus, unless appropriate techniques/tools are used to detect the code integrity of targeted devices, not only customers will be lost, there may also be undesirable impacts.

Despite earlier studies, the integrity verification of remote ESs remains a vibrant research topic (Brasser et al. 2016). Since most (if not all) of ESs usually operate outdoors in physically unprotected environments, users need to conduct integrity verification remotely. Hence, a trustworthy mechanism of system verification is one of the main principles for remote verification, and the existence of a secure protocol based on cryptography to secure communications from attackers is mandatory. Coker et al. (2011) emphasize that it is especially challenging to satisfy the trustworthy principle where data are transmitted over a public network. Thus, the ordinary approach to handling this challenge is to utilize cryptographic primitives.

On the other hand, the early study of the system under development stimulates design decisions for further enhancement. Thus, obtaining early feedback on simulation models of system processes helps in gaining a better understanding of the behaviour of the developed system under more realistic settings. As a result, the ideas that system designers can obtain through prototypes may lead to refined protocols and algorithms, and thus contribute to the whole system design process (Richerzhagen et al. 2015). In fact, many "researchers have found the use and importance of prototypes to be substantial" (Elverum, & Welo 2014, p. 492).

Although it is recognized that the security of cryptographic primitives is guaranteed by the use of state-of-the-art cryptographic algorithms (such as the Rivest-Shamir-Adleman (RSA) public key cryptosystem and Secure Hash Algorithm 2 (SHA-

2), the security of the remaining aspects must be carefully analysed (Basile, Di Carlo & Scionti 2012). Thus, to develop high-integrity systems in which security attributes are important, the researcher needs to prove the defined secrecy features formally (Gargantini, Riccobene & Scandurra 2009, 2010). Indeed, a formal method approach assists designers and users to analyse and verify the proposed system at any point in the system life cycle (Woodcock et al. 2009). Therefore, this research proposes an efficient framework and a secure protocol to verify the integrity of an ES's code and evaluates the proposed protocol in a formal way.

1.4 RESEARCH QUESTIONS

This research intends to address the above-stated problems through the actualization of the thesis objectives, so the following research questions have been defined:

RQ1: What components are required in order to design a framework for code integrity verification in an Embedded System?

RQ2: How can basic cryptographic primitives be utilized to design a protocol to secure the data exchanged when conducting remote-code integrity verification?

RQ3: How can the code integrity verification system be simulated and tested against an Embedded System?

RQ4: Are the secrecy and authentication features of the proposed artifact robust and secure?

1.5 RESEARCH OBJECTIVES

To answer the above research questions, this study attempts to achieve the following objectives:

i. To propose a framework to verify the code integrity of an Embedded System and identify its components;

ii. To design a secure protocol for remote-code integrity verification of Embedded Systems;

iii. To develop a prototype for detecting software tampering in an Embedded System and test it against a simulated measurement instrument; and

iv. To evaluate the secrecy and authentication specifications of the proposed protocol and the entire framework using a formal method approach and an expert evaluation, respectively.

1.6 SCOPE OF THE RESEARCH

This study focuses on proposing a framework with a secure protocol for ES integrity verification in order to help potential users verify the system code integrity remotely. The framework is divided into two modules: (i) registering and scanning the system code at the server site and (ii) validating the system code at the remote site. The framework is based on the literature review and the outcomes of the conducted feasibility study.

For the purpose of the evaluation process in this research, a digital measurement prototype is designed using an open-source hardware platform (Arduino Uno board), while the code integrity verification system is demonstrated in Java language using Eclipse software (Kepler Service Release 2). In addition, a Communication Sequential Processes (CSP) formal method approach, the Compiler for the Analysis of Security Protocols (Casper), and the model checker Failures Divergence Refinement (FDR) described in Lowe et al. (2009), Roscoe (2010) and Ryan et al. (2010) are used to model and analyse the secrecy and authentication specifications of the proposed protocol.

1.7 DESIGN SCIENCE RESEARCH METHODOLOGY

A recognized common framework is essential for design science research studies in the field of Information Systems (ISs) and a rational model or template is required so that readers and commentators can identify and assess the results of such studies. Researchers propose a Design Science Research Methodology (DSRM) for the production and presentation of design science research on IS. Figure 1.1 illustrates a typical DSRM process model.

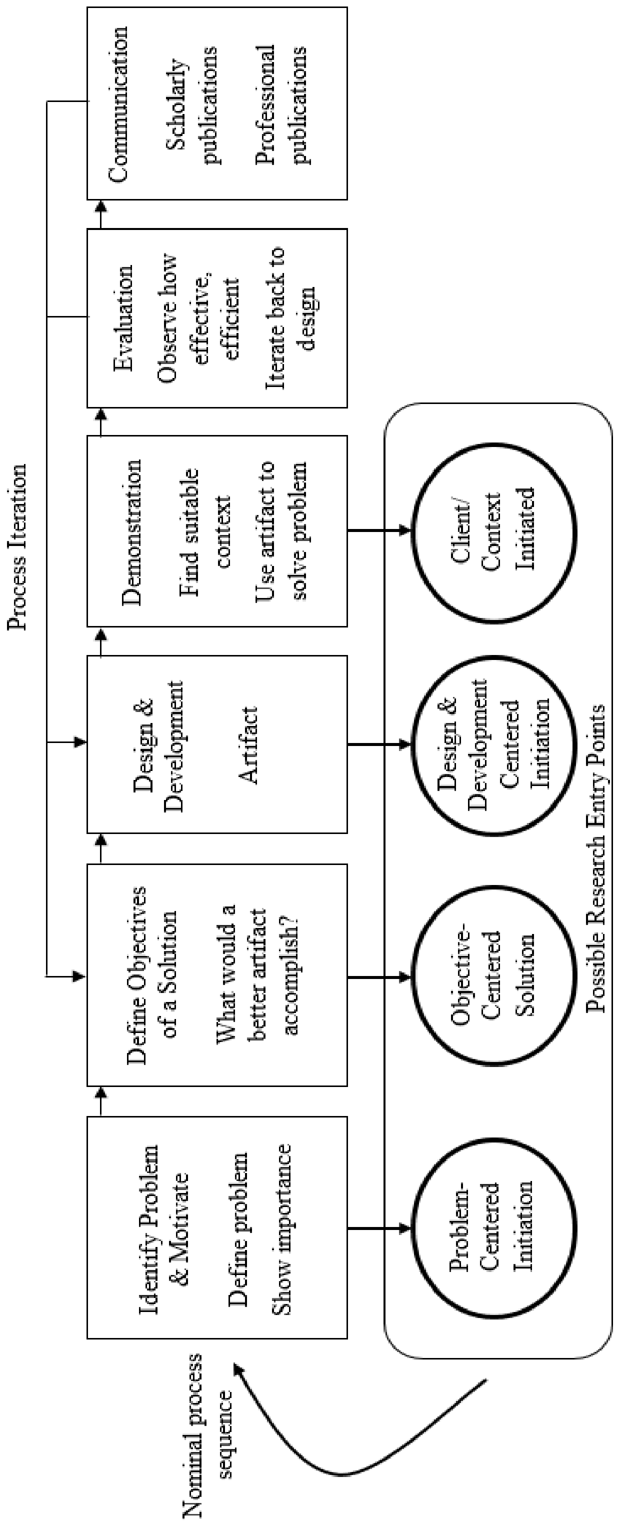
Process Iteration
Identify Problem & Motivate
Define problem
Show importance
Define Objectives of a Solution
What would a better artifact accomplish?
Design & Development
Artifact
Demonstration
Find suitable context
Use artifact to solve problem
Evaluation
Observe how effective, efficient
Iterate back to design
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/ Context Initiated
Possible Research Entry Points
Nominal process sequence

Figure 1.1 DSRM process model

Source: (Peffers et al. 2007)

This methodology contributes to IS research by introducing a commonly accepted outline for effectively conducting design science research and a rational model for its presentation. It is inspired by the need to improve the environment through the proposal of innovative artifacts and novel processes for designing the proposed artifacts (A. Hevner & Chatterjee 2010a; B. Kuechler & Vaishnavi 2008; Peffers et al. 2007).

The Research Process Model

Researchers have identified six phases that need to be conducted during research in order to accomplish the development of a coherent methodology. These phases could be adopted to design the research methodology for this study. Figure 1.1 shows the process model that consists of six phases of activity in a nominal sequence as follows (A. Hevner & Chatterjee 2010a; A. R. Hevner et al. 2004; Peffers et al. 2007; W. L. Kuechler & Vaishnavi 2012):

i. **Problem identification and motivation**. An interesting problem could be defined by reference to multiple sources such as different fields of industry or related disciplines. As one of the aims of design science research is to propose technology-based solutions to defined business issues, this activity involves the gathering and analysis of related information. In fact, defining and representing problems in an actual application environment is the first step in conducting good design science research. The resources needed for this phase consist knowledge of the defined problem and the significance of its solution.

ii. **Define the objectives**. A number of researchers explicitly convert the defined problem into a list of research objectives or requirements. They derive the objectives of a proposed solution from the problem definition and knowledge of what is possible and feasible. These objectives could be quantitative, such as the terms in which a required solution would be better than current solutions. Or they could be qualitative objectives, such as a description of how a proposed artifact is projected to support solutions to addressed problems. The resources needed for this activity comprise knowledge of the state of the defined problem and existing solutions, if available, and their usefulness.

iii. **Design and development**. The implementation of the proposed artifact is executed in this phase, and of course, implementation techniques vary depending on the proposed solution. This phase consists of determining the required functions of the proposed artifact and its architecture, and then creating the actual artifact. Also, constructing a formal proof would be required for an algorithm. Essentially, the implementation itself could be very ordinary and need not show innovation beyond the state-of-practice for the proposed solution; the novelty is mainly in the design and not the implementation of the artifact. The resources needed to achieve this phase include knowledge of theory/theories that could be utilized in a proposed solution.

iv. **Demonstration**. The use of the designed artifact to solve the defined problem needs to be demonstrated. This phase may involve using the proposed artifact in an experiment, simulation, case study, or other suitable activity. The resources required for this phase consist of effective knowledge for solving the defined problem by using the designed artifact.

v. **Evaluation**. To determine how well the designed artifact supports a solution to the defined problem the artifact needs to be evaluated. This phase could be achieved by comparing the research objectives defined in the second phase to actual observed results from use of the designed artifact in the demonstration. The resources needed for this phase include knowledge of related metrics and analysis methods. The evaluation process may take different forms based on the type of problem and the designed artifact. It may consist of a comparison of the functions of the designed artifact with the objectives defined in the second phase, outcomes of satisfaction surveys, client feedback, or simulations. Quantifiable measures of system performance such as response time or availability could also be included. Theoretically, an evaluation may consist of any proper empirical indication or logical proof. Then, if the defined objectives need to be refined or the design of the artifact needs to be improved, the researchers may iterate back to the second or to the third phase, respectively. Otherwise, the researchers may decide to continue on to the communication phase.

vi. **Communication**. The defined problem and its significance, the proposed artifact and its design, and the efficacy of the conducted research should be published in scholarly and/or professional publications for researchers and other related audiences. Presenting the research to a technology-oriented audience, with appropriate detail, facilitates the implementation of the designed artifact and its utilization within a targeted field. Besides, communicating the research outcomes helps the targeted audience to construct a cumulative knowledge base for additional improvement and assessment. This phase involve knowledge of the disciplinary culture.

