© 2008 Suvda Myagmar
SECURE CONFIGURATION FOR SOFTWARE DEFINED RADIO

BY

SUVDA MYAGMAR

B.S., Concord University, 2001
M.S., University of Illinois at Urbana-Champaign, 2005

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Professor Roy H. Campbell, Chair
Professor Marianne Winslett
Professor Carl A. Gunter
Professor Klara Nahrstedt
Abstract

This thesis addresses secure configuration of reconfigurable radio systems such as software defined radio. Software defined radio (SDR) supports integration and co-existence of multiple radio access technologies on a general-purpose radio equipment. Its reconfigurable radio capabilities make it an ideal solution for interoperability among heterogeneous wireless communications systems, especially in public safety domain. Because an SDR device is able to switch its operating mode by configuring its baseband software and change its radio parameters such as frequency, output power, and modulation format, it is challenging to ensure that radiated emissions of the radio conform with FCC regulations, and hinders widespread adoption of this technology. For SDR systems to realize their full potential, they must be reconfigurable through automated deployment of SDR components. As the industry is moving toward open architectures, portability and configurability of third party software must be provided. We present a configuration framework that automates configuration of an SDR terminal using third party software components, validates conformance of radio configuration, and attests the configuration to a service provider.

For automated configuration, we developed a hierarchical two-phase methodology that supports portable configuration profiles and plug-n-play radio composition. We use a graph mapping model to convert configuration profile into a deployable flowraph of waveform components. We show how capabilities and regulations can be reflected in configuration profiles, and how these profiles can be ported using XML templates.
For configuration validation, we present a model for component-based certification of an SDR terminal. Methodologies for verifying certification of downloaded software, checking consistency of configuration, and securing the download channel are used to provide conformance validation for SDR terminal. For configuration attestation, we use trusted computing services to support a trusted configuration platform. We outline a secure boot sequence that allow an SDR terminal to ascertain its current configuration to a remote party in a robust manner.
To my father and brothers, in memory of my mother.
Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my adviser, Roy H. Campbell, who read my numerous revisions and helped make some sense of the confusion. Also thanks to my committee members, Marianne Winslett, Carl Gunter, and Klara Nahrstedt, who offered guidance and support. Thanks to the Information Trust Institute’s TCIP project for awarding me a research assistanship, providing me with the financial means to complete this project. And finally, thanks to my father, brothers, and numerous friends who endured this long process with me, always offering support and love.
Table of Contents

List of Tables .............................................. ix
List of Figures ............................................. x
List of Abbreviations ....................................... xii

Chapter 1 Introduction ....................................... 1

Chapter 2 Background ......................................... 5
  2.1 SDR as Reconfigurable Radio .......................... 5
  2.2 SDR Applications ....................................... 7
    2.2.1 Extending GPS Signals Indoors ................... 8
    2.2.2 Communication for Critical Infrastructure ....... 9
  2.3 Current SDR Hardware Capabilities .................. 15

Chapter 3 Problem Statement ................................. 18
  3.1 Problem Statement ..................................... 18
    3.1.1 Configuration Security .......................... 19
    3.1.2 Automated Configuration ......................... 20
    3.1.3 Configuration Validation ......................... 20
    3.1.4 Configuration Attestation ......................... 22
  3.2 Research Scope ....................................... 22
  3.3 Assumptions .......................................... 24
  3.4 Thesis Statement ..................................... 25
  3.5 Success Criteria ..................................... 25

Chapter 4 Configuration Security ............................ 27
  4.1 Threat Model .......................................... 28
  4.2 Matrix of Threat Scenarios ........................... 31
  4.3 Security Requirements for SDR Configuration ........ 32

Chapter 5 Automated Configuration .......................... 35
  5.1 Configuration Process ................................ 36
    5.1.1 Configuration Example: FM Receiver ............ 39
  5.2 Modeling Configuration Concepts ..................... 41
    5.2.1 Composition Model ............................... 41
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2</td>
<td>Component Model</td>
<td>42</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Configuration Graph Model</td>
<td>44</td>
</tr>
<tr>
<td>5.3</td>
<td>Configuration Profile</td>
<td>45</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Component Classification</td>
<td>46</td>
</tr>
<tr>
<td>5.3.2</td>
<td>XML Overview</td>
<td>48</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Template for Configuration Profile</td>
<td>48</td>
</tr>
<tr>
<td>5.4</td>
<td>Evaluation</td>
<td>50</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Performance Time</td>
<td>50</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Radio Functionality</td>
<td>51</td>
</tr>
<tr>
<td>6.1</td>
<td>SDR Certification</td>
<td>59</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Component-Based Certification</td>
<td>60</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Certifiable by Construction</td>
<td>62</td>
</tr>
<tr>
<td>6.2</td>
<td>Secure Software Download</td>
<td>65</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Protocol Assumptions</td>
<td>65</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Protocol Description</td>
<td>66</td>
</tr>
<tr>
<td>6.3</td>
<td>Conformance Validation</td>
<td>69</td>
</tr>
<tr>
<td>6.4</td>
<td>Compatibility Check</td>
<td>71</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation</td>
<td>72</td>
</tr>
<tr>
<td>6.6</td>
<td>SDR Certification</td>
<td>84</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Component-Based Certification</td>
<td>84</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Certifiable by Construction</td>
<td>85</td>
</tr>
<tr>
<td>6.7</td>
<td>Secure Software Download</td>
<td>88</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Protocol Assumptions</td>
<td>88</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Protocol Description</td>
<td>89</td>
</tr>
<tr>
<td>6.8</td>
<td>Conformance Validation</td>
<td>90</td>
</tr>
<tr>
<td>6.9</td>
<td>Compatibility Check</td>
<td>91</td>
</tr>
<tr>
<td>6.10</td>
<td>Evaluation</td>
<td>92</td>
</tr>
<tr>
<td>7.1</td>
<td>Trusted Computing</td>
<td>74</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Trusted Computing Services</td>
<td>75</td>
</tr>
<tr>
<td>7.2</td>
<td>Trusted SDR Platform</td>
<td>78</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Secure Boot</td>
<td>78</td>
</tr>
<tr>
<td>7.3</td>
<td>SDR Remote Attestation</td>
<td>81</td>
</tr>
<tr>
<td>7.4</td>
<td>Evaluation</td>
<td>83</td>
</tr>
<tr>
<td>8.1</td>
<td>Functional Requirements</td>
<td>84</td>
</tr>
<tr>
<td>8.2</td>
<td>Architecture</td>
<td>85</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Configuration Management Module</td>
<td>88</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Policy Management Module</td>
<td>89</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Configuration Attestation Module</td>
<td>90</td>
</tr>
<tr>
<td>8.3</td>
<td>Secure Execution Environment</td>
<td>91</td>
</tr>
<tr>
<td>8.4</td>
<td>Implementation Platform</td>
<td>95</td>
</tr>
<tr>
<td>8.4.1</td>
<td>GNU Radio Software</td>
<td>96</td>
</tr>
<tr>
<td>8.4.2</td>
<td>USRP Hardware</td>
<td>99</td>
</tr>
<tr>
<td>9.1</td>
<td>SDR Initiatives</td>
<td>104</td>
</tr>
<tr>
<td>9.2</td>
<td>Research in SDR Security</td>
<td>106</td>
</tr>
<tr>
<td>9.3</td>
<td>Research in SDR Configuration</td>
<td>108</td>
</tr>
<tr>
<td>9.4</td>
<td>Component Composition</td>
<td>109</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>9.5 Security Models</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 10 Conclusions</strong></td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>10.1 Conclusions</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>10.2 Summary of Contributions</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>10.3 Future Research</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>References</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Author’s Biography</td>
<td>124</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

4.1 Security threat model. .................................. 29
4.2 SDR security threat scenarios. .......................... 34
5.1 API of amplifier class component. ....................... 47
List of Figures

2.1 A generic digital radio transceiver. .................................. 6
2.2 POSCOMM time-of-arrival network. ................................. 9
2.3 Wireless communication in the power grid. ...................... 13

5.1 Flowchart of the configuration process. .......................... 53
5.2 Configuring a FM broadcast receiver. .............................. 54
5.3 Configuration Composition Model. ................................. 54
5.4 Software Component Model. ....................................... 55
5.5 XML template of FM receiver configuration profile. ............. 56
5.6 FM radio signal received by GNU Radio. .......................... 57
5.7 FM radio signal received by GNU Radio. .......................... 57

6.1 Component-based certification. .................................... 61
6.2 Secure download protocol. ......................................... 66

8.1 Architecture of the configuration framework. ..................... 86
8.2 Runtime overhead of the memory protection mechanism. ........ 95
8.3 GNU Radio flowgraph for a simple FM Receiver. ................ 97
8.4 USRP mother board with two RX and two TX daughter boards. . 100
8.5 Block diagram of a USRP board. ...................... 101
8.6 Block diagram of the USRP receive path. ............... 103
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Analog-to-Digital</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DA</td>
<td>Digital-to-Analog</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite impulse response</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU’s Not UNIX (recursive acronym)</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Component Interconnect</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>SDR</td>
<td>Software Defined Radio</td>
</tr>
<tr>
<td>USRP</td>
<td>Universal Software Radio Peripheral</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The world of wireless is rapidly evolving as new technologies such as Wi-Fi, WiMAX and OFDM join already deployed wireless solutions such as 2.5G and 3G. This proliferation of wireless technologies means that devices in the future will be required not only to support multiple air interfaces but also be capable of working in multiple frequency bands. Implementing these multiple functionalities in hardware would be expensive and extremely difficult, hence the drive towards Software Defined Radio (SDR).

Software defined radio technology implements radio wave functionalities such as modulation, amplification, phasing, mixing, and up/down-converting as software modules running on a generic hardware platform. Traditional radios are built for a particular frequency range, modulation type and output power. In SDR, these radio frequency (RF) parameters can be configured when the radio device is in use rather than when it is manufactured. This enables highly flexible handheld devices that can switch from one network technology to another to suit a particular application or environment. Furthermore, the software that implement various radio technologies and services can be downloaded over-the-air onto the handsets.

Interoperability across heterogeneous radio networks supported by SDR is crucial in public safety. In emergency situations, public safety agencies such as police, fire departments, and emergency medical services need to be able to communicate with one
another and share information. A 2004 U.S. Conference of Mayors Interoperability Survey [1] found that in 77% of the 192 cities surveyed, police and fire departments could not talk to one another, and in 66% of them, all three agencies were not interoperable. Communication has been fingered repeatedly as the single largest barrier to emergency response after Hurricane Katrina devastated the Gulf Coast. A similar problem existed earlier in the aftermath of the 9/11.

SDR based equipment may be configured to practically any setting and may potentially implement any radio interface, a radio access standard or even a rogue scheme. The reconfigurability capability opens way for any type of intended as well as unintended incorrect system implementation. In particular SDR technology based terminals can be easily circulated and may be put into use in administrative areas where regulation or law prohibits the use of a reconfiguration capability. There also will be the problem of how to prevent unintentional and intentional configurations causing interfering emissions and the unintentional incrimination of users arises.

As we see, SDR technology has numerous technical challenges that need to be resolved before it can be successfully deployed. General SDR challenges are the provision of: advanced spectrum management for dynamic allocation of spectrum according to traffic needs, robust security measures for secure configuration of terminals, secure software download, and to prevent misuse of the system, open software architecture with well defined interfaces.

In this work, we investigate secure and automated configuration for SDR. Several previous works researched SDR configuration issues [2, 3, 4, 5]. However, they assume that software modules composing a radio configuration is supplied only by the manufacturer and its configuration is pre-defined by the vendor or hardware manufacturer. None of them addresses security implications of a highly configurable radio
with portable configuration specifications and reusable third party components. To the best of our knowledge, the problem of composing a valid radio given third party components has not been extensibly addressed.

We need to understand the dimensions and objectives of our system security. Chapter 4 we define the threat model we used to identify and describe security threats. Then we present example of threat scenarios that pertain to an SDR terminal. Finally we identify security requirements for SDR configuration.

To facilitate a plug-n-play component composition and make the configuration specification portable, we propose an automated configuration methodology for SDR in Chapter 5. The core of the methodology is a configuration processor which composes DSP components according to a configuration specification of the desired radio mode. We give a schematic view of the configuration process along with an example scenario, and model such notions as component, configuration, mapping of a configuration profile into a deployable flowgraph.

We consider certification of reconfigurable devices such as SDR, and address this issue within the configuration framework in Chapter 6. FCC requires each SDR hardware and software combination be tested for certification. This severely limits the widespread application of SDR equipment. We postulate that it is possible to certify a component-based SDR terminal where the operating software, digital signal processing (DSP) software components, and configuration description are decoupled. We present a methodology for component-based certification which assures SDR terminal’s conformity based on a separate certification of its architecture layer. We outline the component-based certification process, and present a secure download protocol based on trusted computing functionality. We also discuss how to verify regulatory conformance of downloaded software, and what pragmatic compatibility
checks exist out there.

Remote attestation is a technique for ascertaining the operating state of a radio platform to a remotely located party in a secure manner. The platform software that manages and controls the device configuration should be trusted, and be able to prove its trustworthiness to the remote party. We use *Trusted Computing* concept to ensure the integrity of radio platform in Chapter 7. We briefly review trusted computing concepts, then describe what specific TPM functionalities we use to provide a trusted SDR platform and remote attestation service for SDR.
Chapter 2

Background

This chapter we present some background information related to this thesis. Software defined radio is a relatively new technology and its important to understand the unique features of SDR compared to other wireless communications systems in order to accurately assess its capabilities and challenges. In section 2.1 we briefly discuss SDR architecture and features. Then in Section 2.2 we give examples of SDR applications in public safety and critical infrastructure domains. Finally we discuss current hardware capabilities of SDR in Section 2.3.

2.1 SDR as Reconfigurable Radio

Federal Communications Commission (FCC) adopted the following regulatory definition for software defined radio [6].

*Software Defined Radio.* A radio that includes a transmitter in which the operating parameters of frequency range, modulation type or maximum output power (either radiated or conducted) can be altered by making a change in software without making any changes to hardware components that affect the radio frequency emissions.
Figure 2.1 shows the block diagram of a generic digital radio transceiver consisting of the radio frequency (RF) front-end, the intermediate frequency (IF) section and the baseband section.

The RF front-end functions as the transmitter and receiver for the RF signal received via the antenna. On the receive path, it down-converts the RF signal to IF signal for further processing in the IF section. On the transmit path, it performs up-conversion to convert the IF signal to RF signal followed by power amplification.

The IF section is responsible for analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC) on the receive path and the transmit path, respectively. The digital down converter (DDC) and digital up-converter (DUC) that proceeds and precedes the ADC and DAC respectively, jointly assume the functions of a modem.

The baseband section performs baseband operations such as connection setup, equalization, frequency hopping, timing recovery and correlation. In software defined radio, the baseband processing and the DDC and DUC modules (highlighted in the figure) are designed to be software programmable [7]. The link layer protocols, modulation and demodulation operations are implemented in software.

![Figure 2.1: A generic digital radio transceiver.](image)
Reconfigurability provides essential mechanisms to terminals and network segments to adapt dynamically, transparently and securely to the most appropriate radio access technology. SDR presents many benefits to various communications entities. Equipment manufacturers can use a common design for multi-functional radios leading to increased market size for a single product. For operators, the interoperability of different networks is enhanced and system upgrades and bugs fixing becomes easier to manage and implement. Consumers gain enhanced functionality of their radio devices and the possibility to achieve ubiquitous connectivity. SDR enables the fast introduction of new services into mobile networks without requiring the purchase of new terminals.

SDR implements in software the LL/MAC layer components such as MAC protocols, framer, encoder, and some parts of the physical layer such as modulation and demodulation. Just few very time intensive functions are still implemented in hardware, for example the sampling of signals and the synchronization between component parts. This makes an SDR very flexible and theoretically, it can morph into a cell phone using GPRS, wireless communication system using IEEE 802.11, navigation system using GPS, AM/FM or HDTV receiver.

2.2 SDR Applications

In this section we present applications of software defined radio technology. Due to its powerful reconfigurable capabilities, SDR is an ideal solution in public safety and military domains where interoperability among heterogenous wireless communications systems is important. First we discuss how SDR can be used to extend GPS signal reception indoors for first responders. Then we discuss how power grids can deploy
2.2.1 Extending GPS Signals Indoors

A networked radionavigation approach that augments GPS signals with time-of-
arrival observations using a software-defined radio can overcome attenuation, and
often complete blockage, of GPS inside buildings or in urban canyons. The SDR can
operate both as a GPS receiver and also as a 900 MHz transceiver operating within
the ISM band. Applications for this technology include firefighters and other first
responders, and military operations in urban terrain.

While GPS is the natural choice for providing navigation in an outdoors environment,
the urban environment places a significant challenge for positioning using GPS. The
GPS signals can be significantly attenuated, and often completely blocked, inside
buildings or in urban canyons. Personnel tracking inside buildings for improved com-
mmand, control, and rescue rates high on the list of priority needs prepared by the
U.S. Department of Homeland Security for first responders. Tracking firefighters will
allow for better tactical decisions and faster fire suppression, resulting in decreased
property losses.