1.8 STRUCTURE OF THE THESIS

This research thesis consists of six chapters that contain a number of sections and subsections. The thesis begins with Chapter 1, the introduction chapter, which introduces the background to the study and the research problem, the research questions, the main objectives of the study, the scope of the research, the adopted DSRM, and an overview of the structure of the thesis.

Chapter 2 provides a review of the research work related to the study scope. It consists of eight sections. The first five sections focus on the related definitions and concepts, some reasons and potential threats for conducting system tampering, and effective techniques for detecting tampering. Then, the sixth section shows the outcomes of the conducted systematic review on selected studies published between 2008 and 2016 that are related to software tampering in ESs, while the seventh section introduces two real-world projects.

Chapter 3 presents the DSRM for the research. In particular, the third section applies the formal method approach on a given example in order to explain how to utilize the approach for analysing the secrecy and authentication features of the tested protocol.

Chapter 4 describes the conceptual framework and the proposed system architecture and provides an explanation of its components in the second and third section, respectively. The fourth section of the chapter describes the remote integrity verification protocol proposed by the study. After that, section five discusses other functions for the law enforcer that are highlighted in the conceptual framework, and section six lists the security features of the proposed artifacts.

Chapter 5, the demonstration and evaluation chapter, presents the potential attack model related to the proposed solution and then demonstrates the designed prototype and the results of applying the CSP formal method approach to validate the proposed solution. It then discusses the results of an evaluation of the entire framework by five experts.

Finally, Chapter 6 provides the conclusion to this research. The chapter indicates how the research objectives were achieved, highlights the main contributions of the research, and then makes some suggestions for future research.

CHAPTER II

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter provides a review of a number of previous studies on the topic of interest. There are nine sections in this chapter including this introduction section. The second section identifies the main components of system security, while the third section explains some of the goals of an adversary when conducting attack(s). Section four introduces the latent security threats that may lead to system tampering. Then section five introduces varieties of techniques for detecting a system tampering. Section six consists of three subsections that explain the modelling and analysis of protocol specifications using the formal method approach and the testing of the protocol in Casper and the FDR model checker. Section seven presents the results of the systematic literature review, and the results of the conducted processes are provided in five subsections. Then, section eight briefly describes a number of real-world examples of system tampering, while the last section summarizes the chapter.

2.2 RELATED DEFINITIONS AND CONCEPTS

Goel (2010, p. 285) defines a threat, vulnerability, and a security attack as follows: "A threat is a potential violation of security and causes harm. A threat can be a malicious program, a natural disaster, or a thief. Vulnerability is a weakness of system that is left unprotected. Systems that are vulnerable are exposed to threats. Threat is a possible danger that might exploit vulnerability; the actions that cause it to occur are the security attacks."

For example, if a developer creates a database with a default user name and password, it is vulnerable to being accessed (accidentally triggered or intentionally exploited); an unauthorized user (either locally or remotely) might exploit that security flaw and be a security threat; and an intruder that compromises the system and logs into the database constitutes a security attack.

On the other hand, defining the security aspects of an ES is related to the main principles of information security: integrity, availability, and confidentiality; to ensure that only authenticated and legal entities are able to reliably access secure information (Davis & Clark 2011). Integrity means that properties can be altered only by authorized parties or by an authorized means and relates to data, software and hardware (Zissis & Lekkas 2012). While system integrity has been defined as “assures that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system” (Stallings 2014, p. 10), trust is negatively affected by the exposed vulnerability of systems that have been promoted as reliable and secure (Lippert & Swiercz 2005). As regards the ES, the system supplier/operator guarantees the user of its integrity by providing a reliable system in terms of ensuring that any modifications or enhancements (if necessary) are made by parties of sufficient authority. Therefore, the user is given an assurance that the system can be trusted for use and does not have any unauthorized modifications. Availability refers to ensuring the system/data will be accessible when an authorized user needs it, and without improper delay or undue halting. Confidentiality is about preventing the unauthorized reading of data and programs (Al-Wosabi & Shukur 2015; Rogers & Milenković 2009).

2.2.1 Symmetric and Asymmetric Cryptography

The most known and applied types of cipher systems are symmetric and asymmetric. In symmetric cipher system, it is easy to infer the decryption key from the encryption key. In practice, for the encryption and decrypting key are often identical. In fact, the symmetric key value needs to be distributed between the sender and the receiver before exchanging the secret messages. Thus, ensuring adequate protection for those keys must be estimated and managed well. In fact, key management that includes key generation, distribution, storage, change, and destruction, is considered as one of the ultimate difficult aspects of achieving a secure system (Chandra et al. 2014; Kumar, Munjal & Sharma 2011; Tripathi & Agrawal 2014).

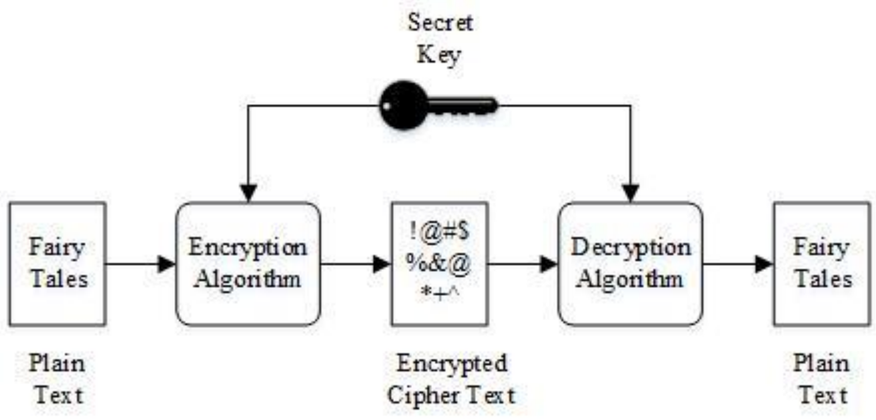

Figure 2.1 Symmetric key cryptography
Source: (Chandra et al. 2014)

However, in asymmetric cipher system, it is practically impossible to infer the decryption key from the encryption key, then the system is called asymmetric (It is also called a public key cryptosystem). In asymmetric cipher system, knowledge of the encryption key is of no practical use to the interceptor. In fact, this key usually made public and thus there is no need to share the encryption key between the sender and the receiver since it is not secret. Hence, there is no need to ensure mutual trust between the two parties (Chandra et al. 2014; Kumar et al. 2011; Tripathi & Agrawal 2014).

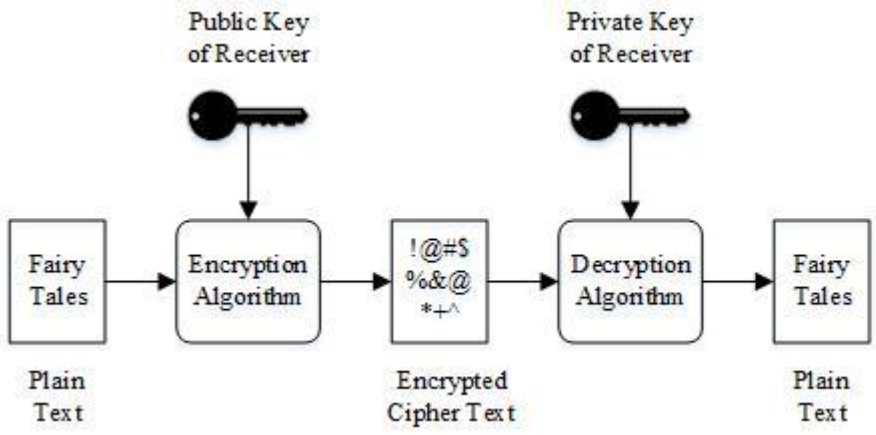

Figure 2.2 Asymmetric key cryptography
Source: (Chandra et al. 2014)

For example, those types of keys are used in many security applications and more precisely in a two-level key hierarchy system, such as Pretty Good Privacy (PGP). In such system, symmetric keys (also called session keys) are utilized to encrypt data transferred, and asymmetric keys are utilized to encrypt those symmetric keys. In fact, PGP has varieties of usages including protecting email and files. Moreover, the most well-known symmetric block cipher is Data Encryption Standard (DES), and RSA public key cryptosystem is considered as the most well-known asymmetric block cipher (Chandra et al. 2014).

2.2.2 Hash Functions

The main purpose of using hash functions is that the calculated hash value is a condensed representative image of the original text. The hash value is known as a digital fingerprint, a message digest, or a hash. In fact, hash functions usually accept input messages of arbitrary lengths and generate hash values of fixed lengths. If two input messages result in identical hash values then a collision has occurred. Hash functions widely applied in many cryptography applications, such as integrity verification and as part of the digital signature process.

One of the essential uses of hash functions is a provision of data integrity. Firstly, the hash value corresponding to a targeted data d is calculated at first time t1. The calculated hash value (not the original data) is protected in a certain way. At a later time t2, an integrity verification is executed to decide whether the original data has been modified, i.e., whether a data d' is identical to the original data. The hash value of d' is calculated and then compared to the previously calculated hash value; when the two values are equal, one accepts that the inputs are also equal, and thus that data has not been modified.

In asymmetric cipher system, a hash value of the message is calculated by utilizing a hash function to that message. Then the digital signature is generated from the hash value, which represents the message, by using the asymmetric algorithm with the secrete (i.e., the private) key (as shown in Figure 2.3). Hence, only the owner of that secret key can produce the generated digital signature.

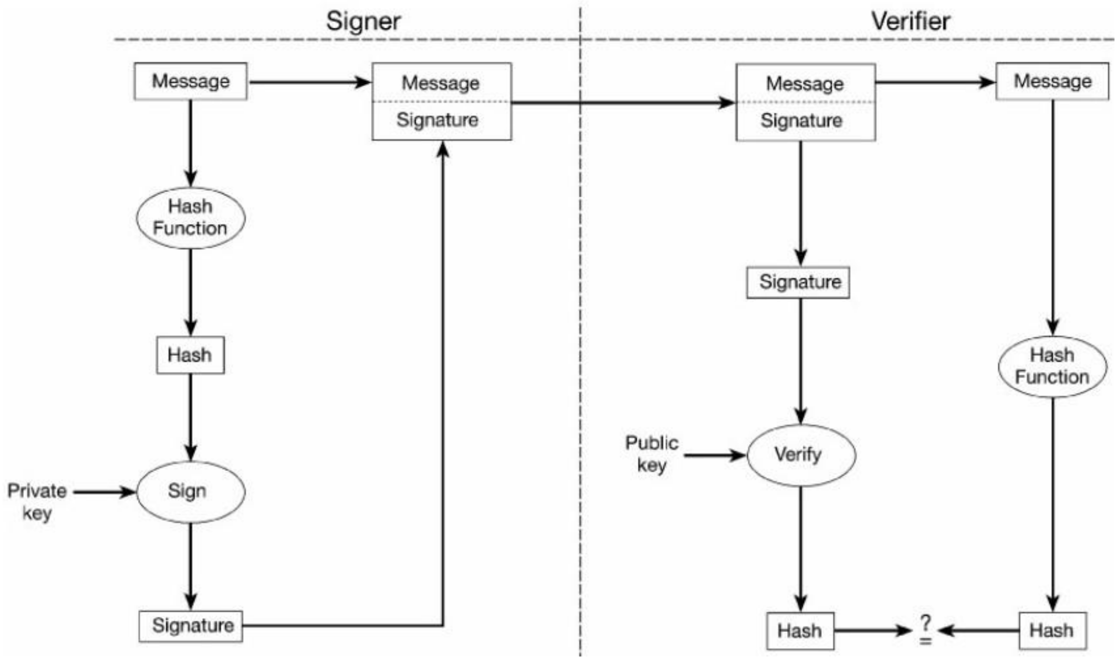

Figure 2.3 Hash function and digital signature process

Source: (Piper & Murphy 2002)

2.2.3 Message Authentication Code (MAC)

Unlike un-keyed hash functions, the keyed hash functions uses a secret key. The keyed hash functions whose specific purpose is message authentication are referred to as message authentication code (MAC). Certainly, in order to verify the sender of the received message, hash functions can also be applied with other cryptographic methods. Combining hash algorithms with encryption method can produce special value called message digest (or called message hash value) that identify the sender of the received message. Those special digests are called MACs (Mouha et al. 2014). There are three main types of MAC algorithms based on their underlying building blocks (Nandi 2009):

Hash MAC (HMAC): The HMAC algorithm has been standardized by the National Institute of Standards and Technology (NIST). It has been widely used by a number of secure internet protocols (such as some versions of IPSec) to produce authenticity of data. Moreover, this algorithm can prevent replay attack by adding a timestamp value to every message before sending it. Then, the receiver of that message will be able to verify that the message has not been previously received (Preneel 2010; Silva 2003).