A company called NavSys developed an SDR that includes the capability to operate
both as a GPS receiver and as a 900 MHz transceiver operating within the industrial,
scientific, and medical (ISM) band. Since both the GPS and communications
functions reside within common radio hardware, this positioning and communica-
tions (POSCOMM) [8] device can use the two functionalities to provide a positioning
capability that leverages both the GPS-derived pseudorange and carrier-phase obser-
vations and the communications channel’s time-of-arrival (TOA) observations.
The POSCOMM SDRs are designed to operate in a networked architecture, as shown in Figure 2.2, where master units designated as transmitters provide TOA-augmented navigation to slave units operating as receivers in a GPS-denied urban environment. The master units transmit a TOA message that includes a pseudorandom sequence from which the TOA at the slave unit can be precisely determined. A message is also sent including the precise time of transmission of the TOA message and the precise location of the master unit based on the GPS observations. The TOA differenced with the time-of-transmission provides the slave unit with a pseudorange observation from each of the master units’ locations. This can be used to solve for the position of the slave either using the TOA updates alone or a combination of both the GPS and TOA observations.

2.2.2 Communication for Critical Infrastructure

Critical infrastructures are systems whose failure or destruction would have a debilitating impact on the defense or economic security of the nation [9]. These systems
include electrical power systems. The North American power grid involves nearly 3,500 utility organizations delivering electricity over more than 200,000 miles of transmission lines to 300 million people. And yet this critical infrastructure is not able to cope with grid-wide phenomena such as the 2003 Northeast Blackout that affected 50 million people and caused financial losses of up to $6 billion due to the power outage. A major culprit is the inadequate communication infrastructure of the power grid.

The power grid’s existing communication architecture limits the deployment of control and protection schemes to manage electric power generation, transmission, and distribution effectively. Ideally, grid companies want fine grain monitoring and control of their distribution network, even down to the last transformer. Supervisory Control and Data Acquisition (SCADA) systems have been used for years in the power grid to monitor and control substations and field instruments. However, there are still many distribution substations that are not equipped with SCADA and require manual, human maintenance. During a recent field trip to AmerenIP’s sites, we found that out of their 550 substations in Illinois, only 200 were equipped with SCADA systems.

A SCADA system gathers information (such as where high voltage has occurred) from field instruments, transfers the information back to the substation and control center, alerts the control center of any alarms, carries out any necessary local analysis and control such as determining if the voltage level has risen above or dropped below a critical level, and allows the control center to modify control on the distribution system. The importance of this system is that it can provide early warning of potential disaster situations and provides safe, non-destructive operation of devices and transmission lines. Unfortunately, many of those substations that require SCADA would require installing necessary communication lines to the control center and to field instruments such as pole-top Remote Terminal Units (RTUs). The technology
of field instruments has evolved beyond simple RTUs, with increasing deployment of Intelligent Electronic Devices (IEDs) and synchronous Phaser Measurement Units (PMUs) that give a much more detailed insight into grid dynamics and post-incident analysis [10]. However, this data cannot easily be utilized beyond the substation when the grid has limited communication lines.

There is also a need for point-to-point communication between substations to implement Special Protection Schemes (SPS). SPS address some of the wide-area control issues where the occurrence of particular events or measurements at one point in the grid triggers actions (such as a breaker tripping) at another. The existing approaches to communication architecture do not link substations directly. In short, communication networks are needed to connect SCADA control centers with substations and fields instruments, and to link substations with other substations. Such a network can be very expensive to build and maintain.

Traditional solutions for implementing communication lines have been to lease lines from telecom providers at very high installation and maintenance costs. Leased telephone channels also provide limited reliability and sometimes may not be even available at the substation site. During our field trip to AmerenIP, it was disclosed to us that the local phone company will no longer give them dedicated copper lines for their substations. The other difficult aspect of installing physical communication lines is that distribution networks cover very large geographic areas.

One might think that the Power Line Carrier (PLC) is a good solution for this problem. PLC uses the power lines to transmit radio frequency signals in the range of 30-500 kHz [11]. PLC is not subject to the unreliability of leased telephone lines. However, power lines are a hostile environment for signal propagation, with excessive noise levels and cable attenuation. Also PLC is not independent of the
power distribution system, thus making it unsuitable for emergency situations when the communication lines must operate even if the power lines are out of service.

Such difficult networking problems can be solved with wireless radio technologies. In general, wireless communication offers lower installation and maintenance costs than fixed communication lines, and they provide more flexibility in network configuration. There are many different types of wireless technologies such as satellites, very high frequency radio, ultra high frequency radio, and microwave radio. Each has its own advantages and disadvantages. The satellite system contains a number of radio transponders which receive and retransmit frequencies to ground stations within its coverage on the earth’s surface. Advantages of the satellite system are wide area coverage, easy access to remote sites, and low error rates. Its disadvantages are transmission time delay, and continual leasing costs incurred on time-of-use basis.

The Very High Frequency (VHF) radio operates in 30-300 MHz band and mostly reserved for mobile services. On the other hand, Ultra High Frequency (UHF) systems operate in 300-3000 MHz band, and can be Point-To-Point (PTP), Point-To-Multipoint (PTM), Trunked Mobile Radio (TPR), or spread spectrum systems. VHF radios, PTP and PTM radios in UHF have advantages of propagating over non-line-of-sight paths, low cost radios, and available frequency assignments. Their disadvantages are low channel capacity and low digital data bit rate. Spread spectrum systems are the basis for many wireless applications including 802.11 networks, and can operate with low power radios without licenses. However, these radios are subject to interference from co-channel transmitters and have limited path lengths because of restrictions on RF power output.

Microwave radio is a UHF radio operating at frequencies above 1 GHz. These systems have high channel capacities and data rates. However, microwave radios require line
of sight clearance, are more expensive to develop than VHF and UHF, and sometimes the appropriate frequency assignments are not available in urban areas.

A SCADA radio device can be implemented using any of the above mentioned radio technologies. Figure 2.3 illustrates how wireless communication could be deployed in the power grid. Researchers have conducted experiments and evaluations of these radios including 802.11, GPRS, and 900 MHz [12, 13, 14]. Each one has one or more disadvantages, and the technology may become outdated in the long term. It is no easy task to upgrade thousands of equipment in the power grid. It is costly and time consuming. The existing power grid communication lines and equipment are outdated for a reason- they have been installed decades ago.

![Figure 2.3: Wireless communication in the power grid.](image)

Thus, the ideal radio platform for the power grid should accommodate future wireless communication needs, have low installation and maintenance costs, and be capable of reconfiguring and updating its operation and software. Such considerations favor examining Software Defined Radio (SDR) as a possible radio platform. SDR implements the functions of radio devices such as modulation, signal generation, coding and link-layer protocols as software modules running on a generic hardware platform.
Traditional radios are built for a particular frequency range, modulation type and output power. In SDR, these radio frequency (RF) parameters can be configured when the radio device is in use rather than when it is manufactured. This enables highly flexible radios that can switch from one communication technology to another to suit a particular application or environment. Furthermore, the protocols that implement various radio technologies and services can be downloaded over-the-air onto the radio device.

Software radio is a suitable wireless media to replace legacy communication devices in power grids. The reconfigurability of SDR supports the integration and co-existence of multiple radio access technologies on a general-purpose radio equipment, enabling implementation of powerful SCADA networks. At the same time, the wireless and reconfigurable nature of SDR introduces potentially serious security problems such as unauthorized access to the SCADA system, spoofing or suppression of utility alarms, and configuration of a malfunctioning or malicious radio equipment.

We investigate the security challenges of deploying software radios in the power grid [15, 16]. To the best of our knowledge, this problem has not been addressed before. The security goals are to prevent installation and execution of unauthorized software, ensure the device operates within allowed frequency bands and power levels, and prevent the device from operating in a malicious manner. The main challenges are how to dynamically and securely configure software components on the radio that are possibly originating from different vendors as the power industry is shifting from proprietary protocols toward open and standard protocols, and how to attest the validity of the radio configuration to a master node. We presented these and other security challenges in detail, and based on our analysis, we formulated security requirements of a trusted configuration framework for SDR in the power grid.
2.3 Current SDR Hardware Capabilities

The purpose of this discussion is to illustrate the inherent hardware limitations, and therefore, the intrinsic security, expected to be found in current and next generation handsets. The notion of a Software Defined Radio as a device capable of near limitless flexibility is not quite true in the current state of its hardware capabilities. This is the reason why so far, only a limited number of academic, government, and commercial efforts are under way to address SDR issues. The technology itself quite remarkable and provides many advantages. However, its hardware has not caught up with all the possibilities of the technology, and hinders a widespread adoption of SDR. This is particularly true for the commercial wireless handset market, where the public demand for small, lightweight, low cost, battery efficient products, is a paramount consideration for equipment manufacturers.

Manufacturers have, and will continue to, design products that operate with very specific and limited radio parameters (e.g. modulation, frequency, output power). It is true that with the emergence of new communications systems (e.g. wireless LAN) that the market will demand devices with increasing degrees of multi-band and multi-mode functionality. This demand will drive equipment manufacturers to seek out the most optimal implementation technologies (like SDR) to address the product requirements. Even in these cases, however, the capabilities of these multi-mode devices will be essentially limited to the specific set of wireless services that were considered at the time of product design.

Advances in semiconductor technologies have enabled transmitter and receiver architectures to have fewer Intermediate Frequency (IF) stages, and less signal processing/filtering achieved in hardware circuitry. Nevertheless, current and future
generation equipment must still depend on electromechanical devices such as RF filters and resonators. Demanding product size and cost constraints dictate that these hardware elements be properly specified. Consequently, it might be unreasonable to expect (current or future generation) handsets to have the inherent hardware ability to operate significantly outside of the frequency bands, in which, they were designed to operate.

With the continuing trend toward sophisticated modulation protocols, it is increasingly common for modem functionality to be implemented digitally. Advances in microprocessor technologies will enable the trend toward software programmable modulators and demodulators. Therefore, of the three RF parameters (frequency, modulation, and output power), modulation will typically have the greatest degree of SW flexibility found in current and future generation radio architectures. However, an improper change to only the modulation format has limited potential to produce harmful consequences. This potential is primarily limited to Denial of Service (DoS) scenarios, where individual units are rendered inoperative due to an improper change in modulation format. Providing security against such scenarios, therefore, is important and we included this threat in our threat model.

Much like frequency, output power is limited by inherent electrical and mechanical limitations of the hardware design. Power amplifier circuitry is optimally designed to produce the rated maximum output power, with minimal headroom. What margin does exist is the result of typical design and production tolerances. Of the three RF parameter (frequency, modulation, and output power), output power is the least likely to be impacted by the emergence of SDR technologies. Realistic security threats involving output power are mostly confined to scenarios whereby a handset operates at its rated maximum, when it should be operating at a power reduced state. This threat scenarios is also included in the security considerations of the configuration
framework.
Chapter 3

Problem Statement

This chapter presents the problems addressed by this thesis and briefly explains our approach to solving them. First we state the general problem, then discuss individual problems. Finally we state our thesis and the success criteria for this work.

3.1 Problem Statement

Interoperability across heterogeneous radio networks supported by SDR is crucial in public safety. However, SDR technology has numerous technical challenges that need to be resolved before it can be successfully deployed. General SDR challenges are: security of the SDR terminal, support for third party component configuration, ensuring that radiated emissions of the radio conform with regulations.

The problem I address is that of secure and automated configuration of an SDR terminal. It is automated because it supports plug-n-play software components supplied by different vendors and the blueprint for composing them, called configuration profile, also can be supplied by any certified software provider. It should satisfy the basic security properties identified by the threat model. RF emissions of commercial equipment is regulated by FCC. There are strict FCC rules that need to be adhered to. We enforce regulations conformity in our system.
3.1.1 Configuration Security

Despite its many benefits, reconfigurability of SDR may lead to serious security problems such as unauthorized interception of configuration data and software, malfunctioning radio equipment, and impersonation of terminal or network. Security of SDR configuration is just as desirable as its automated nature. We specify the threat model and security requirements of the configuration framework in Chapter 4.

In general, it is essential that the radio functionality of the SDR terminal is not unintentionally altered or that non-authorized sources do not have access to SDR related components. For example, software can be introduced into a device that changes its RF operating characteristics so that it no longer functions within the regulated constraints (e.g. frequency range, modulation type, output power). Such changes in RF parameters may be used to launch denial of service (DoS) attacks on the SDR device or entire wireless network. For instance, such an application could cause the terminal transmitter to always transmit at maximum power, allowing the user to get better performance, but at the same time actually degrading the overall performance of the system.

The challenge is to provide security services for the configuration framework that ensures integrity, confidentiality, and authentication of relevant data and processes. Access to configuration functions within the terminal must be safeguarded and also attempts at fraudulent access from the outside world must be prevented. Configuration data and software modules must be downloaded in a secure manner from verified sources, and the security functions themselves have to be trustworthy.
3.1.2 Automated Configuration

Previously it was assumed that the configuration setups are pre-defined and pre-verified by an external party such as a hardware manufacturer, and they either already reside in the SDR terminal or are downloaded over-the-air at configuration time [2, 3]. The implication was that software modules composing a particular configuration were originated from the same vendor, most likely the hardware manufacturer. Since the software vendor is full aware of the properties and dependencies of the software modules, it can provide configuration setup that precisely defines the sequence and identity of individual components.

The situation becomes complicated when a single radio configuration uses radio software modules originating from several, different providers. It would be necessary to compose the radio configuration automatically. The challenge is to provide composition of radio software components with certain constraints (e.g. user preferences, regulatory body and network operator requirements, hardware specifications). These constraints are provided by machine-readable policies and specify the radio access technology (e.g. GSM, UMTS), allocated frequency band (e.g. 806-902 MHz) and hardware parameters (e.g. IF, power, interfaces). The difficulty lies in mapping configuration policies into a functional dataflow graph and then, further, into an executable dataflow graph. The executable dataflow graph states which software modules implement functional blocks, and it is used to activate a new radio configuration.

3.1.3 Configuration Validation

FCC requires each SDR hardware and software combination be tested for certification [6]. This implies that (1) the entire software stack (operating system, signal processing
software, communication protocols, and applications) must be certified as a whole, and (2) the software must be certified for each type of hardware it intends to couple with. However, this does not mean the software has to be monolithic.

The challenge is to enforce conformance with regulations in a component-based system. The general case of validating a candidate radio configuration to verify that it does not emit forbidden radio signals is not decidable. If such a validation function exists, it could be used to solve the halting problem that is known to be undecidable over Turing machines [17]. The decision problem known as the halting problem can be informally stated as follows:

\textit{Given a description of a program and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting.}

**Theorem 2.3.1.** The general configuration validation problem is undecidable.

\textit{Proof.} We sketch an informal proof here. Let’s construct a program RadioConfig as follows:

\begin{verbatim}
Program RadioConfig
  begin
    Halts();
    RadioEmit ("forbidden signal");
  end;
\end{verbatim}

The RadioConfig program calls Halts() that is a program for which it shall be decided
whether it halts, i.e. the halting problem shall be solved for this program. If \text{Halts()} halts, the program \text{RadioEmit()} is called to emit a forbidden signal, and consequently the program \text{RadioConfig} would not be valid as it emits a radio signal that is not allowed. If, however, the program \text{Halts()} does not halt, then \text{RadioConfig} would not emit a forbidden signal, and therefore it would be a valid radio configuration. As halting problem is undecided, no radio validation program exists.

Thus, we take more practical approach of checking functional properties and interfaces of software modules.

3.1.4 Configuration Attestation

The service provider may request a proof of conformity with the standards before allowing the terminal access to their network. The challenge is to provide a remote attestation scheme enabling the terminal to prove that its activated configuration is in compliance with standards and regulations, and not a rogue or malfunctioning device.

The challenger should be able to determine whether it is: (1) Safe to trust the platform from which the current state information has originated; (2) Safe to trust the software environment running on the platform.

3.2 Research Scope

To clarify the communication layer scope of SDR function, lets take a look at the Open System Interconnection (OSI) reference model. It consists of the following
seven layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application. SDR pushes reconfigurability further down from the network layer. In general SDR functionality lies at the Link layer and the Physical layer.

In this work, we deal with digital signal processing functionality of SDR strictly. Thus we operate strictly at PHY layer, which essentially determines the RF characteristics of a radio. PHY is the layer that is pre-determined at design time in a traditional digital radio. However, in SDR, the PHY layer can be changed to suit the appropriate radio environment, be it GPS or cellular network. The MAC layer is already completely in software, thus any MAC layer protocol can be activated on top of SDR PHY layer, provided the MAC protocol implementation is available on the device.