Universal hash-based MAC: This type of MAC algorithm comprises of two building blocks; an effective keyed compression function that decreases long inputs to a fixed length and a method to process the short hash result and an output transformation. In practical structures, the encryption with the one-time pad is typically replaced by applying a pseudo-random function with a secret key (Handschuh & Preneel 2008).

CBC-based MAC algorithm: The most commonly used MAC algorithm based on a block cipher makes use of Cipher Block Chaining (CBC). However, it is widely known that the CBC MAC is not secure if the message length is not fixed due to the length extension attack (Iwata & Kurosawa 2003; Zhang et al. 2012). Thus, there are many different alternatives have been developed include OMAC, one-key CBC-MAC, or XCBC.

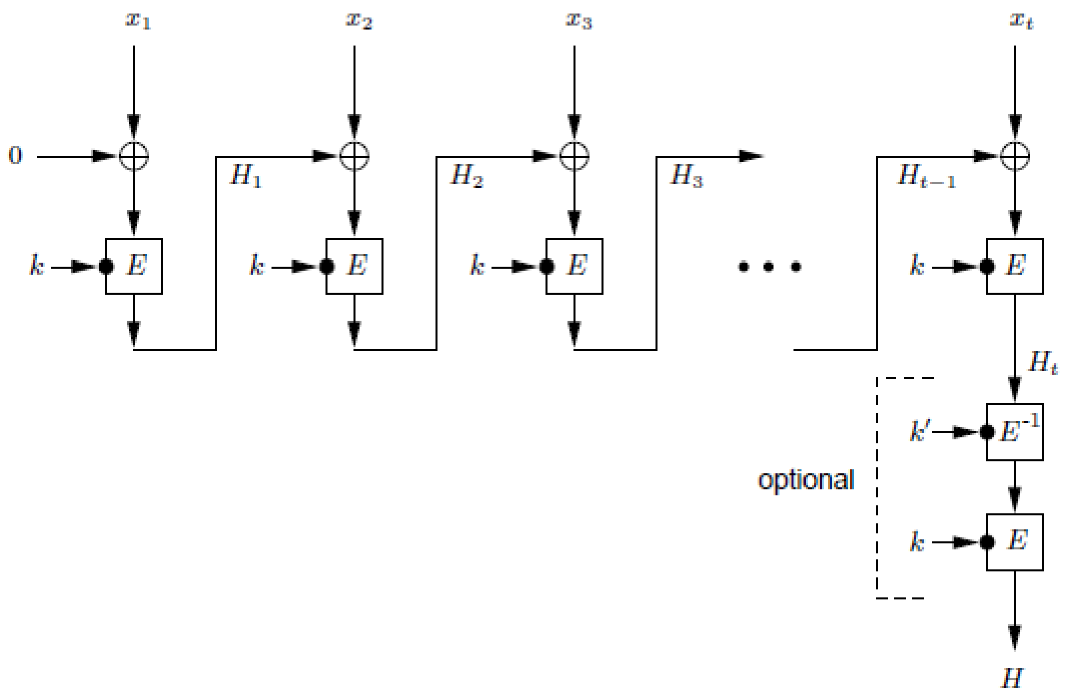

Figure 2.4 CBC-based MAC algorithm
Source: (A. J. Menezes, Oorschot & Vanstone 2001)

2.2.4 Nonce Value

A nonce value, which is often random number or pseudorandom number value, is a value that varies over a given time period that is non-foreseeable and thus can be used, along with encryption and/or hash functions, to restrict or deny unauthorized replay or re-execute of a connection (Ahlquist 2011). Informally, this type of function is a function that a computationally limited attacker cannot identify with likelihood substantially greater than half from a function chosen uniformly at random from all functions with a similar range and domain (Handschuh & Preneel 2008).

Usually, a random number generator used for generating nonces (Zenner 2009). Nonce values (i.e., random numbers) are used by a wide variety of network security algorithms and protocols, such as (Stallings 2014):

- Cryptographic keys distribution and mutual-authentication schemes. In such schemes, the sender and receiver participate by exchanging data to distribute cryptographic keys and mutual-authenticate each other. Hence, nonces values usually used to successfully execute handshaking to deny replay attacks. In fact, using the randomness function for the nonces values frustrates an attacker's efforts to identify or estimate the nonce value to regenerate an old transaction.

- Session key generation. A secret key for symmetric encryption is created and used for a single session (or transaction) and is valid for a limited period of time. This secret key is widely known as a session key.

- Key generation for the RSA public key cryptosystem. The essential definitions of security for public key cryptosystem provide no guarantees if the randomness is inadequate (Bellare & Tackmann 2016). For instance, generating a pair of keys (i.e., the private key and public key) requires determining two prime numbers. In order to prevent the discovery of those two numbers, those primes must be large numbers. Indeed, there are no useful methods that produce arbitrarily large primes and thus there is a need to manage this issue. The most widely used procedure is to pick at random an odd number of the desired order of magnitude and check if that number is prime. If not, then pick sequential random numbers till a desired prime number is found.

- Bit stream generation for symmetric stream cipher. With a well-designed nonce value generator, a stream cipher can be can be as secure as a block cipher of similar key size. In fact, stream ciphers, which do not utilize block ciphers as a building block, are typically faster and require quite smaller code than do block ciphers. For example, a developer can implement the Rivest Cipher 4 (RC4) algorithm by writing a few lines of code.

2.3 DIFFERENT REASONS FOR SYSTEM TAMPERING

In order to design an acceptable framework for software tampering detection, the adversary’s goals need to be identified. What do attackers hope to gain through the implementation of such illegal action(s)? Obviously, their purposes vary and typically include one or more of the following (Dalai, Panigrahy & Jena 2012; Ravi 2004; Stamp 2006):

i. **Unauthorized access/usage:** when an attacker illegally bypasses the predefined software access control privileges, the attacker can gain full/partial access to software functions. For example, an attacker may attempt to remove the software watermarks by tampering with the original code. However, if the original software is used to access classified data, then the attacker’s purpose is to gain unauthorized access to that data. In certain scenarios, such as in tampering with an e-commerce application, an attacker may aim to attack the application in order to gain unauthorized discount or free services.

ii. **Unlicensed clones:** this kind of illegal act is one of the most common causes of financial loss for many software companies, and especially because it leads to loss of competitive advantage for the original software developer. The attacker’s goal is to understand enough about the main functions of the targeted software. This type of attack can be achieved by simulating the software’s routines either for the purpose of accessing some of the intellectual properties, or for the purpose of reusing the software’s crucial processes in some other programs.

iii. **Attacking software integrity:** injecting a malicious code, such as a virus or malware, into the targeted software may lead to a breach in software integrity. The purpose of such an attack is to add to/modify software functions illegally. In fact, this type of attack can be made by an authorized or unauthorized user, and that is why the goal of a tampering detection model is not only to detect illegal modifications by unlicensed users, but also to detect illegal modifications by accredited users.

iv. **Disrupting system functions:** attacking the system's availability could cause system halting or an annoying delay in providing a normal service. This type of attack is usually executed by exhausting the system's resources and is popularly known as a denial-of-service attack (DoS attack).

2.4 LATENT SECURITY THREATS THAT MAY LEAD TO EST

Essentially, the aim of this research is to enhance ES security by facilitating tampering detection when a device is being utilized. Theoretically, there is, as yet, no such thing as a 100% secured system (Kim et al. 2009), but incorporating a verification system into an ES would (at the very least) increase the overhead efforts involved in conducting attacks on such systems.

There are a number of potential attacks that must be taken into account and more effort is needed to identify security tools/techniques that will enable the success of this framework (as shown in Figure 2.5). Actually, defining the latent threats is the first step in overcoming them. A number of researchers (Kim et al. 2009; Ravi et al. 2004; Sagstetter et al. 2013) have identified the following threats that may pose a security risk to ESs and that have to be taken into account:

i. **Reverse engineering:** the intention of such an attack is to reproduce (illegally) the ES functionality to make an unauthorized modification. The altered system then pretends to offer normal functionality while it is actually doing something else.

ii. **Complex design process:** there is no guarantee that an ES that is well designed will be free of malfunctions or malicious software. Also, in a complex design it may not be possible to handle a pre-validation process for each system component to ensure security. Furthermore, even if each part of a system is secure in itself, it is known that the composition of these parts may expose new vulnerabilities.

iii. **Malicious software:** this is the most widespread form of attack that may threaten any system, and it is the most easily and cheaply available to most attackers. There are different types of malicious software such as viruses, worms, Trojan horses, etc. Most of these agents exploit the latent vulnerabilities in a system's programming, such as buffer overflow which is a common loophole in operating systems and application software that can cause undesirable effects.

iv. **Wireless access:** while using wireless communication protocols like 3G or Bluetooth to facilitate execution of the system's service(s), potential/latent vulnerabilities have been demonstrated to exist for all of those protocols. For instance, exploiting a vulnerability in 3G to gain access to the software in an ES may allow hackers to make an illegal modification or may cause the system to be halted.

v. **Physical access:** introducing physical access, such as using a USB port for accessing the system, introduces another threat as an attacker could use it to make an unauthorized modification/alteration to the software.

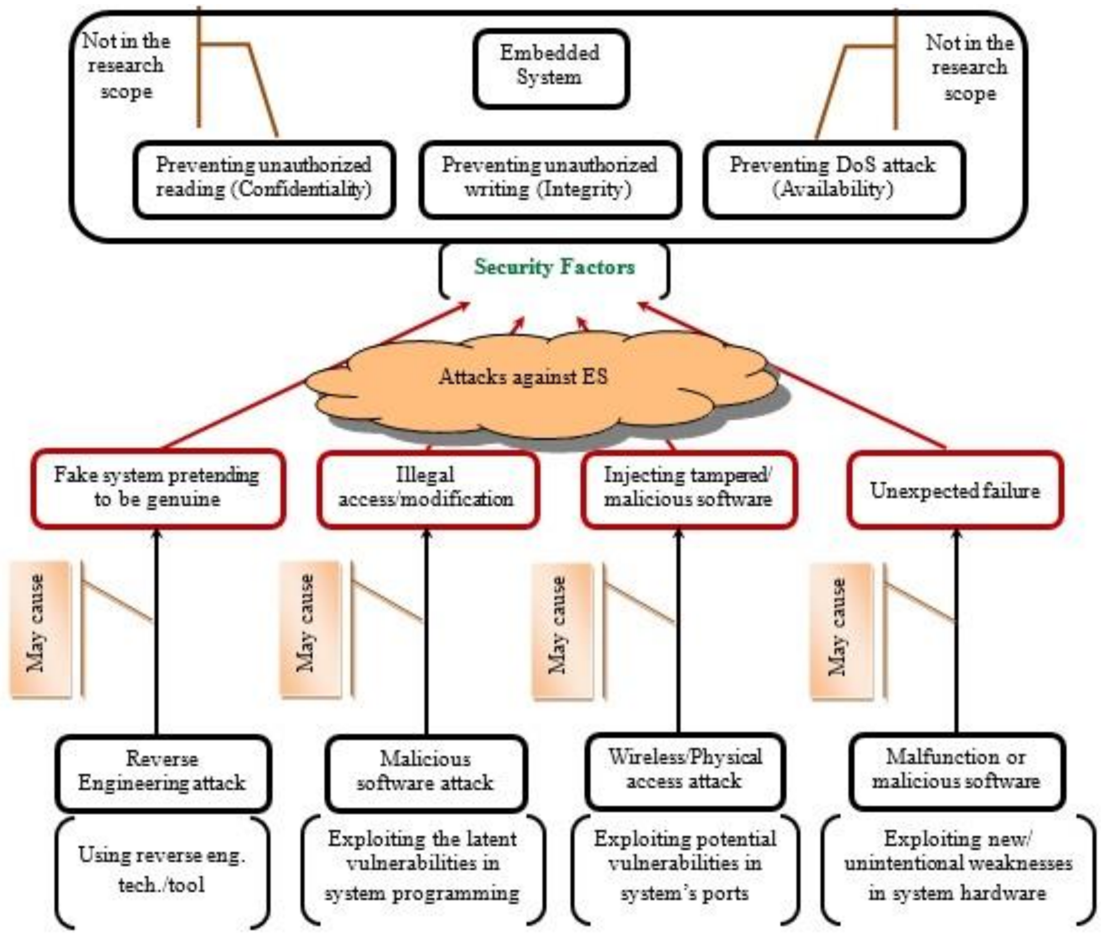
Not in the research scope
Embedded System
Not in the research scope
Preventing unauthorized reading (Confidentiality)
Preventing unauthorized writing (Integrity)
Preventing DoS attack (Availability)
Security Factors
Attacks against ES
Fake system pretending to be genuine
Illegal access/modification
Injecting tampered/ malicious software
Unexpected failure
May cause
May cause
May cause
May cause
Reverse Engineering attack
Malicious software attack
Wireless/Physical access attack
Malfunction or malicious software
Using reverse eng. tech./tool
Exploiting the latent vulnerabilities in system programming
Exploiting potential vulnerabilities in system's ports
Exploiting new/ unintentional weaknesses in system hardware

Figure 2.5 Potential vulnerabilities in and threats to an ES

2.5 TECHNIQUES FOR TAMPERING DETECTION

In their books, Christian S. Collberg, and Jasvir Nagra (2009), and Hossein Bidgoli (2006) identify a number of feasible techniques for protecting the intellectual property of software, including:

i. **Obfuscation:** the specific purpose of the process is to increase the burden and duration of reverse engineering that is aimed at targeted software. It is performed by rewriting (transforming) the original code into an obfuscated version with the same observable behaviour. There are three ways of achieving code obfuscation:

   a. **Lexical transformation:** the code's lexical structure is changed to annoy the reverse engineer. For example, by hiding the source code formatting information that is sometimes available from the Java bytecode or by scrambling identifier names.

b. **Control transformation:** the program's control flow is obscured. For example, by using more complicated but equivalent loop conditions, and implementing multithreaded instead of sequential processes.

c. **Data transformation:** this kind of obfuscation can be implemented by obscuring the data structure. For example, splitting variables, and changing the encoding of logical values True and False into Boolean variables.

ii. **Watermarking:** a secret message can be embedded into software in order to prove ownership without adversely affecting software performance. Also, it must be detectable by the original developer. There are two types of software watermark:

a. **Static watermarks:** the watermarks are embedded into the application executable itself.

b. **Dynamic watermarks:** the watermarks are constructed at runtime and stored in the dynamic state of the program.

iii. **Tamper-proofing:** to detect software tampering, tamper-proofing code can be added to the targeted software. This technique can be implemented by examining the integrity value of software code using certain functions to ensure that it is identical to the original code (Kaczmarek & Wrobel 2014). The most popular techniques to achieve integrity include Checksum, Cyclic Redundancy Check (CRC), Message Authentication Code (MAC), and Message Integrity Code (MIC) (Bishop 2005; Piper & Murphy 2002). Furthermore, the cryptographic hash function is one of the possible ways to generate MACs, also called digest values (as shown in Figure 2.6). The use of a hash function like Message Digest algorithm 5 (MD5) or the Secure Hash Algorithm (SHA) has become a standard approach in many applications and protocols (Dalai et al. 2012; Myles & Jin 2010).

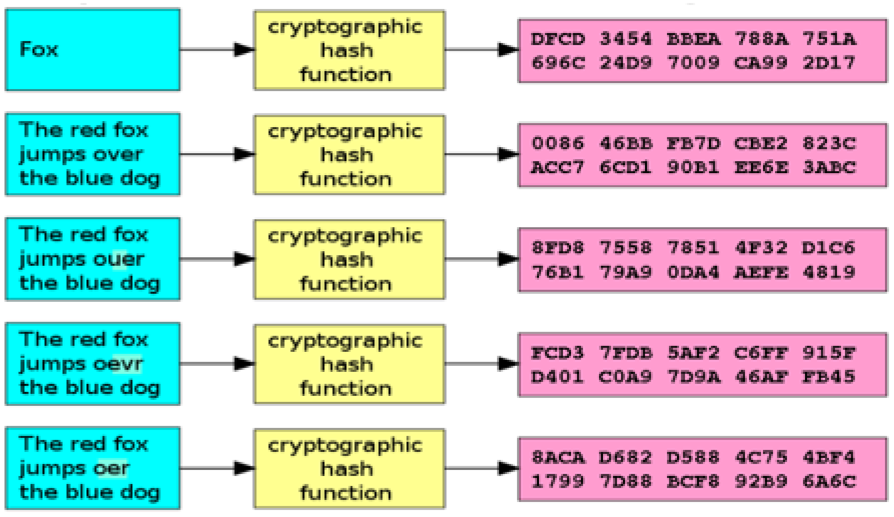
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C

Figure 2.6 Example of hash function

Source: (Donohue 2014)

2.6 SYSTEMATIC LITERATURE REVIEW

The aim of this section is to review the current state of the art on software tampering in ESs. This research applied the following processes based on the systematic literature reviews mentioned in a related technical report (Kitchenham et al. 2009; Kitchenham & Charters 2007). The next subsections summarize the research questions, search process, inclusion and exclusion criteria, bibliography management and document retrieval, and data extraction and analysis.

2.6.1 Research Questions

The research questions constitute one of the key factors to consider in any systematic literature review (Kitchenham et al. 2009; Kitchenham & Charters 2007). For the purpose of this research, four research questions were defined (see Chapter 1, section 1.4).

2.6.2 The Search Process

Essentially, this research started by implementing a literature search for related studies, and a number of research studies/papers were found using Universiti Kebangsaan

Malaysia (UKM) Online Library and Internet services. However, the search process continued until the end of this study. This process was conducted by using the search engines of several digital libraries, such as:

i. IEEE Xplore
ii. ACM Digital Library
iii. Scopus
iv. Science Direct
v. Springer Link
vi. Google Scholar.

Key terms that are closely related to this research project are: “Software tampering”, “Code integrity”, “Anti-tamper techniques or tools”, “Tampering detection”, “Integrity verification”, and “Embedded system”.

2.6.3 Inclusion and Exclusion Criteria

For the purpose of conducting this review, a number of criteria were defined to specify those studies to be included and those to be ignored/excluded. The following inclusion criteria were applied:

i. Research studies published between 2008 and 2016 that relate to software tampering in ESs;
ii. Research studies on techniques/tools related to software tampering detection.

On the other hand, this review excluded certain studies that met the following criteria:

i. Informally published (not defined or unknown journal or conference);
ii. Irrelevant to the above research questions;

iii. Older duplicate versions of the same study (research). However, if the research paper was published in more than one journal or conference proceeding, then the most complete version was chosen.

2.6.4 Bibliography Management and Document Retrieval

Mendeley Desktop 1.17.10 was used to manage all the citations and bibliography in order to formulate the thesis report. The key terms defined above were used for searching the above-mentioned search engines. The selected studies that appeared in journal/conference publications were scanned using their title and abstract. All relevant papers were then downloaded for further assessment and data extraction.

However, a number of research studies that were gathered from digital libraries were duplications, so the older versions were neglected according to the exclusion criteria (as mentioned above). Also, scanning and skimming techniques were applied on the collected papers in order to capture the most closely related studies. Accordingly, a number of studies were considered for further review.

2.6.5 Data Extraction and Analysis

This section involves collating and summarising the results of the included primary studies, which could be represented in descriptive (non-quantitative) way (Kitchenham et al. 2009; Kitchenham & Charters 2007).

a. Attack Model and Proposed Solutions

It is essential to introduce an attack model to discover the most convenient techniques for securing an ES. In general, an ES can be exposed to two types of attack that relate to access to the ES: remote attacks and physical attacks (Gelbart et al. 2009). Some other researchers (Babar et al. 2011; Rogers & Milenkovic 2009) summarize attacks against ESs as follows:

i. **Physical attacks:** involve direct tampering with hardware components, and include spoofing attacks, splicing attacks, and replay attacks.

ii. **Side-channel attacks:** attempt to indirectly capture secure data based on side-channel information from the system's operations, and include timing attacks, power analysis, and fault analysis attacks.

iii. **Software attacks:** exploit potential vulnerabilities (like buffer overflow attacks) in many software, or inject malicious code (like Trojan horse programs or viruses) in order to overwrite data on system memory or cause the processor to execute an unordered or malicious section(s) of code.

iv. **Network attacks:** exploit potential vulnerabilities in the transmission medium. These can be classified into active attacks (e.g., DoS attacks), or passive attacks (e.g., monitoring and eavesdropping, and traffic analysis).

Mainly, there is a lack of joined-up effort to establish security in the development methodology of information technology frameworks, which is a consequence of various elements that are related to the development process, or the environment in which the framework works. Embedded security challenges may include heterogeneity, complexity, adaptability, decentralized control, time-to-market pressures, performance, energy efficiency (power consumption), and security cost (Babar et al. 2011; Mirjalili & Lenstra 2008).

Hence, those elements may make designers hesitant to concentrate on information security accurately in the early phases of system development as it is considered as an exercise in futility. Mirjalili and Lenstra (2008) define a four-level security strategy in order to overcome those challenges. The proposed strategy consists of:

i. Preventing the event or presentation of vulnerabilities by enhancement of design and development processes;

ii. Applying different tolerance methods, for example, vulnerability recognition, attack recovery, and self-adaptive procedures;

iii. Eliminating vulnerability during the development stage and the utilization stage; and

iv. Predicting vulnerability, which is accomplished by conducting a system assessment with respect to attack occurrence.

Furthermore, Babar et al. (2011) state that the main features of the security framework and architecture consist of lightweight cryptography, physical security, standardized security protocols, secure operating systems, future application areas, and secure storage.