There are several efforts underway to build MAC protocols for SDRs. Pant et al. [18] implemented a slotted ALOHA protocol with GNU Radio as the PHY. They replaced the MAC and PHY layers with a custom MAC based on slotted ALOHA and PHY using GNU Radio. The MAC and PHY communicate via UNIX domain sockets. Ethernet frames are passed between the network layer and MAC using the TAP/TUN interface. Holger von Malm [19] implemented a pure ALOHA and a send-and-wait ARQ protocol using the GNU Radio framework. The work also evaluates the performance of the system in terms of latency.

It was pointed out that GNU Radio platform’s current architecture makes it extremely difficult to develop MAC protocols for it [20, 21]. Since GNU Radio was designed for signal processing, it does not provide support for maintaining global state. MAC protocols generally maintain a state machine, which is updated by both the transmit and receive paths and is used to coordinate access to the shared medium. Moreover, MAC protocols often need to keep per-flow or per-destination state, e.g. transmission parameters, flow control information, bandwidth use, etc. Finally, the GNU Radio
framework lacks the concepts of time and timers. MAC protocols need support for timers, for example, to implement back off mechanisms, various interframe gaps, or TDMA-style gaps.

In this work, **we do not seek to make the radios composable in real-time.** Instead, we follow the prevalent practice of stopping the radio and loading new code to change the radios performance. Real-time composability, where the ability to swiftly adapt to changing conditions is required, is an important issue related to cognitive radios [22]. Cognitive radio is able to change its transmission or reception parameters to communicate efficiently avoiding interference with licensed or unlicensed users. This alteration of parameters is based on the active monitoring of several factors in the external and internal radio environment, such as radio frequency spectrum, user behavior and network state. Software radio does not have this capability.

### 3.3 Assumptions

We assume a communication situation in a spectrum market, where reconfiguration of base stations and terminals are permitted. The reconfiguration software may be obtained from different sources (e.g. third party software vendors, operators, manufacturers), and reconfiguration can take place at any time.

We assume that some security elements of SDR terminals are enforced in hardware to prevent tampering. SDR standards recommend a hardware security mechanism called Trusted Security Module. TSM provides non-volatile storage for cryptographic keys and certificates [23]. The manufacturer inserts cryptographic credentials into boot ROM during manufacturing; the terminal securely stores the trusted root key of the manufacturer. We can assume that all SDR terminals are equipped with an
asymmetric key pair and a certificate for their public key.

We further assume that some kind of isolation mechanism is provided to protect the execution environment on the SDR terminal.

3.4 Thesis Statement

Valid configuration of Software Defined Radios supporting third party components can be achieved through automated composition of signal processing components. Security and regulations conformity of such devices can be provided through a combination of trusted computing, configuration validation, compatibility check, secure download, and remote attestation.

3.5 Success Criteria

As stated earlier in this chapter, we address a range of problems concerning reconfigurability of software defined radio. The solution to each problem is evaluated based on its feasibility, correctness, and efficiency. In addition, we propose the following criteria to evaluate this thesis:

- Does the composition algorithm scale?
- Is it able to handle configuration profiles where some parameters are missing?
- Does the resulting configured radio function properly?
- Is the XML template for configuration profile and component description expressive enough?
• Is it feasible to prove that composite of certifiable components is also certifiable?

• Does the secure download protocol satisfy integrity and confidentiality properties?

• Does the proposed trusted SDR platform guarantee platform integrity?

• Are the proposed graph compatibility checks practical?
Chapter 4

Configuration Security

We define security in a broad sense as that system attribute that maintains the privacy and integrity of the system and the information distributed across it. It includes mechanisms to ensure accurate content delivery to intended recipients, denial of interception by intruders, rejection of attempts to gain unauthorized access, mechanisms for configuration management of software download, and record-keeping with non-repudiation of actions taken by all participants.

The design of communication system security must consider the potential threats, and trade off the cost of meeting threats with the probability of encountering them and the loss if they are successful. It is also necessary to evaluate the cost of a successful penetration. Interception of a cryptographic key, for example has more far-reaching consequences than retrieving transitory data, such as the current selling price of a stock.

One of our main goals of this thesis is to provide security for SDR configuration. We need to understand the dimensions and objectives of our system security. In this chapter we define the threat model we used to identify and describe security threats. Then we present example of threat scenarios that pertain to a SDR terminal. Finally we identify security requirements for SDR configuration.

We do not attempt to summarize all of the threats and security requirements of the
various wireless standards. Instead, we adopted a model that can be used to classify specific threats to SDR-based communications systems.

4.1 Threat Model

Providers, vendors, and users of mobile wireless systems need assurance that the system will perform the tasks allocated to it without compromise. In general, regarding a system security they expect that a system will:

- **Protect Content** - Most users participate in communications systems because they are interested in the content of the traffic. Different types of content are subject to different threats. Some specific issues to address are privacy, funds transfer, and intellectual property rights.

- **Provide Regulatory Conformance** - Because there is a great deal of contention for use of the electromagnetic spectrum regulators are interested in maintaining control of how it is used, and for certifying that radio equipment meets their emission specifications. Issues to address are: initial certification, delivery/download, and verification.

- **Protect Operating Information** - Operating information is information used by the system, not communication content. Types of operating information: radio parameters, signaling/control, keys and passwords, traffic volumes, user identity and location.

- **Protect Billing and Payment** - The mobile terminal carried by system users has a potential for use in making payment for goods and services. The system security mechanisms must be adequate for accurate billing and payment. Is-
sues to address are reliable payment for services, credit card transactions, and prepaid funds control.

From these security concerns, providing regulatory conformance and protecting operating information pertains most to SDR configuration. Our system ensures that the radio configuration complies with FCC regulations on device emissions and certification requirements for SDR. It also protects configuration information. We discuss the security requirements for SDR configuration in Section 4.3.

We use a four-part model shown in Table 4.1 describe and categorize security threats.

<table>
<thead>
<tr>
<th>Point of Attack</th>
<th>Access Mode</th>
<th>Perpetrator Motive</th>
<th>Security Violation</th>
</tr>
</thead>
</table>

Table 4.1: Security threat model.

**Point of Attack** refers to the device or system component within the communication system where the security breach occurs. It is not necessarily the same as the target of the attack. For example, the target could be terminals operating within a wireless network, whereas the point of the attack could be the network that provides services to those terminals. The following points of attack are considered in this model:

- **Terminal**: The security breach occurs at the handset or other terminal equipment.
- **Infrastructure**: The security breach occurs within the Radio Access Network or Core Network.

**Access Mode** refers to the means by which the perpetrator obtains access to the Point of Attack.
• **Physical**: the threat requires physical control of, or access to, the device or network entity.

• **Remote**: the threat can be perpetrated remotely, by exploiting some external interface to the device or network entity, including wireless interfaces.

**Perpetrator Motive** refers to the motivation of the party responsible for the threatening action.

• **Negligent**: accidentally harmful consequences of a legitimate action. (e.g. the download of authenticated software which contains an unintentional software “bug”).

• **Unauthorized**: unintentionally harmful consequence of an improper or unauthorized action. (e.g. download of unauthorized black market software which is advertised to “boost” handset performance).

• **Malicious**: deliberate, improper action, specifically intended to cause harmful consequences.

**Security Violation** refers to the category of action taken by a perpetrator.

• **Impersonation**: Pretending to make the system think that access is being attempted by a legitimate user. The attack can be further categorized as impersonating a user, impersonating a network, and man-in-the-middle.

• **Unauthorized Access**: Extracting data from the system or injection data into the system by means other than impersonation. These attacks are attempts to get around the normal barriers to unauthorized access rather than dupe them.
The attack can be further categorized as interception, unauthorized access to control, and unauthorized access to data.

- **Denial of Service**: Performing an operation or a series of operations that consume system resources to the extent that performance is reduced. Or finding a way to alter system parameters so that one or more users are wrongfully denied access.

- **Interference with other Services**: widespread performance impairment of, or improper access to, other networks or services.

- **Digital Rights Violation**: Unauthorized access to, or theft of, digital content and software.

### 4.2 Matrix of Threat Scenarios

In Table 4.2 we present several examples of SDR security threats, and illustrate how the four-part model can be used to classify a given threat. We proposed a threat modeling approach for specific types of systems in [24], and identified most of these threats in our earlier work [25]. A wireless base station or handset, employing SDR technologies, should be protected against these threats.

This matrix is by no means an exhaustive list of threats. There are threats such as fraudulent use of network services, masquerading as a communications participant, or traffic analysis that apply to any mobile wireless system but are outside the scope of this thesis. The above identified threats serve as valuable basis to derive security objectives and requirements of our configuration framework described in the next chapter.
4.3 Security Requirements for SDR Configuration

In this section we identify security requirements for the configuration of SDR terminal. Each of these requirements address several of the threats that pertain to SDR configuration, and if properly implemented will effectively mitigate these threats. We surveyed best practices established by security experts [26, 27, 28, 29, 23, 30], and performed our own analysis to compile the list of threats and requirements.

The main requirement, or “the mother of all requirements” is this:

An SDR device MUST provide a secure configuration control.

The individual requirements that apply to SDR configuration and deliberate on the above main requirement are:

- **Configuration integrity**- An SDR device SHALL detect the unauthorized modification of SDR configuration software. This prevents a hacker from changing the device configuration.

- **Regulatory conformance**- An SDR device SHALL ensure that its RF emissions (transmit frequency, power output, modulation format) are limited to those of regulations. It SHALL only install and instantiate SDR-related software that has been appropriately certified to be compliant with FCC regulations.

- **Device attestation**- An SDR device SHALL provide trusted configuration information to its communications service providers on request.
• **Secure download** - A secure infrastructure **MUST** be provided to provide confidentiality and integrity services for download of SDR-related software and configuration data. All software components **MUST** be cryptographically verified before they are executed. At minimum this should be done at start-up.

• **Secure execution environment** - Process separation methodology **SHOULD** be provided to ensure that third-party software components could not access memory or modify operation of the configuration process.

• Private cryptographic keys **MUST** be stored securely. This allows the equipment to securely identify itself.

• Public cryptographic keys (root keys) used to verify certificates **MUST** be stored so that the value cannot be modified.

• **Cryptographic mechanisms** - The cryptographic level of the algorithms used **SHOULD** be consistent with the current state of the art and designed to prevent a dedicated attacker from using weaknesses in the algorithms to modify the specified operation of the equipment.

These security requirements shall be satisfied through suitable functional and security mechanisms in the proposed configuration framework. For some of these requirements we propose our own unique approaches. For others we adopted suitable security mechanisms that already work well for other types of computer systems. We assume that the following requirements are implemented on any SDR terminal:

• The equipment **SHOULD** include a unique non-alterable identifier (Serial Number). This enables certificates to be linked securely to the device.

• Watchdog processes **MUST** be used to insure that processors are executing instructions correctly and that software routines are not locked up.
<table>
<thead>
<tr>
<th>Threat Scenario</th>
<th>Point of Attack</th>
<th>Access Mode</th>
<th>Perpetrator Motive</th>
<th>Security Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Disreputable parties modify device software, causing them to transmit and/or receive on different frequencies, for example a military band, thus enabling covert communications, eavesdropping, or jamming.</td>
<td>Terminal</td>
<td>Physical</td>
<td>Malicious</td>
<td>Interference</td>
</tr>
<tr>
<td>2. A new release of software inadvertently contains a &quot;bug&quot; and is distributed to users in the network. The bug causes terminals to reset unexpectedly, or change its configuration, causing widespread denial of service.</td>
<td>Terminal</td>
<td>Remote</td>
<td>Negligent</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>3. A sophisticated hacker creates and distributes a virus or malicious application that causes widespread interference to the effected or other communication systems, such as public safety, emergency, and navigation control communication systems.</td>
<td>Terminal</td>
<td>Remote</td>
<td>Malicious</td>
<td>Interference</td>
</tr>
<tr>
<td>4. A black market company creates and distributes a rogue application which causes an SDR terminal to deviate from its normal performance limits, and in so doing, causes widespread disruption of service to the effected communication system.</td>
<td>Terminal</td>
<td>Remote</td>
<td>Unauthorized</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>5. An unethical company intercepts software downloaded to phones operating in the network, and illegally re-uses the software to build and sell black market devices.</td>
<td>Terminal</td>
<td>Remote</td>
<td>Malicious</td>
<td>Digital Rights</td>
</tr>
<tr>
<td>6. An unethical company modifies the electronic identifier information on phones intended for sale in one country, and profitably resells the phones in another country where the sale is not legal. (As an example: low cost phones with reduced spectral emission specifications may be legal in one country, but illegal in another country).</td>
<td>Terminal</td>
<td>Physical</td>
<td>Unauthorized</td>
<td>Interference</td>
</tr>
<tr>
<td>7. An unethical company takes in old model phones, illegally reprograms and resells the devices as &quot;new&quot; on the black market. The hardware/software combination of the modified phones is unreliable, and causes the devices to eventually &quot;crash&quot; (i.e. suffer an unrecoverable failure)</td>
<td>Terminal</td>
<td>Physical</td>
<td>Unauthorized</td>
<td>Denial of Service</td>
</tr>
</tbody>
</table>

Table 4.2: SDR security threat scenarios.
Chapter 5

Automated Configuration

Early implementations of SDR terminals were based on the vertical model in which the operational mode is merely switched between two or more independent implementations of air interface standards. Current developments are aiming at truly reconfigurable software architectures, where the goal is to provide open programmable software and hardware platforms and to define any implementation of the terminal purely by software configuration.

In order to rapidly and easily support multiple radio modes within a wide spectrum, software radio uses reconfigurable software programs running on generic hardware and operating system (such as POSIX platforms) to perform radio signal processing. To facilitate the rapidly composable and reconfigurable requirements, SDR waveform applications are commonly architected as composites wired up from software components. This component-based radio architecture facilitates a plug-n-play waveform composition where components can be supplied by third party vendors.

To facilitate a plug-n-play component composition and make the configuration specification portable, we propose an automated configuration methodology for SDR. The core of the methodology is a configuration processor which composes DSP components according to a configuration specification of the desired radio mode.

In this chapter, first I give a schematic view of the configuration process along with
an example scenario in Section 5.1. Then I define DSP component model and composition model in Section 5.2. In Section 5.3 I discuss the model and representation of configuration profile. Finally in Section 5.4 I present the evaluation of the automated configuration methodology.

5.1 Configuration Process

As stated earlier, the automated configuration consists of two main steps. In the first step, the configuration processor composes a functional graph based on a waveform design and configuration policies. The functional graph is stored as a configuration profile in a machine-readable portable format. In the second step, the processor maps the functional graph into an executable graph which connects software components through compatible ports. But this is a very high-level overview of the configuration scheme. In this section, we describe the configuration process in detail and expose the issues we had to address while implementing our framework.

Remember that the first stage of the two-step configuration scheme takes place off-the-terminal. Figure 5.1 displays the flowchart of the configuration process at an SDR terminal. We can describe it as a sequence of the following events:

(1) The user or network provider requests a reconfiguration of the SDR terminal. The request should contain the name or identifier of the desired waveform. For example: **FM Broadcast Receiver**. The request may also provide one or more configuration parameters, such as the frequency band to listen to. These parameters are user preferences and become part of the configuration policy. Since we enforce the most restrictive policy, user preferences will not relax other policies such as the network provider’s policies.
(2) The system tries to retrieve a configuration profile for the requested radio mode. If found, go to the next step. Otherwise the system attempts to download it. At user level applications, download decision can be made in many ways: ask the user, follow preset user preferences, check whether download source is specified and available. Configuration profiles can be downloaded from the network provider, or a trusted vendor. The download channel should be secured, the supplier’s identify should be authenticated, and the downloaded profile should be validated for FCC compliance. We address these issues in the next chapter. Templates are usually generated by radio experts. We discuss profiles in more detail in Section 5.3. After downloading the profile, the configuration process resumes.

(3) The parser parses the configuration profile and generates a functional graph in the radio environment’s address space. The parser is an XML parser. Each node in the functional graph indicates the functionality of the component, i.e. a component class from our classification system. For example: Quadrature Demodulator. All input parameters for components are also specified in the configuration profile.

(4) We can start the composition of the candidate graph. We try to find a matching software component for each functional component, starting from the first node in the dependency chain.

(5) Search the terminal’s repository of software components for a matching component. We only need to parse component descriptors which specify the class of the component, its parameters, and input/output types. We discussed component descriptors in Section 5.2.2. If the software component repository is small enough, it could be preloaded into the system’s memory to speed the search. If we find at least one match whose class, parameters, and input/output ports match the functional component’s class then proceed to the next step. Otherwise, the configuration pro-
cess must terminate since we are unable to generate a complete executable graph that implements the full functionality of the desired radio mode.

(6) At this point we are composing a candidate graph. A candidate graph is similar to the executable graph, except its each node is a set of software components that all implement the same functional component. We will analyze the candidate graph later to derive the best executable graph optimized for the configuration requirements.

(7) Check to see whether configuration policies are loaded into the system. If so, go to the next step. Otherwise, parse and load the policies. Network’s policies should be stored in machine-readable files, e.g. XML. In this stage of configuration, by configuration policies we mean network policies and user preferences. These contain rules and preferences that are enforced in the radio configuration.