A wide variety of research has been carried out on three main types of solution: hardware-level solutions, software-level solutions, and a combination thereof. For example, a solution can be implemented by incorporating a hardware system as an external checker/tester, or on the product level where a trusted part of the code exists to verify the targeted system's security. Table 2.1 summarizes the proposed solutions (software only approach, hardware-only approach, and hybrid approach) in relation to some of the mentioned issues and challenges (Babar et al. 2011).

Table 2.1 Comparison of existing solutions

Solution approach	**Issues (challenges) to be solved**				**Comments**
	Cost	**Flexibility**	**Performance**	**Power consumption**	
Software only	Yes	Yes	Partially No	No	Sometimes leads to the processing capacity of the Embedded System general-purpose processor being overwhelmed.
Hardware only	No	No	Yes	Yes	
Hybrid approach (Software and Hardware)	Partially Yes	Yes	Yes	Yes	Requires a clear vision of the complete system and good interaction between the hardware designers, the software designers, and the security experts.

b. Data and Software Security

As shown above, this section presented the views of a number of researchers on the threat model, ways in which to integrate security into the development methodology of the ES, and the main types of proposed solution. Some additional studies that relate to data and software security in ESs are reviewed below.

Firstly, Schwartz, Sie and Hallin (2009) suggest creating numerous hashes for executable software, the combination of which represents a signature of the whole executable software. Every individual hash corresponds to a particular part of the executable software (called a code segment) such that every fractional digest is a signature of less than all of the code bytes. When a request to load a code segment (e.g., a page or something else) of the code into the memory is made from a storage device, a verification hash of the code segment is calculated. Then the verification digest is contrasted with a fractional digest of the numerous hashes to verify the integrity of the code segment.

A solution based on analysing the real-time execution of a section of code is proposed by Zimmer et al. (2010). They utilize worst-case execution time (WCET) bounds data to recognize code tampering in real-time cyber-physical systems (CPSs) by instrumenting the tasks and schedulers to confirm timing analysis results in order to ensure that the execution time has not exceeded the expected time bounds.

Another research study (Venčkauskas et al. 2012) attempts to develop a secure mechanism for ensuring the software integrity of the ES that does not need a peripheral hardware and infrastructure for generating the security key, storage and management, and gives an adequate security level. Also, in order to save the code in an encrypted format, cryptographic keys are created in real time, on interest, before the execution of the encoded code module.

On the other hand, Rogers and Milenković (2009) outline the application of a Parallel Message Authentication Code (PMAC) algorithm that takes into account the utilization of a single hardware encryption module for both encryption and validation, hence it is system-resource wise and cost-effective. They utilize block cipher encryption as a part of their signature generation process. They introduce a mechanism for saving memory costs by securing various instructions and/or data blocks by calculating cryptographic signatures.

Also, a hardware monitoring solution has been applied by a number of researchers. For instance, Mao and Wolf (2010) present a checking system to verify proper software execution. Their proposal is focused on monitoring how the ES processor utilizes system resources that are discrete from the binary code, and it could initially be introduced for ESs in general and then for all targeted ESs. Since both encryption and validation are regularly computationally intensive, some authenticated-encryption algorithms are suggested.

In addition, a study has been conducted to design a remote verification system where the proposed system is not required to exist on the network. Remote verification needs a secure network protocol. For instance, Basile et al. (2012) add hardware-level components to externally verify system integrity. They concentrate on recognizing whether executed code has been altered by utilizing a field programmable gate array (FPGA) to construct a secure architecture. The researchers' objective is to design a system that makes it hard to conduct a successful real-world attack. However, the researchers do note that this security technique is not particularly suitable for high-security systems such as those operated by the military and government.

A hardware monitoring module is also proposed by Abad et al. (2013) as a means to validate a control flow graph when the system's components are running in real time. They develop a tool to generate a control flow graph of tasks running on the system processor. The proposed module has its own memory to load the generated graphs for further checking, and thus this may cause the processor to halt while executing small blocks.

Alternatively, Perrig et al. (2015) introduce a protocol that enables secure detection of and recovery from sensor node compromise. They claim that the protocol can deal with recovering sensor nodes after they have been compromised, and that the code update protocol can securely update the code of a sensor node, offering a strong guarantee that the node has been correctly patched, or detect when the patch failed. However, the assumption that attacker’s hardware devices are not present in the sensor network during the repair is the main limitation of their study.

Whenever a pure hardware solution or pure software solution fails, a combination of hardware and software solutions can be exploited. For example, Gelbart et al. (2009) propose a system that adopts a joint compiler-hardware approach to protect the software in the ES by encrypting data and code in the memory. They use FPGA to decrypt executables, and validate the code integrity before it is executed on the processor.

Additionally, Nimgaonkar et al. (2013a) introduce Memory Integrity Verification (MIV) to ensure data and code integrity. In their method, data and code are encrypted before they are inserted into the memory and are decrypted after they have been read. This prevents an attacker from observing or modifying the protected data/code. They consider energy efficiency and use the Merkle hash tree with timestamps and timestamp cache to reduce the energy consumption of the verification procedure. Kaczmarek and Wrobel (2014) have utilized cryptographic hash generation and verification to introduce an integrity checking and recovery system solution to increase computer system security by integrity the checking of files that are vital for system operation. Also, they suggest storing all of the essential data in physically write-protected storage to reduce the threat of illegal alteration.

Actually, information security and privacy has direct influences on the current smart metering infrastructure and on intelligent vehicles. So, a number of studies have been carried out to investigate issues related to these two fields. For instance, seals can increase the level of system integrity since they detect tampering when it occurs, and then protective action can be applied when the seal is tampered with. Ransom et al. (2010) present an invention that introduces suitable or applicable artifacts for securing related information while it is being formed (created), saved and transferred by an Energy Management (EM) device that is secured by a tamper detection seal unit that functions to detect illegal access to the EM device and specify any action needed. In one embodiment, the proposed system performs the following tasks: produces the information; categorizes the information according to its integrity; recognizes when the tamper detection seal unit has detected that an illegal modification has been executed; and secures the integrity of the information when an alteration has occurred (i.e., by applying the assigned protective action(s)).

At this point it is also worth mentioning the trusted computing base with secure storage and public key cryptography proposed in Garcia and Jacobs (2011). The researchers outline how multiparty processing units (local substations) can compute the sum of their energy consumption without revealing the user’s information. They argue that the current smart metering structure could be redesigned to include a trusted segment in the meter device instead of relying on a one-sided trust approach at the central station or local substation. This more versatile architecture where meter devices have their own trusted segment would provide a certain level of independence.

Furthermore, Kumari, Kelbert and Pretschner (2011) propose usage control mechanisms for information that has to be shared over the network by smart meters connected to online social websites. They suggest that the information sent to the user should be controlled by requesting that the user provide confirmation to the targeted provider that he/she has the required usage control mechanism present and activated on his/her system before the information is transferred.

In order to develop an applicable framework to protect code integrity against intentional tampering, Nilsson, Sun and Nakajima (2008) introduce a Secure Firmware updates Over The Air (SFOTA) protocol for intelligent vehicles in order to secure the transmission of the firmware code between the portal and the vehicle. The proposed framework facilitates code verification for firmware updates based on a simple hash chain calculation on memory contents, a challenge-response mechanism, and the inclusion of random numbers to prevent pre-image attacks. However, the key management for using and storing the encryption key is not considered well as they assume that a single cryptographic key is used for all the car’s control units.

Mobile phones are affected by similar issues. So, Chaugule, Xu and Zhu (2011), propose the Specification Based Intrusion Detection Framework (SBIDF) that exploits whether there are hardware interrupts in order to distinguish between activity that is purely programming initiated and activity that is human initiated. It characterizes specifications to identify the typical conduct pattern, and imposes this specification on all third-party applications on the mobile phone at runtime by observing the inter-component interface pattern among critical modules. At whatever point these critical modules start up for implementation, the Authentication Module calculates an MD5 hash over the TEXT portion of the module. Then, in order to establish the integrity of the critical module, the Authentication Module compares the hash with a pre-computed value of the hash of that segment. This pre-computed hash value is available in the Specification Database. It can be computed by the phone stack supplier before delivery to the client, and after that statically saved in the Specification Database for future utilization.

2.7 REAL-WORLD EXAMPLES

There exist a number of real-world projects on data and code security in ESs, including for instance, the EVITA project (Community et al. 2013) and INSIKA project (Reckendorf et al. 2010) that have been introduced and managed in European countries. The EVITA project (http://www.evita-project.org/) has introduced three different security modules to protect on-board communications in vehicles by applying a principle that prevents external car connections. The project uses a Hardware Security Module (HSM) that facilitates the means to secure platform safety, to guarantee the integrity and secrecy of significant items, and to improve cryptographic processes, thereby the securing crucial resources of the system. The HSM contains the following components: a Symmetric Cryptographic Engine, Asymmetric Cryptographic Engine, Hash Engine, Random Number Generator, and Secure CPU.

On the other hand, the INSIKA project (http://www.insika.de/en/), a German working group on cash registers funded by the German Federal Ministry of Economics and Technology, was established in 2008. The aim of this project is to introduce an applicable innovation for prohibiting information deception in Electronic Cash

Registers (ECRs). The main idea is based on using digital signatures to detect any illegal modifications to the protected information. The basic idea of this project is based on asymmetric cryptography (public and private key algorithms) and the SHA-1 algorithm.

2.8 CRITICAL ANALYSIS

Researchers have identified two main types of attack that may target an ES: remote attacks and physical attacks (Gelbart et al. 2009). This research focuses on physical attacks that may be conducted by manipulating certain functions or modules or their parameters of an attacked system. In addition, lack to integrate security into the development methodology of information technology frameworks from early phases may produce potential vulnerabilities (unintentionally) hidden within the developed system.

Several solutions have been proposed for tackling such attacks. For example, researchers have suggested creating numerous hashes for executable software (Schwartz et al. 2009) while using the timing information based on an analysis of the static worst-case execution time of a targeted system (Zimmer et al. 2010). Other studies have proposed developing a secure mechanism for ensuring system integrity (Venčkauskas et al. 2012), an algorithm that utilizes block cipher encryption and a cryptographic signature (Rogers & Milenkovic 2009), and monitoring the ES processor to verify that there is proper system execution (Mao & Wolf 2010). Meanwhile, Basile et al. (2012) have utilized FPGA to construct a secure architecture intended for low-security systems and Abad et al. (2013) have developed a tool to validate control flow while the targeted system is running.

However, a number of the above-mentioned solutions are not adaptable and may need reprogramming or additional hardware as in the case of Venčkauskas et al. (2015) or additional constraints as in the case of Basile et al. (2012) as the key allocation should be carefully considered in order to minimize collision. In addition, certain solutions may cause the processor to halt while executing small blocks because the developed system needs its own memory to load the generated graph, as in the case of Abad et al. (2013).

Even though the protocol introduced by Perrig et al. (2015) enables secure detection and deals with recovering sensor nodes after a compromise, the assumption that the attacker’s hardware devices are not present in the sensor network during the repair is the main research issue outstanding in their work.

Despite the different types of technique that have been invented to help in implementing system integrity verification, the utilization of hash functions is still an effective solution in many proposals (Chaugule et al. 2011; Nimgaonkar et al. 2013a, 2013b; Schwartz et al. 2009). However, there are certain weaknesses in some of the proposed designs, such as the use of a single cryptographic key as in the case of Nilsson et al. (2008). Besides, a number of researchers’ proposals were based on hardware solutions such as using an on-chip FPGA-based hardware component (Gelbart et al. 2009), or dedicated storage (Kaczmarek & Wrobel 2014).