(8) Once we have a complete candidate graph, it is time to perform a consistency check. Also the configuration profile has been certified, we check to ensure the correctness of the configuration profile and accuracy of component descriptor. We check whether a component’s input port type is consistent with the output port type of the preceding component. If the consistency check fails, we notify the user or application layer and quite the configuration process.

(9) Some configuration parameters that do not effect the waveform’s emissions might not be specified in the configuration profile. It is up to the user and network to set these parameters, or let the system set default values. The network’s policies might indicate which version of a DSP software to use, which channel to tune to for control information, or what bandwidth width works best in the local environment. There might be very few parameters that the user is allowed to set due to FCC regulations. The user for example, can set the frequency of AM/FM broadcast receivers, spec-
ify which soundcard or peripheral devices should serve as the sink (the last component of SDR execution graph).

(10) If we have a full candidate graph, i.e. it contains at least one component in each set, then select one valid path from the candidate graph. This is the resulting final executable graph. We select the best path based on the configuration parameters. The user might be provided with an opportunity to select a path. Or the system could choose the first valid path found. Presenting all possible candidates to the user might be too costly—the mapping algorithm’s running time would become exponential.

Above we described the process of automatically re-configuring an SDR terminal. We mapped the configuration profile of the requested radio mode into an executable graph, which upon execution realizes a new radio configuration. You will notice that during the configuration process we perform two types of checks: (1) Consistency check to ensure that the configuration’s components are compatible with each other, and (2) check the component descriptions and their parameters against the configuration policies.

5.1.1 Configuration Example: FM Receiver

Configuring FM broadcast receiver is a classic example favored by the GNU Radio community. It has few and simple components and easy to understand for anybody. I refer to this example throughout this thesis. When presenting the processes and models of the secure configuration framework, I explain how it applies to FM receiver configuration. Figure 5.2 illustrates how the composition model presented in Section 5.2.1 applies to this example.

First, the system received a re-configuration request for a user. The desired waveform
type and the FM broadcasting channel to tune to are passed as the arguments of the request command. The user wants to configure the SDR terminal as a FM radio receiver, and wants to listen to the 91.5 MHz station. It is easy to check that the user request is in compliance with FCC rules. FCC allocated the frequency band 88.0-108.0 MHz for FM radio broadcasting, and there is no power limit on the receiver. This is why when experimenting with GNU Radio platform together with an RF front-end hardware, many people use this example. No licensing is required.

The system retrieves the configuration profile of the FM receiver, identifying it by the waveform type. Configuration profiles contain the type of the waveform it configures, and a list of components and parameters in the order they appear in the dependency chain. Here I just show you a graphical representation of the functional graph expressed in the profile. In Section 5.3.3 you can see the XML specification of the configuration profile.

The configuration processor parses the configuration profile, creates a functional graph, and generates candidate executable graphs. After applying the configuration policies such as user preferences, the processor generates an executable graph. You can see in the Figure 5.2 that each functional component is mapped into a software component. For example, the Channel Filter is mapped into a software component called gr.freq.xlating.fir.filter. The match is made based on the software component’s class type, which is Channel Filter. The processor checks whether the filter component’s input/output port types are consistent with that of its adjacent components in the graph. Its input type is short, which is the same as the output type of the RF front-end. Its output type is complex, which is the same as the input type of the quadrature demodulator.
5.2 Modeling Configuration Concepts

The core of our automated configuration methodology consists of a hierarchical configuration process and a configuration processor that drives it. To better present this methodology, several abstract concepts need to be outlined. In this section, I formulate three concepts that play a significant role in our automated configuration strategy. These are component composition model, DSP component model, and mapping of configuration graphs.

5.2.1 Composition Model

The end goal of radio configuration is to compose a pipeline of DSP software modules that process incoming or outgoing signal stream of SDR. In our configuration approach, we compose this pipeline in two main steps. In step one, the configuration processor takes a waveform design of the desired radio mode and configuration policies, and creates a functional graph. We call this functional graph a configuration profile because it makes it possible to port this description to compatible radio platforms and compose an executable DSP chain. The configuration policies are constraints and rules that need to be enforced when parameterizing and validating the configuration profile. The functional graph is a DAG where each node identifies a DSP component by its class and parameters.

In step two, the processor maps the functional graph into an executable graph that activates the requested waveform when executed by the radio runtime environment. The executable graph is a DAG where each node is a software implementation of a DSP component. If suitable software components cannot be found in a local software repository of the SDR terminal, they need to be downloaded from an authorized
software vendor or the network provider. Thus, we can model the configuration composition as a 2-tier hierarchical mapping problem. Figure 5.3 depicts the graphical view of the composition model.

To apply configuration policies in the configuration composition, we need to consolidate them. The configuration policies are consolidated following the principle: *When two or more policies are attempting to control the same functionality or parameter, the most restrictive policy should rule.* For example, a radio equipment is capable to transmit at up to 5 watts of output power, and FCC limits this particular type of radio’s output power to 1 watt when operating in 88-108 MHz frequency band. Then when transmitting in the 88-108 MHz band, this SDR terminal must not exceed its power level more than 1 watt.

A configuration profile specifies the order and type of DSP components used to process a signal stream. Each DSP component applies some kind of mathematical function on the input stream. For example, if the desired radio access technology is GSM, its corresponding configuration profile specifies the order and types of modulators, encoders, and filters used for processing receive or transmit signals. We assume most common configuration profiles come pre-installed from the equipment manufacturer, but they also can be downloaded from an external entity when a need arises. The consolidated policies serve for tuning the input and output parameters of DSP components described in a configuration profile.

5.2.2 Component Model

There are two types of component models we deal with. Both are a part of a waveform, but at a different level. One is the component composing a functional graph,
the other is in an executable graph. They are quite similar, except the functional component does not have a source code. The executable graph’s component is a small software module, called a digital signal processor. Next, we describe the model of this component.

A software component, denoted by \( s_i \), is a self-contained computational unit providing a certain functionality (e.g., modulation, filtering, phase shifting). Figure 5.4 illustrates its graphical model. Each software component has typed input ports and output ports. Each port is associated with a buffer used to dump the output stream or read in the input stream. In other words, each port is typed by the data it accepts or transmits. Each software component consists of (1) a function name \( N_i \), (2) software code \( Code \), and (3) a component descriptor \( CD_i \).

A component descriptor is not necessarily contained within the software component. Rather it is usually implemented as an XML file associated with the component. A component descriptor consists of (a) a component class name \( F_i \), (b) input port types \( T_{in}^i \), (c) output port types \( T_{out}^i \), and (d) parameters it accepts \( P_i = \{p_{i1}, ..., p_{in}\} \). We formally define a software component and component descriptor as follows,

**Definition 5.2.1.** A software component is defined as \( s_i = \langle N_i, Code, CD_i \rangle \), where \( N_i \) represents the provided software function name, \( Code \) defines the function implementation, and \( CD_i \) represents component descriptor.

**Definition 5.2.2.** A component descriptor is defined as \( CD_i = \langle F_i, T_{in}^i, T_{out}^i, P_i \rangle \), where \( F_i \) represents the component class name, \( T_{in}^i \) is input port type, \( T_{out}^i \) is output port type, and \( P_i \) represents a set of parameters it accepts.

Similarly, we can define a functional component.

**Definition 5.2.3.** A functional component is defined as \( f_i = \langle F_i, T_{in}^i, T_{out}^i, P_i \rangle \),
where $F_i$ represents the component class name, $T_i^{in}$ is input port type, $T_i^{out}$ is output port type, and $P_i$ represents a set of parameters it accepts.

5.2.3 Configuration Graph Model

We adopt a synchronous dataflow graph model [31] widely used in signal processing applications to specify the functional and executable graphs of configuration. This model consists of a set of components $S = \{s_1, ..., s_t\}$ and a set of directed edges $E \subseteq S \times S$ such that $s_i \rightarrow s_j$ denotes a data dependency of $s_i$ on $s_j$.

**Definition 5.2.4.** A functional graph is defined as a directed acyclic graph $G_f(F, E)$, where $F = \{f_1, ..., f_n\}$ is a set of functional components and $E \subseteq F \times F$ is a set of directed edges.

**Definition 5.2.5.** An executable graph is defined as a directed acyclic graph $G_e(S, E)$, where $S = \{s_1, ..., s_n\}$ is a set of software components and $E \subseteq S \times S$ is a set of directed edges.

We also define a candidate graph. When the configuration processor is assembling an executable graph based on a configuration profile’s functional graph, often it finds a set of software components that meet the specification. To optimize the decision making without backtracking, the processor first composes a candidate graph, then applies network provider’s or user’s preferences and policies to select the optimal executable graph. The candidate graph may have a set of components at each node.

**Definition 5.2.6.** A candidate graph is defined as a directed acyclic graph $G_c(C, E)$, where $C = \{c_1, ..., c_n\}$ is a set of composite nodes, $c_i = \{s_{i1}, ..., s_{im}\}$ is a set of software components, and $E \subseteq C \times C$ is a set of directed edges.
The problem of mapping a functional graph into an executable graph can be stated as follows:

*Given a functional graph* $G_f$ *and the set of configuration policies such as network policies and user preferences, first compose a candidate graph* $G_c$

*where* $f_i = CD_{ij}$ *for* $\forall f_i \in G_f$, $\forall s_{ij} \in c_i$, *and* $\forall c_i \in G_c$. *Then, select an executable graph* $G_e$ *such that* $G_c \Rightarrow G_e$ *optimizes the configuration requirements.*

Note that the configuration requirements used to compose an executable graph are different from the ones used while composing a functional graph based on FCC rules and hardware specifications. The functional graph expressed in a configuration descriptor is not composed on an SDR terminal. This is due to FCC requirement to certify each software and hardware combination. We explain this in detail in Chapter 6. Configuration profiles are generated by SDR experts, off-the-terminal.

### 5.3 Configuration Profile

In this section, we discuss the concept of configuration profile as it pertains to our configuration framework. A configuration profile serves as basis for composing the right components for the radio. It identifies the components by their class, and specifies the dependency order of these components. It is a blueprint of a radio design, expressed in XML. First we introduce our component classification system, then give a brief overview of XML.
5.3.1 Component Classification

Most software radios use similar hardware for the RF and modem sub-systems. For example, most radio systems include a modulator, an antenna system, a mixer, and an amplifier. From this assumption we can group common hardware and software components into specific classes. From these classes, APIs can be developed to provide a basic function set for the individual classes. Providing the developers of hardware and software components with a component classification system not only promotes the portability of the components by forcing components to provide basic functions, but allows the software radio to clearly identify a basic set of services provided by the component by simply identifying the components classification.

Such classifications include: amplifier, mixer, demodulator, modulator, antenna system, filter, and several others. In our configuration framework on GNU Radio platform, we use more specific classes such as quadrature demodulator, frequency modulator, channel filter, audio filter, low pass filter, RF front-end, and audio sink. We use several of these component classes in our FM Receiver example presented throughout this thesis.

The set of classification types must be a standardized set in which developers must agree to classify their components into. Essentially a simple API is developed for each classification. This API includes only the basic functions and attributes that every component with the specific classification must provide to the radio system. Table 5.1 below provides an example API for an amplifier component classification:

In addition to the control functions the components provide, the attributes that allow external control must have operational limits. As Table 5.1 shows, each controllable attribute has a maximum and minimum limit. These limits must be provided to
<table>
<thead>
<tr>
<th>Identifier</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dutyCycle</td>
<td>attribute</td>
<td>Specifies duty cycle for specific amplifier.</td>
</tr>
<tr>
<td>maxGain</td>
<td>attribute</td>
<td>Specifies the maximum operational gain for specific amplifier.</td>
</tr>
<tr>
<td>minGain</td>
<td>attribute</td>
<td>Specifies the minimum operational gain for specific amplifier.</td>
</tr>
<tr>
<td>gain</td>
<td>attribute</td>
<td>Current operating gain.</td>
</tr>
<tr>
<td>setGain</td>
<td>function</td>
<td>Function provided to set the current gain.</td>
</tr>
<tr>
<td>getGain</td>
<td>function</td>
<td>Function provided to get the current gain.</td>
</tr>
<tr>
<td>getDutyCycle</td>
<td>function</td>
<td>Function provided to get the duty cycle.</td>
</tr>
<tr>
<td>getMaxGain</td>
<td>function</td>
<td>Function provided to get the maximum gain.</td>
</tr>
<tr>
<td>getMinGain</td>
<td>function</td>
<td>Function provided to get the minimum gain.</td>
</tr>
</tbody>
</table>

Table 5.1: API of amplifier class component.

Ensure the hardware components do not operate outside of its operation range which could possibly damage itself permanently. These limits may also be provided to ensure the radio components stay within the legal limits according to the geographical location of the radio system.
5.3.2 XML Overview

We use Extensible Markup Language (XML) in our configuration framework. In our system, the component descriptors, configuration policies, and configuration profiles are expressed in XML. You see the examples of these documents throughout this thesis. In this section we provide a short overview of XML.

XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote.). Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents. Unlike HTML, the set of tags in XML is flexible; the tag syntax is defined by a document's associated DTDs. In fact, XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there is no predefined tag set, there can be any preconceived semantics. All of the semantics of an XML document will be defined by the applications that process them.

The parsing of the XML files into Python is done using a standard Python XML package PyXML, which utilizes eXpat, a widely used open-source XML parser [32].

5.3.3 Template for Configuration Profile

A configuration profile serves as a blueprint in the configuration process. A configuration profile specifies the order and type of mathematical functions used to process a
signal stream. For example, if the desired radio access technology is GSM, its corresponding configuration profile specifies the order and types of modulators, encoders, and filters used for processing receive or transmit signals.

Earlier, in Section 5.1.1 we gave an example of a configuration process. Figure 5.2 displays a schematic view of the FM receiver configuration. The functional graph consists of these components and in that order: RF Front-End, Channel Filter, Quadrature Demodulator, Audio Filter, and Audio Sink. These functional names indicate the class of the component. The configuration profile of the FM receiver must contain this information. A profile makes automated configuration possible, and supports portability of radio designs across various hardware and software platforms for SDR.

A configuration profile must contain at the minimum: (1) type of the radio, (2) functional component list, and (3) configuration parameters and their values. Optionally, parameters that do not have specific values can indicate a range. For some components, specially for components that control a hardware, a name or model of the component can be specified. It is quite simple to represent the functionality of a radio in XML. We predefined a set of tags for this purpose: ConfigTemplate, RadioType, Components, ComponentClass, Parameters, Parameter. Their usage is self-explanatory. Figure 5.5 shows the XML representation of the FM Receiver configuration profile.

There should be a standardized set of radio mode such as FM Broadcast Receiver, AM Broadcast Receiver, GPS Receiver, FM Transmitter, Bluetooth Transceiver, 802.11 Transceiver etc. For some of these types of radios, it is not enough to configure the waveform, which is the PHY layer of the wireless device and deals with raw bits. Changing a waveform or signal processing pipeline only changes how bits are encoded in radio waves. For technologies like Bluetooth and 802.11, it is also necessary to load
and fire up the corresponding MAC layer protocols in order to establish a meaningful communication.

5.4 Evaluation

We implemented the configuration processor on GNU Radio platform. The main control software is implemented in Python and DSP component library is implemented C++. See Chapter 8 for more details on the implementation platform. We developed the automated configuration methodology outlined in Figure 5.1 in Python language. Configuration profiles, DSP component descriptors, hardware specifications, user preferences, network policies, and FCC rules were expressed in XML files.

Evaluation of the automated configuration methodology was performed in two parts: (1) Running time of the composition algorithm is scalable as number of components increases, and (1) Generated composition of components functions properly.

5.4.1 Performance Time

For the first evaluation, we measured the performance time for composing an executable graph based on an XML template. We did not introduce any consistencies with policies or duplicated components. Thus the performance time reflects the time to parse an XML template and components descriptions, search for matching components, and add it to the gnuradio flowgraph. Python uses its powerful reflection capability to reference functions without knowing their name until runtime. This makes it possible to define a new flowgraph at runtime. Figure 5.6 illustrates the configuration time for a graph consisting of 2, 3, 4, 5 nodes.
5.4.2 Radio Functionality

For the second evaluation, we automatically configured an FM broadcast receiver tuned to FM 94.5 MHz radio station. Instead of passing the data to a soundcard, we directed it to the GNU Radio’s spectrum analyzer, \texttt{fftsink}, which uses the fast Fourier transform (FFT) algorithm to extract characteristics of the input signal. Figure 5.7 shows the output of the quadrature demodulator. We see a normal FM radio audio signal with a peak at 19kHz, which is the stereo pilot tone.

The executable graph generated by the automated configuration method can be directly executed by the radio runtime environment since it is already compiled and loaded. Also the system can generate the script of the executable graph and store it in a file to be loaded later if necessary. Below you can view the Python script of the FM receiver configuration.

```python
#!/usr/bin/env python
# Configuration script for FM broadcast receiver.

from gnuradio import gr
from gnuradio import audio
from gnuradio import mc4020
import sys

def build_graph (rf_freq):
    # Set the parameters
    input_rate = 20e6
    fir_decimation = 125
    IF_freq = 5.75e6
    fm_demod_gain = 2200.0/32768.0
    audio_decimation = 5
    audio_rate = 20e6/625

    # Create the flowgraph
    fg = gr.flow_graph ()

    # The signal source is mc4020 microtuner with high-speed ADC
    rf_front_end = mc4020.source (input_rate, mc4020.MCC_CH3_EN | mc4020.MCC_ALL_1V)  
```
# Tell the front end to tune to rf_freq.
rf_front_end.set_RF_freq (rf_freq)

# Select the channel
channel_filter = gr.freq_xlating_fir_filter_scf (fir_decimation,
           IF_freq, input_rate)

# Extract the data signal from the baseband signal
fm_demodulator = gr.quadrature_demod_cf (fm_demod_gain)

# filter out certain audio information
audio_filter = gr.fir_filter_fff (audio_decimation)

# sound card as final sink
audio_sink = audio.sink (audio_rate)

# Connect all the components of the graph
fg.connect (rf_front_end, channel_filter)
fg.connect (channel_filter, fm_demodulator)
fg.connect (fm_demodulator, audio_filter)
fg.connect (audio_filter, audio_sink)

return fg

def main (args):
    if len (args) == 1:
        # get station frequency from command line
        rf_freq = float (args[0]) * 1e6

    # build the flow graph
    fg = build_graph (rf_freq)

    fg.start ()
    raw_input ('Press Enter to quit: ')  
    fg.stop ()

if __name__ == '__main__':
    main (sys.argv[1:])

52
Figure 5.1: Flowchart of the configuration process.
Figure 5.2: Configuring a FM broadcast receiver.

Figure 5.3: Configuration Composition Model.
Figure 5.4: Software Component Model.
<ConfigTemplate domain="SoftwareDefinedRadio">
  <RadioType> FM Broadcast Receiver </RadioType>
  <Components>
    <Component>
      <ComponentClass> RF Front End </ComponentClass>
      <ComponentName> mc4020.source </ComponentName>
      <OutputType> short </OutputType>
      <Parameters>
        <Parameter name="input_rate" type="int">20*10^6</Parameter>
        <Parameter name="rf_frequency" type="int">
          <LowerBound> 88*10^6 </LowerBound>
          <UpperBound> 108*10^6 </UpperBound>
        </Parameter>
      </Parameters>
    </Component>
    <Component>
      <ComponentClass> Channel Filter </ComponentClass>
      <InputType> short </InputType>
      <OutputType> complex </OutputType>
      <Parameters>
        <Parameter name="if_frequency" type="int">5.75*10^6</Parameter>
        <Parameter name="fir_decimation" type="int">125</Parameter>
        <Parameter name="input_rate" type="int">2*10^6</Parameter>
      </Parameters>
    </Component>
    <Component>
      <ComponentClass> Quadrature Demodulator </ComponentClass>
      <InputType> complex </InputType>
      <OutputType> float </OutputType>
      <Parameters>
        <Parameter name="demodulation_gain" type="float">2200/32768</Parameter>
      </Parameters>
    </Component>
  </Components>
</ConfigTemplate>

Figure 5.5: XML template of FM receiver configuration profile.
Figure 5.6: FM radio signal received by GNU Radio.

Figure 5.7: FM radio signal received by GNU Radio.
Chapter 6

Configuration Validation

In Chapter 4 we stated that one of the security requirements for SDR configuration is to provide regulatory conformance. Federal Communication Commission (FCC) regulates radiation emissions by SDR devices [33]. For example, the maximum radiated power of unlicensed radio FM transmitters within 88-108 MHz band cannot exceed 0.1 watts. For traditional digital or analog radio devices, regulatory conformance is not an issue once the device is certified since the device is not capable of changing its RF parameters, therefore will not exceed its normal operating limits.

We consider certification of reconfigurable devices such as SDR, and address this issue within the configuration framework. At the time of writing this thesis, FCC requires each SDR hardware and software combination be tested for certification [6]. This implies that (1) the entire software stack (operating system, signal processing software, communication protocols, and applications) must be certified as a whole, and (2) the software must be certified for each type of hardware it intends to couple with. This severely limits the widespread application of SDR equipment.

We are optimistic that given a reconfiguration methodology with high assurance for conformity, in time FCC might adopt new regulations that fully supports the reconfigurable nature of SDR. We postulate that it is possible to certify a component-based SDR terminal where the operating software, digital signal processing (DSP) software components, and configuration description are decoupled. An open scheme
for conformance validation will allow the entrance of new players to the SDR market.

In this chapter we present a methodology for component-based certification which assures SDR terminal’s conformity based on a separate certification of its hardware, radio operating environment, third party software, and configuration profile. In Section 6.1 we outline the component-based certification process. Then in Section 6.2 we present a secure download protocol based on trusted computing functionality. In Section 6.3 we discuss how to verify regulatory conformance of downloaded software. Finally in Section 6.4 we discuss pragmatic check for detecting software compatibility.

6.1 SDR Certification

Certification is a common technique used to solve trust problems among parties where information asymmetries exist. It is often more efficient than relying simply on market factors such as reputation and warranties. The purpose of certification is to establish a certain level of assurance that a specific product conforms to its specifications.

Equipment to be used in commercial service must pass a set of qualification tests to receive a regulatory certification. Procedures for doing so in the US have recently been changed by the FCC to accommodate SDRs. The current requirement is that each combination of hardware and software must be certified together. The primary justification for mandating joint testing of hardware and software is that this is the only way at the present time to ensure that equipment complies with the technical standards in the FCC rules to prevent interference and to protect users from excessive RF radiation.

Anticipating a widespread push for open architectures, we propose a methodology for
component-based certification. This scheme supports portable configuration profiles and third party components.

6.1.1 Component-Based Certification

We can view a basic SDR terminal architecture as a hierarchy of hardware, platform software, and DSP software. Platform software is the base software or firmware provided by the manufacturer. Platform software consists of an operating system, device drivers, and other critical software such as the configuration processor. DSP software currently running on a terminal consists of DSP software components assembled according to a configuration profile.

In order to certify an entire SDR device for conformance, each of these components need to be certified, and the integrity of the new configuration should be satisfied. Using a pre-certified fully-integrated terminal eliminates the need to test for RF conformance testing since the system is already certified by regulatory agency, only piece by piece. In Section 6.1.2, we assert that this scheme of composable certification is valid with some degree of confidence.

Figure 6.1 gives a simplified view of the component-based SDR certification. We assume that the initial terminal configuration has already been certified by the original equipment manufacturer. Let’s denote this initial configuration \( R_0 = HW\text{SW}_0 \), where \( HW \) is the hardware platform, and \( SW_0 \) is the software platform and initial DSP software configuration. The system is in conformed state. The hardware platform is unalterable, the software platform can be modified only by the equipment manufacturer.

We assume that each SDR terminal is equipped with Trusted Platform Module
Figure 6.1: Component-based certification.

(TPM). The manufacturer performs secure boot before releasing the device. During this boot process the integrity of the device’s initial is measured and store securely. Public keys of the manufacturer, regulatory agency (RA), and Certificate Authority (CA) are also stored in the secure store of TPM. We discuss secure boot in detail in Section 7.2.1.

Whenever a reconfiguration is initiated, the system transitions into a temporary unconformed state. The configuration processor downloads a new configuration profile from a software provider. Several security properties must be satisfied here.

1. The software provider must be authorized by a regulatory agency. Thus the authorization and authenticity of the provider must be checked.

2. The download communication channel must be secured. This protects the integrity and confidentiality of the downloaded software. Details about secure download are given in Section 6.2.

3. Certification of the downloaded configuration profile must be checked. Each configuration profile is certified to comply RF emissions standards on a specific set of hardware, software platform. Platform compatibility, as well as the regulatory
agency’s approval should be checked.

Other download scenarios exist. The downloaded software could be new DSP components. In this case, the security checks are the same. Sometimes the device may reconfigure without requiring any software download. Let’s assume that the device downloaded a combination of configuration profile and DSP components, denoted by $SW_x$. The configuration processor will perform automated composition of components as described in Chapter 5. If the reconfiguration is a success, then the SDR terminal transitions back to conformed state. The radio at time $t$ becomes $R_t = HW(SW_0 + SW_x)_t$.

### 6.1.2 Certifiable by Construction

We proposed a component-based certification methodology for SDR. The end goal is to certify an entire SDR device for conformity. One of the main challenges is posed as a question:

*Given certified components and a certified protocol for assembly, can we deduce that the composite is certifiable?*

The Software Engineering Institute at Carnegie Mellon University is conducting research in *predictable assembly from certifiable components* (PACC). The goal of the work is to achieve predictability by construction. Predictable assembly is an approach for integrating individual software components into a collection of parts where critical run-time properties (e.g., performance, safety, etc.) of that collection are reliably predicted. That is, by using predictable assembly it can be known before the actual components, i.e. software code, are integrated that they will play together with re-
spect to one or more run-time properties of interest [34]. This can be done if the properties of individual software components are known \textit{a priori} to their selection or acquisition.

A component is certifiable if it has properties that can be demonstrated in an objective way. For example, hard disk drive (HDD) manufacturers often provide data sheets that attest to various properties of their products before receiving type approval for the equipment. Same is true for SDR hardware and software. If the assembly of components is well formed, then it is possible to predict the property of the collection. An assembly is well formed if the way components interact with their environment and with each other are made explicit. How well they are formed is checked automatically thus, \textit{assembly behavior is predictable by construction}. The accuracy and reliability of reasoning framework predictions is objectively validated using statistically sound sampling and measurement.

We use the principles of the PACC team’s work to reason about SDR’s component-based certification. In our context, an assembly is a functional graph expressed in a configuration profile. We can define an assembly and well formedness of assembly as such:

\textbf{Definition 6.1.1.} An assembly must have at least two \textit{components}.

Each component except source and sink components must have at least one \textit{output port} and at least one \textit{input port}.

Each input and output port must be connected exactly once.

A component cannot be connected to itself.

An assembly is well formed if and only if all input and output ports are connected.

\textbf{Definition 6.1.2.} An assembly is predictable if it is well formed with respect to the assembly constraints imposed by one or more property theories.
**Definition 6.1.3.** If a configuration profile is FCC certified, then it has been tested for all combinations of hardware and software platforms specified in its template. Furthermore, the profile has been tested for all specified parameter ranges. It produces output signals with the specified characteristics.

**Lemma 6.1.4.** If a configuration profile is certified, then it is well formed and predictable.

*Proof.* From Definition 6.1.3, it has been tested extensively. Then from 6.1.1, it is well formed. And finally from 6.1.2 it is predictable. \(\square\)

Remember that software components stored in a local repository of an SDR terminal are also certified. Thus, we can predict that *if a configuration profile is certified, then the resulting assembly is certifiable.* We do not guarantee that the assembly will succeed. It is possible that some components are missing in the local repository. Kurt Walnau [35] states that the accuracy and reliability of assembly predictions is objectively validated using statistically sound sampling and measurement. FCC regulations require extensive sampling and measurements.

**Definition 6.1.5.** FCC requires that each SDR hardware and software combination must be tested for certification.

**Lemma 6.1.6.** If a configuration profile is certified, then the resulting radio configuration is certifiable.

*Proof.* According to Definition 6.1.3, a certified profile has been tested by the software provider for all hardware and software platform specified in its template. This implies that it has been deployed and tested on the current SDR terminal’s hardware and software platform. Then by Definition 6.1.5, the current SDR terminal is certifiable. \(\square\)
6.2 Secure Software Download

Several techniques exist for downloading software securely on SDR devices [36, 3] that use traditional protocols such as SSL/TLS. We adopted and augmented a protocol based on trusted computing services [37]. The key requirements for secure download of SDR components and configuration profiles are: (1) The software provider must be authorized by a regulatory agency. Thus the authorization and authenticity of the provider must be checked. (2) The integrity and confidentiality of the communication channel must be protected. (3) Software must be checked for standards conformance.

In Section 7.1 we review TPM concepts and services. The Mobile Phone Working group of Trusted Computing Group (TCG) [38] is developing trusted computing standards specifically for mobile devices. Therefore it is reasonable to expect that future mobile devices will be shopping with TPMs.

6.2.1 Protocol Assumptions

We assume the presence of a TPM and a memory protection scheme such as an isolation kernel. This is necessary to ensure the integrity of the downloaded software while at rest, and to protect security-critical processes from downloaded software. Within a protected partition, a trusted configuration processor executes upon a trusted operating system. It is this protected environment into which the SDR software, P, will be downloaded and executed.

We also assume that the SDR device and software provider each have a private signing key that is securely stored. In an SDR terminal, this public signing key
is called Attestation Integrity Key (AIK) and is securely stored by the TPM. The public key for signature verification corresponding to the private key is certified by a certification authority, CA. The certificate issued binds the identity of SDR device or software provider to the public key. The SDR’s certificate must be obtainable by the software provider, and vice versa.

6.2.2 Protocol Description

Figure 6.2 illustrates the flow of the protocol. We describe the protocol step by step.

1. C −→ S: Request for P

The protocol begins when the configuration processor C makes a request for a specific SDR software, P, from a software provider S.

2. S −→ C: R_S

The software provider sends a random number R_S to the device to prevent message replay.
3. C $\rightarrow$ TPM: TPM\textsubscript{CreateWrapKey}

The configuration processor asks TPM to generate an asymmetric encryption key pair $C_{public}$ and $C_{private}$ by invoking a TPM command TPM\textsubscript{CreateWrapKey}.

4. TPM $\rightarrow$ C: TPM\textsubscript{Key}

In response to the TPM\textsubscript{CreateWrapKey} command, the TPM returns a TPM\textsubscript{Key} data structure. This data structure contains $C_{public}$ and encrypted $C_{private}$. The data structure also contains integrity metrics $I$ such that the private key can only be utilized by the TPM on which it was generated when the TPM host platform is in the specified state. Specifically, the integrity metrics include Platform Configuration Register (PCR) digests at key creation and the PCR digests required for key release. The PCR digest at creation reflects a trusted execution environment which consists of a correctly functioning configuration processor running on a trusted radio platform.

5. C $\rightarrow$ TPM: TPM\textsubscript{CertifyKey}

The handle associated with the key pair is given to the TPM in a TPM\textsubscript{CertifyKey} command. The keys and integrity metrics are certified by the TPM using its Attestation Identity Key (AIK) so that the state to which the private key is bound can be shown to the software provider. 160 bits of externally supplied data, which in this protocol is a one way hash of $R_S$ and $Id_S$, may also be given as an input parameter to this command. $R_S$ is a random nonce sent by the software provider, and $Id_S$ is its identity.

6. TPM $\rightarrow$ C: TPM\textsubscript{CertifyInfo} $\parallel$

$D_{TPM}(H(C_{public}) \parallel H(R_S \parallel Id_S) \parallel I)$

In response to the TPM\textsubscript{CertifyKey} command, the TPM returns TPM\textsubscript{CertifyInfo} data structure. This data structure describes the key that was created and how the PCR data is used. In addition, the TPM signs the hash of the $C_{public}$ key.
digest and a concatenation of external data $R_S$ and $Id_S$. $D_{TPM}$ is the digital signature of data computer using TPM’s private signature transformation.

7. $C \rightarrow S$: $R_S \parallel Id_S \parallel TPM_{Key} \parallel TPM_{Certify} \parallel Info$
   \[ \parallel D_{TPM}(H(C_{public}) \parallel H(R_S \parallel Id_S) \parallel I) \]

The certified public key and the corresponding AIK credential are then sent to the software provider. The service provider $S$ verifies the signature on the data received, checks $R_S$ to ensure the message has not been replayed and $Id_S$ to ensure that the message was destined for $S$. Assuming that everything is correct, $S$ then verifies the integrity metrics, $I$. If $I$ describes a trustworthy platform, then $S$ generates $K_{1SC}$ used for data encryption, and $K_{2SC}$ used for data integrity protection.

8. $S \rightarrow C$: $E_{C_{public}}(K_{1SC} \parallel K_{2SC}) \parallel$
   \[ D_S(E_{C_{public}}(K_{1SC} \parallel K_{2SC})) \parallel\]
   \[ E_{K_{1SC}}(MAC_{K_{2SC}}(D_{RA}(P \parallel A))) \]

On receipt of the above message, the configuration processor verifies the digital signature $D_S(E_{C_{public}}(K_{1SC} \parallel K_{2SC}))$, and if it is valid, instructs the TPM to decrypt $E_{C_{public}}(K_{1SC} \parallel K_{2SC})$. The configuration processor then decrypts $E_{K_{1SC}}(MAC_{K_{2SC}}(D_{RA}(H(P))))$ and verifies $MAC_{K_{2SC}}(D_{RA}(P \parallel A))$. $D_{RA}(P \parallel A)$ is the requested software and its conformance approval information digitally signed by a regulatory agency. Once the MAC and conformance are verified, the software can be used on the SDR terminal. The software should be encrypted while stored on the terminal.