Nevertheless, hash functions have been useful in developing many applications and protocols, and they have been used as a measurement agent in the Trusted Computing Group (Almohri, Yao & Kafura 2014; Brasser et al. 2016; Coker et al. 2011; Dalai et al. 2012; Ferguson et al. 2010; Myles & Jin 2010; Perrig et al. 2015), and in real-world projects such as the EVITA (Community et al. 2013) and INSIKA (Reckendorf et al. 2010) projects.

However, conducting integrity verification of remote systems needs a secure protocol that is well designed, and thus analysing and evaluating the secrecy and authentication features of such protocols is essential (Basile et al. 2012; Gargantini et al. 2009, 2010; Woodcock et al. 2009).

2.9 MODELLING AND EVALUATING THE SECURITY PROTOCOLS USING THE FORMAL METHOD APPROACH

The era of advanced communications has emerged, increasing the use of digital devices to handle many online transactions, such as online banking, distance education, online shopping, etc. The protocols used in such systems play an essential role in gaining user confidence. Certainly, protocol verification is a major activity during system development, and such verification protocols should be robust and secure (Shaikh & Devane 2010) . Hence, in this research, the secrecy and authentication specifications of the proposed protocol need to be verified.

There are two main methods that can be used for analysing security protocols: formal models and computational models (Blanchet 2012; Cortier, Kremer & Warinschi 2011; Pironti, A. Pozza, D. & Sisto 2011). While formal models are very precise and accurate in presenting system specifications and permit straightforward and proficient reasoning about the properties of the security protocols, they are not widely used. System analysts find that generating formal models is too complicated since they are not familiar with mathematical notations (Shukur et al. 2006; Sullabi & Shukur 2008). Also, the algebraic perspective of cryptography depends on accurate encryption axioms (Ryan et al. 2010):

i. The only way to decode ciphered information is to know the corresponding key;

ii. Ciphered data do not uncover the key that was used to cipher them;

iii. There is adequate redundancy in ciphered data, so that the decoding algorithm can distinguish whether a cipher text was encoded with the normal key;

iv. There is no way to extract original data from a hashed value;

v. Different data are constantly hashed to various hash values;

vi. Freshly calculated values are constantly different from any existing value and not guessable; and

vii. A public key does not uncover its secure key.

On the other hand, computational models represent data as bit-strings and algorithms executed on Turing machines. They utilize a probabilistic approach to permit part of the precise encryption presumptions to be declined. However, they involve more effort during analysis, and typically the proofs are more hard to program (Blanchet 2012).

A number of researchers have built prototype tools specifically tailored for verifying security protocols, such as the Naval Research Laboratory (NRL) Protocol Analyser, on-the-fly model checker (OFMC), and many others. Other researchers have used general model checkers, such as Failures Divergence Refinement (FDR), and have shown how this tool could be used to describe and analyse security protocols (Lowe et al. 2009; Pironti, Pozza & Sisto 2011; Ryan et al. 2010; Shaikh & Devane 2010). Note that FDR is the de-facto model checker for the CSP approach, which has been developed and refined over a number of years (Bartels & Kleine 2011; Kleine & Göthel 2010; Sun, Liu & Dong 2008).

2.10 CONCLUSION

In light of the above, the following can be concluded. Firstly, nowadays, the ES can be considered a core entity for many digital devices and therefore the digital world. However, in many application domains, ESs are facing an increase in security attacks such as system tampering. In order to counter this threat, integrity verification has been explicitly proposed by many researchers. Yet, system tampering remains one of main challenges that must be resolved by ES designers and developers.

Secondly, there are two main types of attack: remote attacks and physical attacks, and three main types of proposed solution: hardware-level solutions, software-level solutions, and combinations of the two.

Thirdly, the hash function has been used in proposed solutions for a long time by many studies, and it is still an effective solution for system integrity verification (Chaugule et al. 2011; Community et al. 2013; Gelbart et al. 2009; Kaczmarek & Wrobel 2014; Nilsson et al. 2008; Nimgaonkar et al. 2013a, 2013b; Reckendorf et al. 2010; Schwartz et al. 2009).

Fourthly, the protocols used in many systems play an essential role in gaining user confidence. Certainly, protocol verification is a major activity during system development, and such verification protocols should be robust and secure.

Fifthly, while formal models are very precise and accurate in presenting system specifications and permit straightforward and proficient reasoning about the properties of security protocols, they are not widely used.

Thus, through this review of previous research, it can be concluded that, despite the progress made in previous works, it remains important to find solutions for tampering detection and to proceed in this research to contribute (even in a simple manner) to solving this problem by proposing an integrity verification framework. The following chapters aim to achieve the objectives of this research, which include the proposal of a system architecture with a secure protocol.

CHAPTER III

METHODOLOGY

3.1 INTRODUCTION

The previous chapter discussed the literature review for the study. This chapter consists of four sections. The chapter begins with this introduction to the chapter and is followed by section two that introduces the six main research phases conducted for the study. Then, section three explains the formal method approach and the use of Casper and the FDR model checker for representing the security protocols and examining their secrecy and authentication features. Finally, section four concludes the chapter.

3.2 USE OF DESIGN SCIENCE RESEARCH METHODOLOGY FOR THE RESEARCH

Information system researchers have adopted an applied discipline by which to conduct their studies. They have used DSRM to incorporate the principles, practices, and procedures needed to create effective artifacts. It serves as an accepted framework for design science research and as a template for presenting the related knowledge. Moreover, it can guide a researcher to create an appropriate solution to a problem when using a conventional research model. Such a framework is necessary for presenting the proposed artifact as a conceptual model, which helps the targeted audience to recognize and evaluate the outcomes of the conducted research (A. Hevner & Chatterjee 2010a; Geerts 2011; Peffers et al. 2007). Hence, DSRM is adopted in this study in order to design the research methodology that acts as guidance for conducting this research effectively. Figure 3.1 shows the process model that consists of six DSRM phases in a nominal sequence.

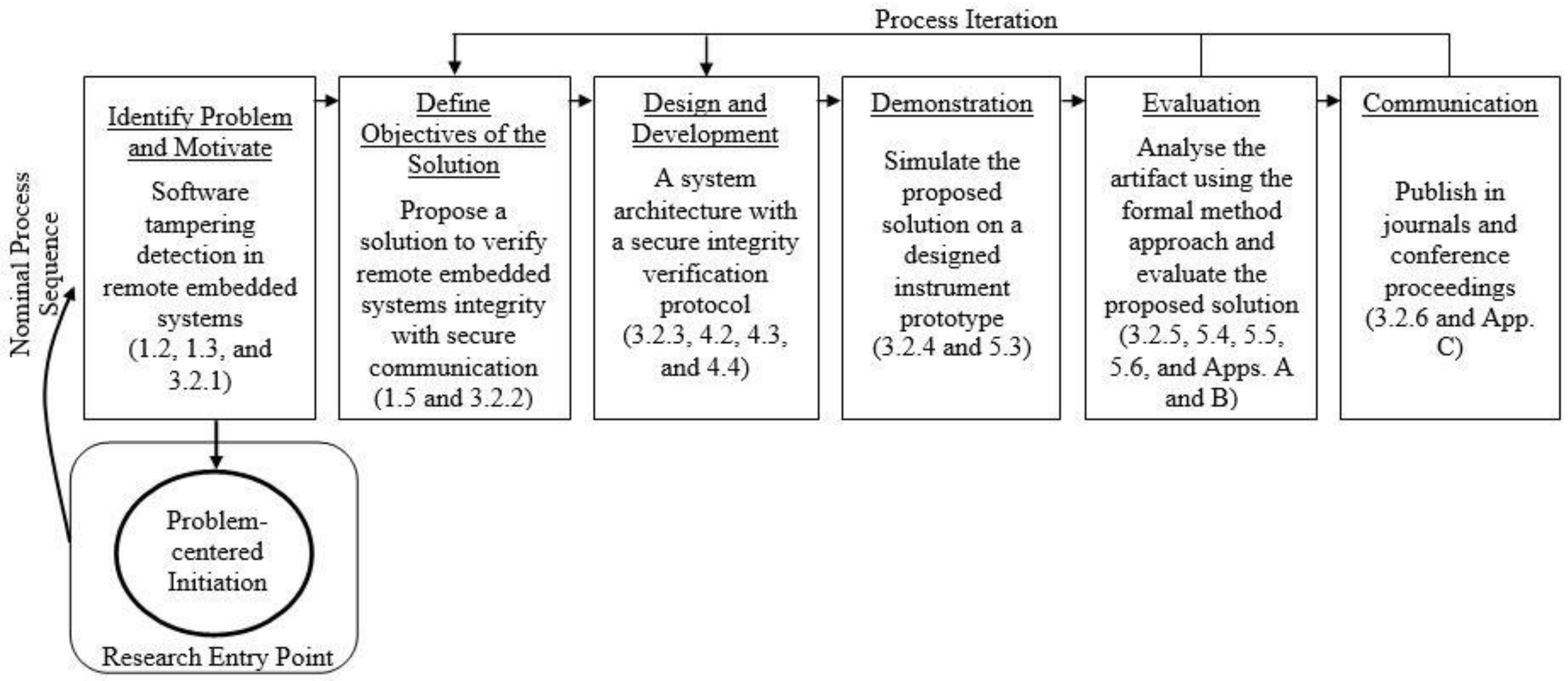
Process Iteration
Nominal Process Sequence
Identify Problem and Motivate
Software tampering detection in remote embedded systems (1.2, 1.3, and 3.2.1)
Define Objectives of the Solution
Propose a solution to verify remote embedded systems integrity with secure communication (1.5 and 3.2.2)
Design and Development
A system architecture with a secure integrity verification protocol (3.2.3, 4.2, 4.3, and 4.4)
Demonstration
Simulate the proposed solution on a designed instrument prototype (3.2.4 and 5.3)
Evaluation
Analyse the artifact using the formal method approach and evaluate the proposed solution (3.2.5, 5.4, 5.5, 5.6, and Apps. A and B)
Communication
Publish in journals and conference proceedings (3.2.6 and App. C)
Problem-centered Initiation
Research Entry Point

Figure 3.1 DSRM process for this research study

3.2.1 Problem Identification and Motivation

Embedded systems have become a part of the digital era, and their applications vary from small digital devices such as digital cameras and mobiles to large and crucial instruments such as smart cars and airplanes. They usually accomplish critical functions such as monitoring and controlling real-time objects and sensors, and processing vital data and information.

However, ESs face security challenges because they usually operate in a physically unprotected environment (Nimgaonkar et al. 2013a, 2013b; Sridhar et al. 2012). Accordingly, the fraudulent usage of digital measurement instruments has become one of the serious issues that needs to be tackled by security experts in (almost) every country (Al-Wosabi & Shukur 2015; Al-Wosabi, Shukur & Ibrahim 2015; Ibrahim et al. 2015).

Hence, tampering detection is gaining more attention and being given greater priority by ES designers and developers (Babar et al. 2011; Santucci 2010). In order to understand the resources required to complete this phase, Chapter 1 (sections 1.2 and 1.3) highlight the problem and motivation for conducting this research.

3.2.2 Objectives of the Solution

After defining the problem, the desired objective is declared in the second phase. The objective can be inferred from the defined problem and the literature review that was conducted to understand promising and achievable solutions. In fact, the research conducted in the previous phase highlights the issue of software tampering in ESs, and therefore, the proposal of a solution (i.e., a designed artifact) with secure communication to remotely verify the code integrity of a targeted ES is considered as the main objective of this research.