Because software download may cause many problems, it must be subject to intense scrutiny and protection. Authorization to operate must be derived from an appropri-
ate Regulatory Agency and be carefully certified. Rogue software in a terminal is a major point of vulnerability. In the next section we discuss how downloaded software can be verified for conformity.

6.3 Conformance Validation

In order to ensure that an SDR device is compliant with RF emissions regulations, we proposed a component-based certification scheme described in Section 6.1. Assuming that the configuration profile and its component software are certified for compliance and assuming that the configuration processor and radio platform are independently certified for conformance, we can expect the entire SDR device is compliant with standards and regulations.

The software and hardware platform do not change, or change only under the manufacturer’s control. The changing part is DSP components and configuration profiles. In the previous section we discussed how to provide secure software download. The integrity and confidentiality of the downloaded software in transit was protected. However, before activating any downloaded software, we must verify whether the software is approved by a Regulatory Agency for compliance, and whether the software is compatible for a particular radio platform.

Certification of compliance for a software is issued by an approved Regulatory Agency. In the United States, such an agency is FCC. Each software or hardware must be extensively tested in a lab for all possible scenarios in order to be certified. A regulatory agency’s approval is expressed as a digital signature attached to a software.

Conformance Certification Format
When issuing a conformance certification, the regulatory agency digitally signs software content and accompanying approval information. Approval information includes the following parts:

- Software serial number
- Hash of software content
- Approval ID
- Regulatory agency ID
- Validity period
- Compatible hardware platforms
- Compatible software platforms

The digital signature is produced by first computing a hash of software content and approval information. Cryptographic hash function such as MD5 or SHA-1 can be used. Then the hash value is encrypted with the private key of the regulatory body. The public key is distributed in a form of a certificate. The regulatory agency’s public certificate is obtainable by SDR terminals.

When downloading a new software, an SDR terminal must also download the approval information. In the previous section we saw that in the final stage of the download protocol the configuration processor receives $E_{K1SC}(MAC_{K2SC}(D_{RA}(P \parallel A)))$. Once decrypted a symmetric encryption key $K1_{SC}$ and MAC is verified we are left with $D_{RA}(P \parallel A)$ - a digitally signed software content and its approval information. Conformance verification can be conducted by:

(1) verify regulatory agency’s signature by applying it’s public key, (2) verify whether
the hash value of the software content is valid, (2) verify whether validity period is up-to-date, (3) verify whether software/hardware platform information is compatible with the host terminal’s.

6.4 Compatibility Check

The general problem of verifying whether the execution of a piece of radio software would lead to non-conformant radio emissions is not decidable, as it would solve also the halting problem [39]. We can apply a set of pragmatic checks with the objective to detect and prevent reconfiguration attempts that would fail later.

There many possible check of different complexities exist that can be performed as part of configuration validation. Some examples are:

- Comparison of the identifier of the target execution environment indicated as part of the software component’s descriptor or a configuration profile.

- Verification of a XML-based configuration to be a valid XML document. Optionally to validate whether it fulfills its XML DTD (Document Type Definition).

- The software can be scanned, to check whether it contains API calls that are not supported by the target platform. For example in GNU Radio, use reflective capability of Python to query object class interfaces.

- Empirical data (log data) can be examined indicating whether a software module has been successfully installed and executed on devices of the same type.
The download and execution can be simulated on a device simulator; occurring exceptions or failure messages indicate incompatibility. Also there is a possibility to digitally analyze the spectrum of the signal in the terminal. This could be realized using powerful functions such as Fast Fourier Transforms.

6.5 Evaluation

We performed informal analysis of the secure download protocol with respect to the security objectives: protecting the confidentiality and integrity of software as it is transported from the software provider to the host platform.

Confidentiality: Symmetric encryption is deployed to protect the confidentiality of the software P. The symmetric key is securely transported to the SDR terminal under a public encryption key of an asymmetric key pair. This key pair is inextricably linked to the requesting trusted SDR terminal where the private decryption key is securely stored. Assuming that the keys are securely managed by the software provider, an attacker cannot intercept and gain read access to the software unless there is a hardware-based attack on the mobile host. This would require the extraction of the private decryption key from the TPM.

Integrity: A message authentication code is deployed to protect the integrity of software P in transit. The MAC key is protected by the same technique used to protect the symmetric encryption key, described above. Thus, assuming secure MAC algorithms are used, integrity protection depends on the security of the tamper proof TPM.

Entity authentication: The software provider can verify the identity of the TPM and
the state of the protected execution environment which utilises TPM security ser-
vices, via the platform attestation mechanism. That is, the signature of the TPM on
$R_S$, $Id_S$ and on the integrity metrics representing the protected execution environ-
ment identifies the platforms state and allows the software provider to authenticate
the trusted platform. the public key of the terminal, $C_{public}$, serves as the nonce
in the response message sent by the radio terminal by virtue of the fact that the
asymmetric key pair is generated for each protocol run. It may be argued that the
protocol outlined above also provides entity authentication with respect to the soft-
ware provider. Since a unique $C_{public}$ is generated for each protocol run, $C_{public}$ acts as
not only a random nonce but also represents the identity of the destination platform.
The signature of the software provider on the unique public key, $C_{public}$, provides
entity authentication.

**Origin authentication:** Since S signs $K_{1SC}$ and $K_{2SC}$, C is able to verify that these
keys have been sent from S. Also, since $K_{2SC}$ is used to compute the MAC, C can
thus verify that the P has been sent from the same source. An attacker attempting
to deliver a malicious application would require the collaboration of S.
Chapter 7

Configuration Attestation

We stated in the security requirements for SDR configuration that the device shall provide trusted configuration information to its communications service providers on request. The main reasons a service provider may request this information is first, to ascertain that the terminal conforms with the emissions regulations of the region, and second, to check whether the terminal’s operating mode is appropriate to receive the service.

Remote attestation is a technique for ascertaining the operating state of a radio platform to a remotely located party in a secure manner. The platform software that manages and controls the device configuration should be trusted, and be able to prove its trustworthiness to the remote party. Only in this condition, the remote party can conclude that the configuration information provided by the device is a true reflection of its current state.

Trusted platform software is also needed for validating device’s conformance with regulations, the technique described in the previous chapter. Composable certification is possible only if the configuration processor and the execution environment can be trusted. We use Trusted Computing concept to ensure the integrity of radio platform.

In this Chapter first we briefly review trusted computing concepts in Section 7.1. Then in Section 7.2 we describe what specific TPM functionalities we use to provide
a trusted SDR platform. In Section 7.3 we present remote attestation service for SDR.

7.1 Trusted Computing

In this section we give a brief overview of trusted computing concepts and services. In the context of trusted computing, a platform is trusted if it behaves in an expected manner for an intended purpose [38]. This does not necessarily imply, however, that a trusted platform is secure. The platform can be, for example, infected with a virus, but this effect should be detectable.

Trusted computing, in its original specification, does not provide a secure platform that prevents malicious or accidental modification or addition of downloaded software, and prevents malicious or buggy software being downloaded to and executed on a device. However, trusted computing functionality can be used to isolate security-critical software in a secure execution environment so that it cannot be observed or modified when executing in parallel with insecure execution environment. We elaborate on securing the execution environment in Section 8.3.

Nor does TPM provide secure boot mechanism in TCG’s main specifications. However, the Mobile Phone Working group [38] is developing trusted computing standards specifically for mobile devices, and have specified several primitives for Mobile Trusted Module (MTM). We use MTM primitives to devise a secure boot sequence in Section 7.2.1.

In order to implement a platform of this nature, a trusted component, which is usually in the form of built-in hardware, is integrated into a computing platform. This
trusted component is then used to create a foundation of trust for software processes running on the platform. Trusted computing is built upon four fundamental concepts: integrity measurement, authenticated boot, platform attestation, and sealing. The trusted component which provides these services is comprised of three so-called roots of trust: the Root of Trust for Measurement (RTM), the Root of Trust for Storage (RTS), and the Root of Trust for Reporting (RTR). A root of trust is defined as a component that must be unconditionally trusted for the platform to be trusted.

In the trusted platform specified by the Trusted Computing Group (TCG), the Trusted Platform Module (TPM) acts as the RTS as well as the RTR whereas the Core Root of Trust for Measurement (CRTM) is most often part of the BIOS. A TPM is generally implemented as a chip which must be uniquely bound to a platform. This hardware component provides four major classes of functions: 1) Cryptographic functions, 2) Secure storage and reporting of hash values representing a specific platform configuration, 3) Protected key and data storage, and 4) Initialization and management functions.

### 7.1.1 Trusted Computing Services

**Integrity Measurement**

An integrity measurement is defined in [40] as the cryptographic digest or hash of a platform component. For example, an integrity measurement of a program can be calculated by computing the cryptographic digest or hash of its instruction sequence, its initial state (i.e. the executable file) and its input.

**Authenticated Boot**
An authenticated boot process represents the process by which a platform's configuration or state is reliably measured, and the resulting measurement is reliably stored. During this process, the integrity of a pre-defined set of platform components is measured, in a particular order. These measurements are condensed to form a set of integrity metrics which can then be stored in a tamper-resistant log. Condensing enables an unbounded number of platform component measurements to be stored. If each measurement was stored separately it would be difficult to decide on an upper bound on the size of memory required to store them. A record of the platform components which have been measured is also stored on the platform.

Attestation

Attestation is the process by which a platform can reliably report evidence of its identity and its current state (i.e. the integrity metrics which have been stored to the tamper resistant log, and the record of the platform components which have been measured).

Sealing

Sealing represents the process of associating data with a set of integrity metrics representing a particular platform configuration, and encrypting it. The data can only be decrypted and released when the state of platform is the same as that indicated by the integrity metrics sealed with the data.
7.2 Trusted SDR Platform

In this section we present how trusted computing functionality is used to provide a trusted software platform for SDR. Trusted platform is the basis for providing other security services for SDR. In Section 6.2 we describe a software download protocol that leverages trusted computing functionality. In the next section we discuss a remote attestation methodology which also uses trusted computing services.

The initial configuration of an SDR terminal, i.e. hardware and software combination, should be certified by the original equipment manufacturer. The integrity of platform software must be provided at all times. A secure boot process can be used to ensure that a set of security-critical platform components boot into the required state. Secure boot is not defined in the Trusted Computing Group’s TPM main specifications. However, the Mobile Phone Working group of TCG [38] is developing trusted computing standards specifically for mobile devices. So far they have specified several primitives for Mobile Trusted Module (MTM). We use MTM primitives to devise a secure boot sequence.

7.2.1 Secure Boot

In principle, during secure boot process the integrity of a pre-defined set of system components is measured, and these measurements are then compared against a set of expected measurements which must be securely stored and accessed by the platform during the boot process. If, at any stage during the boot process, the removal or modification of a platform component is detected, the boot process is aborted. Now we outline how to implement a secure boot mechanism using the primitives defined in the Mobile Trusted Module specifications [41]. First we introduce some terminology
related to Mobile Trusted Module.

**PCR** - Platform Configuration Registers are used by the Root of Trust for Storage to store the platform’s integrity metrics.

**RTV** - Root of Trust for verification.

**RIM** - Reference Integrity Metrics.

Let \( state = [(i, v_1), (j, v_2), ...] \) denote a set of PCRs such that PCR with index \( i \) holds the value \( v_1 \) and PCR with index \( j \) holds the value \( v_2 \) and so on.

Denote by \( SHA1(x) \) the SHA1 hash over the byte-string \( x \), e.g. \( SHA1(imgOS) \) is the SHA1 hash of the operating system \( imgOS \).

**TPM\_Extend** command modifies the current PCR digest relative to the input 20-byte digest.

An event can be for example the loading of a software image and that event can be represented by a SHA1 hash of that image.

Denote by \( TPM\_Extend(state, index, x) \) the result of extending state (a set of PCRs as described above) with the event \( x \) into PCR index.

A **RIM Certificate** (RIM Cert) is an authenticated and integrity protected structure containing a RIM and some auxiliary information.

Let \( RIM\_CertK(state, index, event) \) denote a TPM\_RIM\_CERTIFICATE instance signed by key \( K \) authorizing an extend of event into PCR index when the PCRs already contain the values represented by state.

**TPM\_VerifyRIMCertAndExtend** command is used to verify and to extend the RIM given in the RIM certificate in to a PCR given in the RIM certificate.

Let us assume the boot sequence consists of three software executables that must be loaded and executed in a defined order: \( imgOS \) and \( imgSDR \). For an SDR terminal, the operating system image (\( imgOS \)) includes critical operating services such as a configuration processor. \( imgSDR \) represents an initial set of DSP software deployed on the terminal. Denote these can be provided by independent developers who do NOT have access to each others images. As an additional requirement is that an update to \( imgOS \) must not require any additional actions by the supplier of \( imgSDR \).

The system has at least the following states:

- \( state_0 = [(0, 0), (1, 0), ...] \)

- \( state_1 = TPM\_Extend(state_0, 0, SHA1(RTV\ done)) \)

- \( state_2 = TPM\_Extend(state_1, 1, SHA1(imgOS)) \)
- $\text{state}_3 = \text{TPM}_{\text{Extend}}(\text{state}_2, 0, \text{SHA1(imgOS loaded)})$

- $\text{state}_4 = \text{TPM}_{\text{Extend}}(\text{state}_3, 2, \text{SHA1(imgSDR)})$

- $\text{state}_5 = \text{TPM}_{\text{Extend}}(\text{state}_4, 0, \text{SHA1(imgSDR ready)})$

The state $\text{state}_0$ represents the initialization of all PCRs to zero. For each of the states $i > 0$, we have a corresponding $\text{RIM\_Cert}_i = \text{RIM\_CertRoot(}$state$_i, ...$)$ that authorizes the extend into $\text{state}_i$ from the preceding state. It is also assumed that PCRs 0, 1, 2, and 3 are verified PCRs, i.e. they can only be extended using $\text{TPM\_VerifyRIMCertAndExtend}$. Here is the sequence how the boot would proceed through the above-mentioned states.

1. The trusted module starts up by having $\text{TPM\_Init}$ and $\text{TPM\_Startup}$ being called.

2. All PCRs are initialized with the value 00..00.

3. The RTV records into PCR 0 a SHA1 hash of the string ("RTV done") using $\text{TPM\_VerifyRIMCertAndExtend}$ command and $\text{RIM\_Cert}_1$.

4. Next the RTV measures imgOS and looks up a RIM Cert for it. It should find $\text{RIM\_Cert}_2$ for it.

5. The RTV calls $\text{TPM\_VerifyRIMCertAndExtend}$ for $\text{RIM\_Cert}_2$.

6. Control is then passed to imgOS.

7. imgOS extends into PCR 0 a SHA1 hash of the string (imgOS loaded) using $\text{RIM\_Cert}_3$.

8. imgOS then measures imgSDR and looks up $\text{RIM\_Cert}_4$. 
9. imgOS calls TPM.VerifyRIMCertAndExtend for RIM_Cert_4.

10. Control is then passed to imgSDR.

11. SDR extends finally a SHA1 hash of the string (“SDR ready“) into PCR 0.

This boot sequence for a trusted SDR platform provides the following advantages:

- The secure boot configuration is protected against tampering.

- Any component (imgOS or imgSDR) of the secure boot chain can be updated, without updating the RIM certificates of the following components. This is due to the ability of using PCR 0 as a pre-requisite in the TPM.VerifyRIMCertAndExtend calls.

- Multiple execution paths of the secure boot are possible.

- RIM certificates for imgOS and imgSDR can be produced independently of each other, as long as the platform integrator has fixed and published the strings being extended into PCR 0.

- Boot configuration can be managed remotely, by adding new RIM certificates. This is useful for manufacturers in updating the firmware.

### 7.3 SDR Remote Attestation

Remote attestation is a technique for ascertaining the operating state of a radio platform to a remotely located party, e.g. service provider, in a secure manner. The main reasons a service provider may request this information is first, to ascertain that
the terminal conforms with the emissions regulations of the region, and second, to check whether the terminal’s operating mode is appropriate to receive the service.

In general, trusted computing uses two concepts to implement remote attestation functionality: Platform Configuration Registers (PCR) and Attestation Identities (AI). An Attestation Identity is embodied in a credential, and usually certified by a trusted third party to attest that the owning mobile terminal contains a live, unaltered MTM (Mobile Trusted Module). Producing the AI credential together with PCR values and measurement logs and signed by the AI private key to an external verifier, constitutes the attestation process proper.

Due to chained secure boot process, remote attestation is going to be much simpler and straightforward for an SDR terminal. The AIK (Attestation Identity Key) credentials are associated either implicitly or explicitly with an Root Verification Authority Identifier (RVAI). This RVAI is the root verification key loaded using TPM_LoadVerificationKey that is used to authorize all TPM_RIM_Certificate instances accepted by TPM_VerifyRIMCertAndExtend extending PCR 0.

If a remote verifier is provided with the public part of the RVAI key then the remote verifier can merely check the AIK signature, the AIK credentials, whether it trusts the key RVAI and the contents of PCR 0. The reason why PCR 0 is the only PCR necessary to check is that all extends to PCR 0 have been authorized by the RVAI and the events recorded into PCR 0 translate the events (e.g. extensions of the SHA1(imgOS) in the other PCRs into well-known bit-strings, e.g. SHA1(“imgOS loaded”). The remote verifier would NOT need to be aware of all the multitudes of configurations that are legitimate, it can instead trust a list of verification keys that are used to authorize TPM_VerifyRIMCertAndExtend operations.
During remote attestation of SDR device, the challenger (e.g. service provider) sends a nonce. The SDR platform signs the nonce in conjunction with integrity metrics reflecting the current state of the platform, using one of its private AIKs. This signed bundle is returned to the challenger with the record of the platform components which are reflected in the integrity metrics, together with the appropriate AIK credential. The challenger then uses this information to determine whether it is: (1) Safe to trust the platform from which the statement has originated; (2) Safe to trust the software environment running on the platform. The integrity metrics include the digest of the current configuration, as well as the digest of encrypted source of the configuration profile.

7.4 Evaluation

We experimented with an open source TPM emulator [42], which runs as Unix daemon process that can be controlled using Unix domain socket. GNU Radio platform provides socket interfaces, which were used to connect to the emulator. The TPM emulator has its own small cryptographic library that is based on GNU Multiple Precision Arithmetic library. It provides most of the basic TPM functionalities for generating and storing keys, measuring process integrity, and binding keys to integrity measurements.
Chapter 8

Design and Implementation

In this chapter we describe the design and implementation issues of the secure configuration framework. First we define functional requirements for the configuration framework in Section 8.1, and then present the overall design architecture of the framework in Section 8.2. Then in Section 8.3 we discuss our experiment in developing a mechanism for secure execution environment for GNU Radio. And finally in Section 8.4 we describe the software and hardware platform we used for building the system.

8.1 Functional Requirements

Our configuration framework will ensure secure and dynamic configuration of SDR terminals. It is responsible for the coordination of the configuration processes. Functional requirements of the SDR configuration framework are as follows:

- It shall support various protocols and protocol features, capable of dynamic insertion, replacement and configuration of protocol components from different vendors.
- It shall control and coordinate the reconfiguration of various equipment components.
• It shall support flexible configuration policy mechanism whereby the rules change depending on the preferences of the network operator, service provider, or end user.

• It shall implement local repository for configuration data and reconfigurable software modules.

• Each reconfigurable software module shall be accompanied by a descriptor tag describing its functionality and interfaces.

• Configuration setup shall be expressed in robust manner and be able to interface with descriptor tags of reconfigurable software modules.

In Chapter 4 we presented our security threat model and security requirements for the configuration framework. These functional and security requirements are satisfied through suitable mechanisms in the configuration framework.

8.2 Architecture

Figure 8.1 illustrates the system architecture highlighting its functional units, main resources, and inter-component interactions. The configuration framework consists of the following modules:

• **Execution Environment**- A platform for executing all equipment functions including configuration management and control. The challenge for SDRs is to ensure that users and applications cannot access nor interfere with the flow of information at a higher security classification such as configuration. To support multiple levels of security on a single processor, a secure partitioning method
is needed. We propose a secure Memory Management Unit using hardware-enforced memory protection to ensure data isolation in different partitions.

- **Security Module**- This module ensures that all the security requirements specified for the configuration process are satisfied. It provides the basic security functions to all other configuration modules. For example, it provides authentication function to the Memory Management Unit when a DSP module attempts to access the shared memory buffer of the flow graph. So far, we have specified the security properties of the configuration framework in informal way, using descriptive threat model and requirements. For future work, we would like to use formal security models to demonstrate the feasibility of our security approaches and to claim that the security of the system is provable.

- **Configuration Management Module (CMM)**- A functional entity responsible for the management of all configuration tasks. It initiates, coordinates, and performs configuration functions, and manages the communication between all
configuration related components. It supports such tasks as mode selection, download of configuration policies and software modules, and approval of new radio configurations.

- **Configuration Control Module (CCM)** - A supporting entity for CMM, controls and supervises reconfiguration execution. The selected and verified configuration policy is handed over to the CCM for construction of a flow graph composed of DSP modules specified in the policy. The flow graph is then executed by the runtime environment activating the requested radio operating mode. This module also ensures that the new configuration is in compliance with regulatory requirements before executing the configuration.

- **Policy Management Module (PMM)** - Provisions a configuration policy for a new configuration approved by the CMM. Parses and verifies downloaded configuration policies, manages the update and versioning of the local policy repository. We propose to use XML interface to describe configuration policies, which in turn require XML descriptors for reconfigurable software modules such as DSP modules.

- **Configuration Attestation Module (CAM)** - Software attestations will enable a SDR device to prove to an external party that it is configured properly. This module provides trusted configuration information to the service provider upon request.

Other modules within the framework are local repositories of configuration policies and reconfigurable software modules. The repository of software modules containing DSP modules and link protocols is not strictly a part of the configuration framework. However, these software modules are the main target of the configuration process as it
composes these modules into a flow graph according to the appropriate configuration policy to activate a particular radio operating mode.

8.2.1 Configuration Management Module

To ensure the proper operation of the radio device that not only accommodates the hardware specifications, but also adheres to government regulations, the configuration process must be protected, and only authorized entities can trigger a new configuration and only authorized modules can provision, verify, and enforce the configuration policy.

We propose a Configuration Management Module (CMM) responsible for the management of all configuration tasks. It initiates, coordinates, and performs configuration functions, and manages the communication between all configuration related components.

We shall define the capabilities of CMM as follows:

- It knows the hardware capabilities of the radio system (e.g., RF, IF parameters).
- It monitors requests for a new radio configuration (e.g., FM receiver).
- It ensures that the Policy Management Module provisions a new configuration policy matching the request.
- Upon receiving a valid and correct configuration policy from PPM, it ensures that the Configuration Control Module constructs a flow graph with according to the provided configuration policy.
- It executes the flow graph and activates a new radio operating mode.
8.2.2 Policy Management Module

With the increasing agility and programmability of radio hardware, both the number of modes of operation as well as the range of operating environments for the radio will increase tremendously. As a result, the number of different policy sets that apply to these various modes and environments will grow in a combinatorial fashion. This will make it impractical to hard-code into the radio discrete policy sets to cover every case of interest. We need a more scalable way to express and enforce policy. An expressive policy language is necessary so that regulators can encode policies in one language and all SDR radios understand the encoded policy.

Reconfigurable software modules should have standardized interfaces such as metatags, the methods by which they can be identified and used in the installation and instantiation of these software. Specifically, they would have XML descriptor files that describe the functionality, credentials, interfaces, parameters, and dependencies of the software module.

The Policy Management Module downloads policy files specific to the geography, hardware, and functionality. It verifies that policy files come from a trusted source/vendor. Then it parses the policies expressed in machine-readable policy language to translate them into an executable flow graph. A configuration policy describes which software modules and in what manner compose a new configuration. Once a flow graph is constructed, it is instantiated by the Configuration Control Module.
8.2.3 Configuration Attestation Module

In general, remote attestation provides the capability to know with certainty the hardware and software configuration of a remote host. This type of information is important because many host vulnerabilities are tied to specific host configurations. It allows changes to the computer’s configuration to be detected by the user and others.

In SDR, remote attestation enables a SDR terminal to prove to an external party that it is running approved configuration software and the configuration itself is valid. This proof could be used to authorize access to network services. To this end, we propose a Configuration Attestation Module (CAM) that certifies to network providers that the SDR device has been configured properly.

There are two things that CAM must prove: a) the SDR terminal is running a trusted configuration software; b) the radio configuration is valid and proper. For the first part, Trusted Platform Modules (TPM) specified by Trusted Computing Group might be a proper solution. TPMs [43] enable two new security features, attestation and sealing. Attestation allows a host to verify securely the configuration of a remote host. Sealing enables storing data in a host such that the information can be retrieved only when the host is in the same safe configuration.

To prove that the current radio configuration is valid to third parties, CAM itself could certify its correctness based on the assumption that CAM is trusted. The third party already would have verified the trustworthiness of CAM using TPMS and public keys. SDR standards specify that SDR devices shall contain a physically protected Trusted Security Module, similar to TPM [23].
8.3 Secure Execution Environment

Malicious or malfunctioning software presents serious security risks to SDR devices in which they operate. Memory management can be an extremely effective security measure to guard against surreptitious attempts to modify installed software or any attempts to bypass the normal installation mechanisms.

Support for multiple independent levels of security on a single processor can be provided by an operating system and processor capable of secure partitioning. A secure partition is effectively a virtual computer that shares a processor but has its own tasks, dedicated resources and view of memory. To support this feature, the operating system and hardware must:

- Ensure that data in one partition can never be accessed, compromised or otherwise deduced or detected by a task in a less secure one.

- Ensure that an error or deliberately malicious behavior in one partition cannot compromise processing in other partitions or the system as a whole.

- Guarantee that each partition has the resources it needs to execute successfully. This encompasses processor bandwidth, physical resources such as memory and kernel objects such as semaphores.

- Ensure that a fault caused by a task in one partition whether accidental or deliberate cannot affect another partition.

Memory Protection Techniques

Several types of protection techniques exist to prevent buffer overflow such as dynamic runtime check, address protection, and software fault-isolation. These mechanisms
are implemented as stand-alone system software, kernel extensions, compiler extensions, and loader extensions. Although no mechanism can prevent all unauthorized attempts to access memory, these mechanisms along with good programming techniques will help to provide the best possible solution for such attacks [44].

Dynamic runtime check primarily relies on the safety code being preloaded before a software component is executed. This preloaded component can either provide safer versions of the standard unsafe functions, or it can ensure that return addresses are not overwritten. Source code of the software component is not needed. LibSafe is an example of such solution; it protects a process against the exploitation of buffer overflow vulnerabilities in process stacks = [45]. LibSafe intercepts all calls to library functions that are known to be vulnerable. A substitute version of the corresponding function implements the original functionality, but in a manner that ensures that any buffer overflows are contained within the current stack frame.

Compiler tools allow bounds checking to go into compiled code automatically, without changing the source code. These compilers generate the code with built-in safeguards that try to prevent the return address from being overwritten, as most attacks occur this way. StackGuard detects and defeats smash stacking attacks by protecting the return address on the stack from being altered [46]. It places a canary word next to the return address whenever a function is called. If the canary word has been altered when the function returns, then some attempt has been made on the overflow buffers.

Software-based fault isolation techniques modify the machine code of a program during load time to instrument all critical accesses such as memory read/write, jumps, calls and returns to point to valid and allowed addresses. Sandboxing is an isolation technique that inserts a code before every unsafe instruction. This code sets the upper bits of the target address to the correct segment identifier to ensure that the
address falls in the logically separate portion of the software component within its address space [47].

**Implementation and Evaluation**

We implemented a prototype Memory Management Unit (MMU) which provides protected access to memory buffers used by DSP modules executing in a SDR terminal. MMU allocates a physical large memory pool which it manages. The DSP modules composing the current radio configuration operate in virtual memory space allocated to them by the MMU. Each software module has access only to its own virtual memory space. Additionally, MMU also allocates and manages access to the shared buffers used in the flow graph pipeline to temporary store the signal stream being processed. A shared buffer can be read/written only by those DSP modules that are registered as its reader/writer.

Our solution utilizes the encapsulation capability of object-oriented programming and a LINUX system function for privileged memory access. We defined a new buffer class that allocates and manages buffers in more secure manner than the current GNU software radio platform. A buffer allocated for signal processing can be read or written only by authorized signal processing modules that are registered as readers or writers of the buffer. When a read/write operation is attempted, the buffer verifies the identity of the requester. For write operation, the buffer also checks whether there is enough space to write the requested number of elements into the buffer, thus preventing buffer overflow.

We use the LINUX system function `mprotect` to control accesses to a region of memory containing shared buffers and to prevent malicious or faulty code from obtaining a random pointer within the process address space and overwriting it. `mprotect`
assigns desired access permissions for the memory pages containing the given memory region. If a disallowed access is attempted, the program receives a segmentation violation. A shared buffer stays read-only most of the time. It is unlocked for writing only during a write operation by a module that’s authorized to write to that particular buffer.

We have performed a performance evaluation of our memory protection mechanism. The evaluation setup is as follows:

- Since the main overhead due to new security features lies in memory accesses, we want to observe the effects of increasing number of shared signal processing buffers and size of input data on the running time of the system. We tested the radio system using a self-loop through TCIP/IP socket, without the RF front-end hardware USRP, because delays due to radio transmission and digital-to-analog conversion are orthogonal to our study.

- We used input files containing voice data of varying size. These data were recorded using GNU Radio.

- We used signal processing blocks that applied simple mathematical functions on input data such as multiplication by a vector. As the number of signal processing blocks increased, the number of shared buffers increased proportionally.

Figure 8.2 illustrates running time of the radio system in two different cases: a) unmodified base case; b) security mechanism in place. For each of these cases we run the radio system with three different flow graph constructions: 1 buffer, 3 buffers, 5 buffers. There are six test suites. We execute each test suit with 10 different input files of varying sizes, ranging from 100 Kb to 128 Mb. The dotted lines indicate the performance of the base case; the solid lines indicate the performance of the radio
system with shared buffer protection. The results of our evaluation show that our memory protection mechanism does not add significant overhead (less than 5%) to the running time of the radio system; the overhead is negligible.

8.4 Implementation Platform

Several software environments have been developed for implementing PHY layers for SDR platforms. Besides a number of commercial platforms, two well known open source platforms are GNU Radio and the Virginia Tech OSSIE platform.

We used GNU Radio as our software platform for implementing and evaluating the Secure Configuration Framework.

GNU Radio platform is reasonably hardware-independent. For a computation hard-
ware, a 1 or 2 GHz commodity machine with at least 256 MB of RAM should suffice. We also need some way to connect the analog world to the computer. Low-cost options such as built-in sound cards would limit us to processing relatively narrow band signals. Off-the-shelf, high-speed PCI analog-to-digital boards are available in the 20M sample/sec range, but they are quite expensive. Finding none of these alternatives completely satisfactory, we chose the Universal Software Radio Peripheral for our hardware experimentation. With USRP board, we were able to operate a complete software defined radio—analog antenna on one end and GNU Radio software on the other end, bridged by USRP.

Next, we describe the software and hardware platforms used in our implementation in detail.

8.4.1 GNU Radio Software

GNU Radio is an open source toolkit for building software radios [48]. It is designed to run on desktop computers and, combined with minimal hardware, allows the construction of simple software radios. The project was started in early 2000 by Eric Blossom and has evolved into a mature software infrastructure that is used by a large community of developers.

The GNU Radio signal processing library provides signal processing blocks for modulation, demodulation, filtering, and I/O operations such as file access. In addition, it also provides blocks for communicating with the USRP. New blocks can be added as needed. A radio is built by connecting these blocks to form a flowgraph. This flowgraph is a directed acyclic graph in which the vertices are the GNU Radio blocks and the edges correspond to data streams. Figure 8.3 shows how a FIR Filter, Quadra-
ture Demodulator and Audio Sink are connected in a flowgraph to form a simple FM receiver. Programming in the GNU Radio platform uses a combination of C++ and Python: the processing blocks are implemented in C++ while the flowgraph and the applications that sit on top are developed in Python. We now briefly elaborate on key properties of both processing blocks and flowgraphs.

**Figure 8.3:** GNU Radio flowgraph for a simple FM Receiver.

**Processing blocks** - Generally blocks operate on continuous streams of data. Most blocks have a set of input and output streams: they consume data from their input streams to generate data for their output streams. Special blocks, called sources and sinks, only produce or consume data, respectively. Examples of sources are blocks that read from USRP RX ports, sockets and file descriptors. Similarly, sinks include blocks that write to USRP TX ports, sockets and file descriptors. Each block has an input and output signature (IO signatures) that defines the minimum and maximum number of input and output streams it can have, as well as the size of the data type on the input and output streams.

Each block defines a work function that operates on its input to produce output streams. In order to help the scheduler decide when to call the work function, blocks also provide forecast functions that tell the runtime system the number of input items it requires to produce a number of output items and how many output items it can produce given a number of input items. At runtime, blocks tell the system how many input (output) items they consumed (produced). Blocks may consume data on each input stream at a different rate, but all output streams must produce data at the same rate.
Data buffers  The input and output streams of a block have buffers associated with them. Each input stream has a read buffer, from which the block reads data for processing. Similarly, after processing, blocks write data to the appropriate write buffers of its output streams. The data buffers are used to implement the edges in the flowgraph: the input buffers for a block are the output buffers of the upstream block in the flowgraph. GNU Radio buffers are single writer, multiple reader FIFOs.