Basically, the major challenges encountered in designing a security protocol include integrating the secrecy and authentication features and evaluating those features in an acceptable and reliable approach. Hence, the formal method approach is applied because of the reliability and accuracy of the results of that approach. In order to identify and include the resources required to execute this phase, Chapter 2 reviews some interesting research studies and finds a number of existing (possible and viable) solutions related to the defined problem. Moreover, the defined objectives related to this research are listed in Chapter 1 (section 1.5).

3.2.3 Design and Development

This phase describes how the proposed artifact should act and how it could be assembled. Clearly, the literature review of related research that could be brought to bear in this study is the essential resource required to move from the second phase to accomplish this activity.

Firstly, the study proposes a conceptual framework that is partially extracted from the ISO/IEC/IEEE 42010:2011 model. It links five main components: potential stakeholders, main concerns, system-of-interest, architecture, and targeted mission.

Secondly, the study describes the code integrity verification system that consists of two main components. The task of the first component is to scan and gather information about the targeted system and store the required data in a database prepared for this purpose. The second component is used when the user requests that a code integrity check be performed on the targeted system. In addition, the protocol is designed to secure the data exchanged between the user program and the remote server system that receives the request for system integrity verification. The protocol features that need to be applied include the secrecy and reliability of the messages exchanged between the two parties. The proposed artifact is designed from basic cryptographic primitives that can be applied to achieve the main task of the described solution.

Further details of the proposed conceptual framework, the system architecture, and the secure code integrity verification protocol can be found in Chapter 4 (sections 4.2, 4.3, and 4.4).

3.2.4 Demonstration

The prototype (i.e., the demonstration) that a designer usually develop is one of the typical methods of artifact design. Usually, the prototype is an incomplete version of the artifact under the development that provides the minimum required features either for the purpose of presentation or evaluation (Dennis Wixom & Roth 2012; Frank et al. 2011).

In his article titled “Arduino for Teaching Embedded Systems. Are Computer Scientists and Engineering Educators Missing the Boat?” Peter Jamieson attempts to determine the suitability of utilizing the Arduino platform to teach an ES curriculum for computer science and engineering students. In his conclusion, he mentions that the students highly praised the Arduino platform, and he believes that the students’ final projects, which were related to the ES course, were better and more creative than before (Jamieson 2011).

In fact, the Arduino platform has been widely integrated into many scientific research studies for the purpose of developing different systems, such as designing a teaching and practising interaction system for musical applications (Berdahl & Ju 2011), designing customized educational robots (García-Saura & González-Gómez 2012), designing the Citizens as Sensors platform (Davidovic, Rančić & Stoimenov 2013), creating the Gesture Recognition Toolkit (Gillian & Paradiso 2014), demonstrating a building automation system (Gokceli et al. 2015), and designing a smartphone-based spectrometer for testing fruit ripeness (Das et al. 2016).

Thus, this study designs a digital measurement prototype using the Arduino Uno board, and develops the integrity verification system that performs two main functions using Java language. The first function is used to scan an ES’s code, store the encrypted form of that code into the designed database, and implement a secure one-way function (the cryptographic hash SHA-2) for computing a hash value. Then, the second function scans the targeted system and generates its HMAC value in order to compare it with the equivalent digest value of the system code previously stored in the database, and then shows the result of the integrity verification. Details of the developed prototype are provided in Chapter 5 (section 5.3).

3.2.5 Evaluation

This phase involves observing and estimating how well the proposed artifact supports a solution to the defined problem. Evaluation is an essential component in the design science research process (A. Hevner & Chatterjee 2010b). Validating the artifact means checking that the designed artifact works and does what it is meant to do, and that it is dependable in operational terms of achieving its main objective (Gregor & Hevner 2013). Thus, in this study the defined functions, and the secrecy and the authentication specifications of the proposed architecture and protocol are validated in this phase.

Nowadays, researchers emphasize the importance of using a formal method approach for system development, especially when the researcher needs to formally prove its security attributes (Gargantini et al. 2010). Chapter 5 (sections 5.4, 5.5, and 5.6) present the results of the reliability assessment and the evaluation tools used. Specifically, the study models and analyses the proposed system architecture with the integrity verification protocol using the process algebra CSP approach, and then evaluates the designed artifact using Casper and the model checker FDR. Note that Appendix A shows the Casper syntax used to specify the proposed protocol.

In addition, the approbation of the proposed solution requires the cooperation of professionals interested in system fraud detection. So, an expert evaluation survey is conducted using both closed- and open-ended questions. The expert evaluation questionnaire is designed by referring to two research studies (Najeh 2016; Snijders et al. 2015), and Appendix B contains the expert evaluation questionnaire developed for the proposed framework.

Accordingly, at the end of this stage and based on the evaluation results, the study may iterate back to the design and development stage in order to enhance the proposed system architecture and protocol.

3.2.6 Communication

Releasing the details of the study and the proposed design is important so that interested parties can determine the credibility of the research (Creswell 2014). Therefore, a number of research papers related to this study have been published in selected journals and presented at a number of conferences in order to communicate the research problem and outcomes of the study. The publications covered the literature reviewed for this research study, the proposed system architecture and protocol, and the evaluation results. Appendix C presents the papers that have been published regarding this research.

3.3 FORMAL METHOD APPROACH

Researchers state that formal models are more accurate in representing system specifications and permitting proficient reasoning about the secrecy and authentication features of security protocols. However, a number of researchers find it hard to create formal models because they are not familiar with mathematical notations.

This section explains the CSP formal method approach in a simplified way so that the reader can understand it. For example, Figure 3.2 describes the Needham-Schroeder Public Key Protocol (Lowe et al. 2009; Ryan et al. 2010), which involves a sender `S`, a receiver `R`, a sender's public key `PKey(S)`, a receiver's public key `PKey(R)`, and a message `m` encrypted with key $\{m\}_{key}$.

$$S \rightarrow R : \{S, ns\}_{PKey(R)}$$
$$R \rightarrow S : \{ns, nr\}_{PKey(S)}$$
$$S \rightarrow R : \{nr\}_{PKey(R)}$$

Figure 3.2 Needham-Schroeder Public Key Protocol

In this protocol, a sender attempts to establish secure communication with another agent (a receiver). Hence, the sender seeks to successfully perform mutual authentication, which means that each agent aims to ensure the accuracy of the identity of the other agent. First, the sender generates a nonce value `ns`. Then the sender sends this `ns` along with the sender's identity to the receiver, both encrypted using the receiver's public key (as shown in Figure 3.2: message 1).

When the receiver receives the encrypted message the receiver, and only the receiver, can decrypt that message using the receiver's private key to extract the received `ns`. The receiver then generates a fresh nonce value `nr` and sends this value along with the received nonce value `ns` to the sender, both encrypted under the sender's public key (as shown in Figure 3.2: message 2). When the sender gets and decrypts those values, the sender verifies his/her nonce value that the sender sent to the receiver (i.e., `ns`). If that value is verified, the sender is assured that the sender is communicating with the targeted receiver because only that receiver could have extracted the sent nonce `ns` from message 1.

The sender then returns the nonce value received from the receiver (i.e., `nr`), ciphered using the receiver's public key (as shown in Figure 3.2: message 3). When the receiver gets and decrypts this message, the receiver verifies his/her nonce value. If his/her nonce value is valid, then it would seem that the receiver can be assured that the receiver is communicating with an authenticated sender because only that sender could have extracted the sent nonce `nr` from message 2.

Basically, each process in CSP uses two channels – receive and send – to communicate with other processes via the medium. Thus the input of a message m into a process would be represented as `receive.x.y.m` and the output of a message m from the process would be represented as `send.x.y.m`, where `x` and `y` (either an agent or a server) are the names of the sender and addressee, respectively, and `m` is the message content. The CSP notation for the above protocol messages, in Figure 3.2, can be described as shown in Equation 3.1:

$$Initiator(S,ns) = env?R : Agent \setminus \{S\} \rightarrow send.S.R.\{S,ns\}_{PKey(R)} \tag{3.1}$$

$$\rightarrow \mathop{\square}_{nr \in Nonce} \left(\begin{array}{c} receive.R.S.\{ns,nr\}_{PKey(S)} \rightarrow \\ send.S.R.\{nr\}_{PKey(R)} \rightarrow \\ Session(S,R,ns,nr) \end{array} \right)$$

where Nonce is the set of all the valid nonce values that the agents can accept.

Note that CSP models always ensure that every generated nonce is really different from all the nonce values that already exist in the tested system. Also, whereas the PKey(S) and PKey(R) are the public keys of the sender and receiver, respectively, and can be retrieved by any agent, the corresponding private key is only known to the authenticated agent.

The connection between the two agents is initiated by the environment of the designed system. The “$env?R : Agent \setminus \{S\}$” represents this meaning by telling the sender to open a session with a targeted agent R; and the notation “$\setminus \{S\}$” prevents the sender requesting communication with him/herself. Then, the sender’s role states that the sender S uses its output channel to send to a targeted agent R its ID and nonce value encrypted using the receiver’s public key. Since the processes cannot be certain that the messages transferred during the communication are either correctly delivered or of the right form, an external choice can be used over the rest of all the acceptable messages. However, for the time being at least, the specification of the agent action when it gets into the Session state is not considered.

Now, Equation 3.2 describes the role of the receiver:

$$Responder(R, nr) = \mathop{\Box}_{\substack{S \in Agent \\ ns \in Nonce}} \begin{pmatrix} receive.S.R.\{S, ns\}_{PKey(R)} \rightarrow \\ send.R.S.\{ns, nr\}_{PKey(S)} \rightarrow \\ receive.S.R.\{nr\}_{PKey(R)} \rightarrow \\ Session(R, S, ns, nr) \end{pmatrix} \tag{3.2}$$

Observe that the receiver’s role states that the communication is started when an agent R receives a message from an initiator (i.e., S) who runs the protocol with him/her.

Generally, agents can be modelled as being capable of undertaking many synchronized runs in sender and receiver roles. The generalized interleave operator ($|||_{i \in I}\ P_i$) can be used to describe this in CSP, where each Pi represents a single protocol run. Moreover, to represent the system using different nonce values, pairwise disjoint sets `Nonce_I`$_S$ can be used to denote all of the nonce values of agent `S` in the role of initiator (i.e., sender) and `Nonce_R`$_S$ can be used to denote all of the nonce values of agent `S` in the role of responder (i.e., receiver). Hence, a particular agent `S` can be represented as in Equation 3.3:

$$\begin{gathered} \text{User}_S = |||_{\, n \in \text{Nonce\_Is}} \text{Initiator}(s,n) \\ ||| \\ |||_{\, n \in \text{Nonce\_Rs}} \text{Responder}(s,n) \end{gathered} \tag{3.3}$$

3.3.1 Secrecy Features

In order to describe the secrecy features in the above protocol, `Claim_Secret` message can be inserted at the end of the protocol run description. This property clarifies that the intruder cannot obtain the transferred message during a protocol run whenever the system has defined that particular data items should be known to only authenticated parties (i.e., the secrecy property). Hence, to ensure the secrecy of the sender nonce value `ns`, CSP notation could express this expectation in the description of `Initiator` as shown in Equation 3.4.

$$\begin{aligned} Initiator(S, ns) = env?R : Agent \setminus \{S\} \rightarrow send.S.R.\{S, ns\}_{PKey(R)} \\ \rightarrow \underset{nr \in Nonce}{\square} \begin{pmatrix} receive.R.S.\{ns, nr\}_{PKey(S)} \rightarrow \\ send.S.R.\{nr\}_{PKey(R)} \rightarrow \\ signal.Claim\_Secret.S.R.ns \rightarrow \\ Session(S, R, ns, nr) \end{pmatrix} \end{aligned} \tag{3.4}$$

This property states that the protocol guarantees that the sender nonce value `ns` used in a protocol run apparently between the sender `S` and responder `R` created by the sender `S` should be secret during the entire protocol run, and hence, the intruder should not be able to gain possession of the sender nonce value `ns`.

Moreover, the expectation of the responder’s role could be written in a similar way by using the `Claim_Secret` message at the end of the protocol run description as shown in Equation 3.5:

$$Responder(R, nr) = \underset{\substack{S \in Agent \\ ns \in Nonce}}{\Box} \begin{pmatrix} receive.S.R.\{S, ns\}_{PKey(R)} \rightarrow \\ send.R.S.\{ns, nr\}_{PKey(S)} \rightarrow \\ receive.S.R.\{nr\}_{PKey(R)} \rightarrow \\ signal.Claim\_Secret.R.S.nr \rightarrow \\ Session(R, S, ns, nr) \end{pmatrix} \tag{3.5}$$

The above expression (Equation 3.5) declares the responder’s role and at the end of it states that the nonce value `nr` created by the responder `R` must be secret during the entire protocol run, and hence, the intruder should not be able to gain possession of the responder nonce value `nr`.

Moreover, the requirement by an agent that a certain message `m` must be secret, and hence, the intruder must not be able to access it, is shown in Equation 3.6:

$$signal.Claim\_Secret.S.R.m \text{ in } trace \;\; \Rightarrow \; \neg \; (leak.m \text{ in } trace) \tag{3.6}$$

where a trace of a process is a predetermined sequence of symbols recording the events in which the process has been involved up to a certain instant in time. Therefore, the protocol specification of the sender states that the intruder must not be able to obtain the sender nonce value `ns` as shown in Equation 3.7:

$$\begin{aligned} &Secret_{S,R}(trace) \\ &\qquad = \forall\, ns \bullet signal.Claim\_Secret.S.R.ns \text{ in } trace \\ &\qquad \wedge S \in Honest \wedge R \in Honest \\ &\qquad \Rightarrow \neg\, (leak.ns \text{ in } trace) \end{aligned} \tag{3.7}$$

and the protocol specification of the responder states that the intruder must not be able to obtain the responder nonce value `nr` as shown in Equation 3.8:

$$
\begin{aligned}
&Secret_{R,S}(trace) \\
&\qquad = \forall\, nr \bullet signal.Claim\_Secret.R.S.nr \text{ in } trace \\
&\qquad \wedge S \in Honest \wedge R \in Honest \\
&\qquad \Rightarrow \neg\, (leak.nr \text{ in } trace)
\end{aligned}
\tag{3.8}
$$

Hence, it should be possible to describe the complete secrecy requirements of both agents (the initiator S and the responder R) based on the above expressions as follows: If an agent S has completed a protocol run apparently with an honest and uncompromised agent R, then the nonce value ns created by the agent S and received during that run by the agent R is not known to anyone other than the agent R. Similarly, if an agent R has completed a run with an honest and uncompromised agent S, then the nonce value nr created by the agent R and received by the agent S is not known to anyone other than the agent S.

3.3.2 Authentication Features

Now, it is necessary to conduct an entity authentication procedure, which means verifying an entity's claimed identity. An authentication property provides assurance to the responder R that a communication exchange has occurred apparently with the sender S, and also that the sender S is following a run apparently with that responder R.

In order to represent the authentication property, two events – `Commit.r.s` and `Running.s.r` – can be used. The first event can be introduced into the responder's description to identify the point at which the sender has been authenticated by the responder. The occurrence of this event in the responder's run states that the responder R has completed a protocol apparently with the initiator (i.e., the sender S).

The `Running.s.r` event is introduced into the sender's description to identify the point that must have been attained by the time the responder R performs the `Commit.r.s` event. The occurrence of this event in the sender's run states that the sender S is following a run apparently with the responder R. If a `Running.s.r` event has taken place by the time the `Commit.r.s` event is performed, then authentication between the two agents is achieved.

Since the above protocol aims to establish authentication in both directions, at this point in the discussion it would be worthwhile to describe the authentication of the initiator by the responder and also to explain how the initiator authenticates the responder. Note that the above protocol between the agent `S` and the agent `R` involves the two nonce values, `ns` and `nr`. So, the definition of the above events includes these values that are related to the protocol run.

Firstly, the authentication of the initiator by the responder could be identified (as shown in Figure 3.3), and thus the defined protocol descriptions could be enhanced as shown in Equation 3.9:

$$Initiator(S,ns) = env?R : Agent \setminus \{S\} \rightarrow send.S.R.\{S,ns\}_{PKey(R)} \tag{3.9}$$
$$\rightarrow \mathop{\Box}_{nr \in Nonce} \begin{pmatrix} receive.R.S.\{ns,nr\}_{PKey(S)} \rightarrow \\ signal.Running\_Initiator.S.R.ns.nr \rightarrow \\ send.S.R.\{nr\}_{PKey(R)} \rightarrow \\ Session(S,R,ns,nr) \end{pmatrix}$$

Moreover, the expectation of the responder's role could be written (as shown in Equation 3.10) in a similar way by using the `Commit` event at the end of the protocol run description:

$$Responder(R,nr) \tag{3.10}$$
$$= \mathop{\Box}_{\substack{S \in Agent \\ ns \in Nonce}} \begin{pmatrix} receive.S.R.\{S,ns\}_{PKey(R)} \rightarrow \\ send.R.S.\{ns,nr\}_{PKey(S)} \rightarrow \\ receive.S.R.\{nr\}_{PKey(R)} \rightarrow \\ signal.Commit\_Responder.R.S.ns.nr \rightarrow \\ Session(R,S,ns,nr) \end{pmatrix}$$

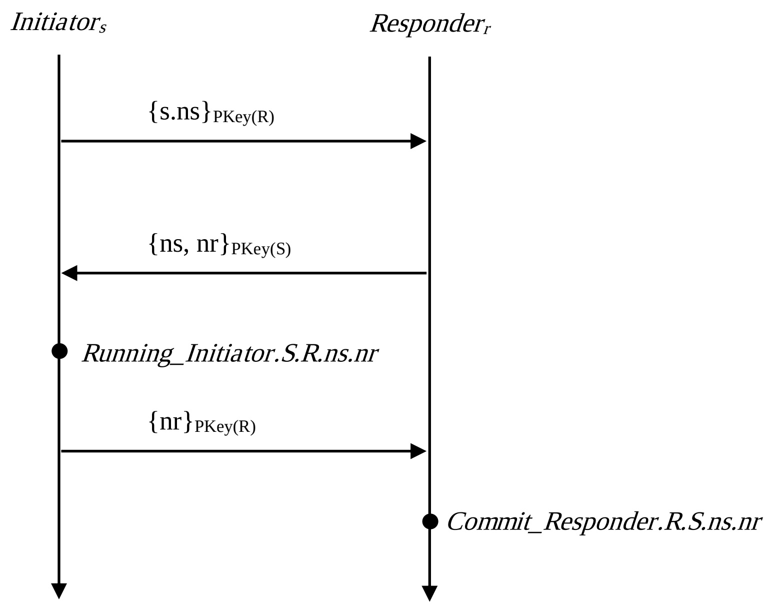

Figure 3.3 Authentication of initiator by responder

Furthermore, it is desirable to define the trace specification which states that whenever *a* `Commit` event appears in the trace then a corresponding `Running` event should present beforehand in that trace:

$$Commit \textbf{ in } trace \Rightarrow Running \textbf{ in } trace \tag{3.11}$$

This specification could be abbreviated as:

$$Running \textbf{ precedes } Commit \tag{3.12}$$

Hence, the above property will require that:

$$\begin{aligned} &Initiator \in Honest \\ &\quad \Rightarrow signal.Running\_Initiator.S.R.ns.nr \textbf{ precedes} \\ &\qquad signal.Commit\_Responder.R.S.ns.nr \end{aligned} \tag{3.13}$$

which could be represented as:

$$signal.Commit\_Responder.R.S.ns.nr \textbf{ in } trace \Rightarrow signal.Running\_Initiator.S.R.ns.nr \textbf{ in } trace \tag{3.14}$$

Secondly, the authentication of the responder by the initiator could be identified (as shown in Figure 3.4), and thus the defined protocol descriptions could be enhanced as shown in Equations 3.15 and 3.16:

$$\begin{aligned} &Initiator(S, ns) = \\ &env?R : Agent \setminus \{S\} \rightarrow send.S.R.\{S, ns\}_{PKey(R)} \rightarrow \\ &\qquad \mathop{\Box}_{nr \in Nonce} \begin{pmatrix} receive.R.S.\{ns, nr\}_{PKey(S)} \rightarrow \\ send.S.R.\{nr\}_{PKey(R)} \rightarrow \\ signal.Commit\_Initiator.S.R.ns.nr \rightarrow \\ Session(S, R, ns, nr) \end{pmatrix} \end{aligned} \tag{3.15}$$

$$\begin{aligned} &Responder(R, nr) = \\ &\qquad \mathop{\Box}_{\substack{S \in Agent \\ ns \in Nonce}} \begin{pmatrix} receive.S.R.\{S, ns\}_{PKey(R)} \rightarrow \\ signal.Running\_Responder.R.S.ns.nr \rightarrow \\ send.R.S.\{ns, nr\}_{PKey(S)} \rightarrow \\ receive.S.R.\{nr\}_{PKey(R)} \rightarrow \\ Session(R, S, ns, nr) \end{pmatrix} \end{aligned} \tag{3.16}$$

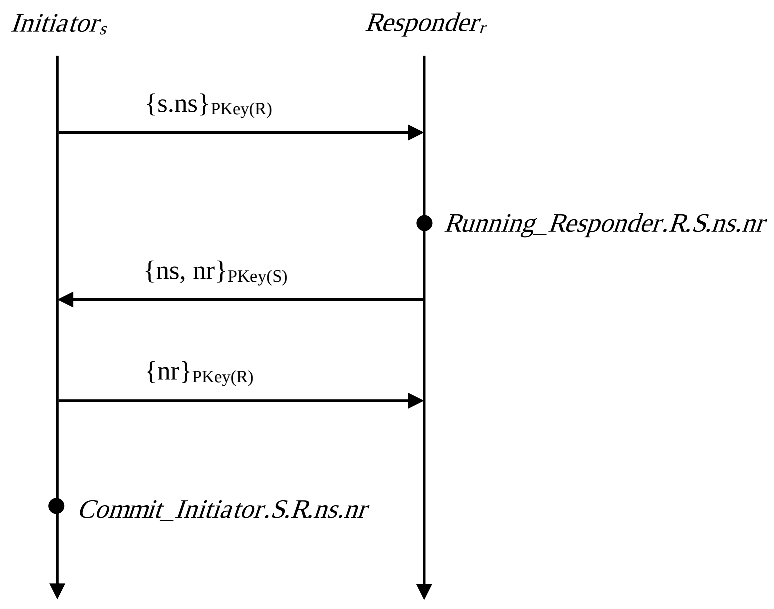

Figure 3.4 Authentication of responder by initiator