Flowgraph mechanisms  Users build a radio by defining a flowgraph using the connect function. The connect function specifies how the output stream(s) of a processing block connects to the input stream of one or more downstream blocks. The flowgraph mechanism then automatically builds the flowgraph; the details of this process are hidden from the user. An key function during flow graph construction is the allocation of data buffers to connect neighboring blocks. The buffer allocation algorithm considers the input and output block sizes used by blocks and the relative rate at which blocks consume and produce items on their input and output streams. Once buffers have been allocated, they are connected with the input and output streams of the appropriate blocks.

Scheduler - The GNU Radio scheduler executes the graph that was built by the flowgraph mechanism. It is implemented as a single thread that loops over all the blocks in the graph, executing each block sequentially until all the data has been consumed. During the execution, the scheduler queries each block for its input requirements and it uses the above mentioned forecast functions to determine how much data the block can consume from its available input. If sufficient data is available in the input buffers, the schedule calls the blocks work function. If a block does not have sufficient input, the scheduler simply moves on to the next block in the graph. Skipped blocks will be executed later, when more input data is available. The scheduler is designed to operate on continuous data streams.
8.4.2 USRP Hardware

The Universal Software Radio Peripheral (USRP) board is a low-cost, high speed hardware component which is very suitable for implementing some real time software radio applications. It is developed by a team led by Matt Ettus for the GNU Radio users [49]. This is an integrated board which incorporates AD/DA converters, some forms of RF front end, and an FPGA which helps to do some high-speed general purpose operations such as digital up and down conversion, decimation and interpolation. Since this board is mainly for developing software radios, the waveform-specific processing, like modulation and demodulation are usually done in the host processing unit. Typically, a USRP board consists of one mother board and up to four daughter boards and it requires a PC or MAC with USB2 interface. Figure 8.4 shows the picture of a USRP board equipped with two Receive (RX) and two Transmit (TX) daughter boards.

The block diagram of a USRP board is shown in Figure 8.5. It has 4 high-speed analog to digital converters (ADCs), each at 12 bits per sample, 64 million samples per second. There are also 4 high-speed digital to analog converters (DACs), each at 14 bits per sample, 128 million samples per second. These 4 input and 4 output channels are connected to an Altera Cyclone EP1C12 FPGA. Moreover, this FPGA connects to a USB2 interface chip, the Cypress FX2, and on to the computer. The USRP connects to the computer via a high speed USB2 interface, and it will also work with USB1.1 after installing some software patches.

AD/DA Converters

Two mixed-signal front end processors (AD9862) from Analog Devices has been used in the USRP board to perform all the analog to digital and digital to analog con-
versions. There are 4 high-speed 12-bit AD converters. The sampling rate is 64M samples per second. In principle, it could digitize a band as wide as 32MHz. The AD converters can bandpass-sample signals of up to about 150MHz, though. If we sample a signal with the IF larger than 32MHz, we introduce aliasing and actually the band of the signal of interest is mapped to some places between -32MHz and 32MHz. Sometimes this can be useful, for example, we could listen to the FM stations without any RF front end. The higher the frequency of the sampled signal, the more the SNR will be degraded by jitter. 100MHz is the recommended upper limit.

The full range on the ADCs is 2V peak to peak, and the input is 200 ohms differential. This is 40mW, or 16dBm. There is a programmable gain amplifier (PGA) before the
ADCs to amplify the input signal to utilize the entire input range of the ADCs, in case the signal is weak. The PGA is up to 20dB. Note that we can use other sampling rates if desired. The available rates are all submultiples of 128MHz, such as 64 MS/s, 42.66 MS/s, 32 MS/s, 25.6 MS/s and 21.33 MS/s.

The Daughter Boards

On the mother board there are four slots for inserting up to 2 RX daughter boards and 2 TX daughter boards. The daughter boards are used to hold the RF receiver interface or tuner and the RF transmitter. There are slots for 2 TX daughter boards, labeled TXA and TXB, and 2 corresponding RX daughter boards, RXA and RXB. Each daughter board slot has access to 2 of the 4 high-speed AD / DA converters (DAC outputs for TX, ADC inputs for RX). This allows each daughter board which uses real (not I Q) sampling to have 2 independent RF sections, and 2 antennas (4
total for the system). If complex I Q sampling is used, each board can support a single RF section, for a total of 2 for the whole system. There are two SMA connectors on each daughter board. We normally use them to connect the input or output signals.

**Field Programmable Gate Array (FPGA)**

FPGA plays an important role in software radio’s operation. All the ADCs and DACs are connected to the FPGA. Some of the high bandwidth math has been done into the FPGA to reduce the data rates so that the data can be transported through the USB 2.0 bus. The FPGA connects to a USB2 interface chip, the Cypress FX2. FPGA circuitry and USB Microcontroller is programmable over the USB 2.0 bus. Our standard FPGA configuration includes digital down converters (DDC) implemented with cascaded integrator-comb (CIC) filters. The FPGA implements 4 digital down converters (DDC). This allows 1, 2 or 4 separate RX channels. At the RX path, we have 4 ADCs, and 4 DDCs. Each DDC has two inputs real and complex. Each of the 4 ADCs can be routed to either of real or the complex input of any of the 4 DDCs. This allows for having multiple channels selected out of the same ADC sample stream.

The digital up converters (DUCs) on the transmit side are actually contained in the AD9862 CODEC chips, not in the FPGA. The only transmit signal processing blocks in the FPGA are the interpolators. The interpolator outputs can be routed to any of the 4 CODEC inputs. The multiple RX channels (1,2, or 4) must all be the same data rate (i.e. same decimation ratio). The same applies to the 1,2, or TX channels, which each must be at the same data rate (which may be different from the RX rate). Figure 8.6 shows the block diagram of the USRP’s receive path. The MUX is like a router or a circuit switcher. It determines which ADC (or constant zero) is connected to each DDC input. There are 4 DDCs. Each has two inputs.
Digital Down Converter (DDC)

DDC down converts the signal from the IF band to the base band. Second, it decimates the signal so that the data rate can be adapted by the USB 2.0 and is reasonable for the computers’ computing capability. The complex input signal (IF) is multiplied by the constant frequency (usually also IF) exponential signal. The resulting signal is also complex and centered at 0. Then we decimate the signal with a factor N. The decimator can be treated as a low pass filter followed by a down sampler. The decimation rate must be in 1 to 256. Finally, the I and Q complex signal enters the computer via the USB for further processing. TX path works reversely. We need to send a baseband I and Q complex signal to the USRP board. The digital up converter (DUC) will interpolate the signal, up convert it to the IF band and finally send it through the DAC.
Chapter 9

Related Work

In this chapter, I present a summary of existing research and show how my thesis complements related topics. In Section 9.1, I summarize the ongoing major initiatives that investigate SDR related issues. In Section 9.2, I survey previously published works in SDR security and discuss how they relate to this thesis. In Section 9.3, I summarize research works on SDR configuration and compare them to our work. In Section 9.4, I review research works on component composition as it applies to systems software. Finally, in Section 9.5, I overview formal security models and discuss whether they are applicable to our work.

9.1 SDR Initiatives

There are several ongoing efforts that investigate SDR related issues. The notable key projects include: the XG program undertaken by DARPA, the JTRS program initiated by DoD, the E2R project of the European FP6 program, the NCASSR SDR project led by NCSA, and the OSSIE project at Virginia Tech.

The DARPA neXt Generation (XG) communications program [50] develops a new generation of spectrum access technology. In order to address the “apparent” scarcity of spectrum and deployment difficulty of services, XG is pursuing an approach wherein static allotment of spectrum is complemented by the opportunistic use of unused
spectrum on an instant-by-instant basis. They stress that the true potential of this new approach can be exploited only if in addition to spectrum agility, they provide policy agility where policies are not embedded in the radio, but can be loaded "on-the-fly". XG specifies a policy language framework to express and enforce policies [51]. They chose the WWW Consortium’s Web Ontology Language (OWL) as a base language for this framework. OWL is a machine-understandable, semantic markup language, and it’s an extension of XML and RDF. XG focuses on the generation and translation of spectrum allocation policies, whereas our proposal pertains to radio configuration policy.

The Joint Tactical Radio System (JTRS) Program [52] is the U.S DoD initiative. JTRS developed a Software Communications Architecture (SCA) [53] that provides an open architecture framework enabling programmable radios to load waveforms, run applications and be networked into an integrated system. The security requirements for this architecture are specified at all layers of radio system: hardware, operating system and applications. They provide a definition of security architecture and associated requirements for future U.S. DoD radios. SCA has been adopted by SDR Forum [54], the international standardization body of software radio technology. SDRF is facilitating broad international acceptance of the SCA across all SDR domains around the globe, including civil, commercial and military sectors.

End-to-end reconfigurability (E2R) has been identified as a major research topic in the European 6th Framework Program (FP6) [55]. The project scope covers the complete end-to-end system (stretching from the user device all the way up to internet protocol and services) as well as reconfigurability support (functionalities such as management and control, download support, spectrum, regulatory issues and business models) along the whole communication path. One of the major working areas in E2R is mechanisms to increase spectrum and resource efficiency through application
of reconfigurability.

The NCASSR SDR Project at NCSA [56] investigates various SDR related issues such as extending the GNU Software Radio receiver design into a 900MHz narrowband software-defined radio transceiver. They are also developing a real-time visualization software that provides a qualitative, high-level block diagram of the operation of the SDR. Related to SDR security area, the NCASSR team is developing a voice authentication application that identifies and authorizes SDR users.

OSSIE [57] is an open source Software Defined Radio (SDR) development effort based at Virginia Tech. OSSIE is primarily intended to enable research and education in SDR and wireless communications. The OSSIE software package includes an SDR core framework based on the JTRS Software Communications Architecture (SCA), tools for rapid development of SDR components and waveforms (applications), and an evolving library of pre-built components and waveforms. OSSIE is an alternative SDR platform to GNU Radio. Our implementation platform of choice was GNU Radio.

9.2 Research in SDR Security

In this section, we discuss previous research related to security of software defined radio. We have categorized this work into three main groups. The first one proposes a secure SDR device architecture. The second group presents security framework for download to commercial wireless devices. The third group proposes schemes for securing the download channel.

Lam, Sakaguchi et al. [58, 59] propose a secure SDR device architecture with the
following key features: 1) separate hardware and software certification; 2) Radio Security Module whose functionality includes installation, storage, operation, storage, and termination of software in the terminal; 3) Global Positioning System used to determine in which geographical location the wireless device is operating for global roaming support; 4) Automatic Calibration Unit to ensure that the output spectrum is compliant with regional radio regulations. It is not clear whether this solution is cost-effective for commercial wireless devices.

Michael et al. [60] propose a framework for establishing secure download for SDR that includes employment of tamper resistant hardware and four different cryptographic techniques: secret key encryption, public key encryption, cryptographic hashing, and digital signature. This framework provides solutions for verification of the declared identity of the downloaded software, verification of the integrity of the downloaded data, disabling the ability to run unauthorized software on the SDR device, and the secrecy of the downloaded data. They assume the existence of tamper-resistant hardware that provides secure storage for the terminal keys. This work also assumes that software is created and distributed by the hardware maker. Sugita et al. [61] propose an electronic commerce scheme that utilizes the ability of SDR to switch between different security algorithms. The following issues are identified, but without precise specifications: a) encryption of download channel, hardware key, and terminal ID to prevent illegal copying of the downloaded program; b) certification against alteration of the downloaded program. While this work addresses the secure software download problem, our work focuses on the problem of secure configuration and execution of downloaded software.

Brawerman et al. [3] propose a lightweight LSSL protocol that uses less bandwidth than SSL to securely connect a manufacturer’s server and SDR devices for downloading radio configuration files. In addition to securing the download connection, their
secure protocol includes mutual authentication, public/private key mechanisms for
data encryption and decryption, and fingerprint calculations to check data integrity.
Uchikawa et al. [36] propose a secure download system that uses the characteristics
of the field programmable gate arrays (FPGAs). The wiring of configuration logic
blocks on FPGAs can be arranged in many different ways (astronomical number),
enabling high security encipherment to prevent illegal acquisition of software using
replay attack. These works assume that SDR devices download software only from
their manufacturers, and they focus on confidentiality and authentication aspects
of SDR security. We assume that software can be provided by a third-party, and
certified software could be buggy.

Some of the SDR Forum documents discuss security issues [5, 28, 29]. Among them,
Falk et al. [5] suggest the use of sandboxing as a possible approach to prevent
harm from potentially malicious software. We elaborated on this idea further and
implemented a prototype of secure memory management unit.

9.3 Research in SDR Configuration

Several previous works researched SDR configuration issues [2, 62, 63, 64]. However,
none of these works specifically address the problem of mapping high-level configu-
ration policies into executable dataflow graphs or propose techniques for validating
candidate configurations. Generally, these works outline a reconfiguration archite-
cture without specific methodologies or security functions.

Moessner et al. [2] propose a distributed Reconfiguration Management Architecture
(RMA) offering the means to control and manage reconfiguration of Software Defined
Radios and introduces a mechanism capable to verify the radio standard adherence
of intended terminal configurations. A particular focus of this work was on the description of software for the reconfigurable radio part within a SDR terminal, as tag-files. Specifications for security functions are not included in this work.

Srikanteswara et al. [62] present a unified design architecture of soft radios on a reconfigurable platform called the Layered Radio Architecture. This architecture makes it possible to incorporate all the features of a SDR while minimizing complexity issues. Burgess et al. [64] present a framework developed in the SCOUT project that allows the software modeling of SDR configuration. The framework is aimed at a heterogeneous hardware platform and takes into account real-time constraints as well as being power conscious.

Kountouris et al. [63] cover issues related to the device-level support for SDR reconfiguration. Device level support implies appropriate hardware and software architectures as well as design approaches accounting for the supported reconfiguration scenarios and for the device specific constraints. A customized, lightweight component-based approach has been proposed that can be tailored to handle various types of reconfiguration scenarios necessitating network involvement.

9.4 Component Composition

Component composition has been studied in different research areas, such as service composition [65, 66] and systems software composition [67, 68, 34]. Although we adapted some principles and techniques developed for service composition, our work fits in the later category.

X. Gu, K. Nahrstedt et al. [65, 66] introduce an integrated peer-to-peer service
composition framework that provides high quality and failure resilient service composition. From this work we adapted the composition model based on graph mapping. In our work, we map a configuration profile of a radio device into an executable graph of software components.

Much of the early research in component-based systems software involves the design and implementation of microkernels such as Mach [69] and Spring [70]. With an emphasis on flexible and extensible architectures microkernel research is essentially complementary to research on reconfigurable, component-based radio platforms.

The most popular topic in this area is the use of component kits for building systems software. Reid et al. [67] developed Knit, a component definition and linking language designed for component kits. It provides support for component configuration and detects errors in component composition. The Scout operating system [71] consist of a modest number of modules that can be combined to create software for network appliances and protocols. The Click system [72] also focuses on networking, but specifically targets packet routers. These systems do not support non-functional requirements for composition and do not provide automated configuration, in contrast to our work.

Ensuring the correctness of software composition at construction time has been addressed in the literature in a number of different ways. In [68] the authors introduce the notion of micro-components. Micro-components represent programming language idioms. they have assigned contracts and requirements. When being composed those contracts and requirements are statically checked using first order predicate logic. However, non-functional requirements and composition rules are not considered.

Walnau [35] aims to achieve predictability by construction. By using predictable
assembly it can be known before the actual components are integrated that they will play together with respect to one or more run-time properties of interest. We use the principles of this work to reason about SDR’s component-based certification. We discuss this in detail in Section 6.1.2.

9.5 Security Models

Models are important in determining the policies a secure system should enforce [73, 74], and understanding whether the properties of protection systems can be achieved [75, 76, 77]. In particular, security models are used to test a particular policy for completeness and consistency, document a policy, design an implementation, and verify whether an implementation meets its requirements [78].

The Bell-La Padua model [73] describes the allowable paths of information flow in a secure system. The model is a formal state transition model of computer security policy that describes a set of access control rules by the use of security labels on objects, from the most sensitive to the least sensitive, and clearances for subjects: top secret, secret, confidential, unclassified. The notion of a secure state is defined, and it is proven that each state transition preserves security by moving from secure state to secure state, thereby inductively proving that the system is secure. While the Bell-La Padua model focused only on confidentiality of information, the Biba model [74] describes rules for the protection of information integrity and is characterized by the phrase: "no write up, no read down". In this model, users can only create content at or below their own security level, and they can only view content at or above their own security level.

The Graham-Denning model [75] is a formal model having generic protection prop-
erties. The model has eight basic protection rules that outline how to securely create/delete an object, create/delete a subject, provide the read/grant access right, and securely provide the delete/transfer access right. The Harrison-Ruzzo-Ullman model [76] states certain characteristics that can or cannot be decided by an arbitrary protection system. This model is based on commands, where each command involves conditions and primitive operations. The Take-Grant model [77] is a formal protection model used to establish or disprove the safety of a given computer system that follows specific rules. It shows that for specific systems the question of safety is decidable in linear time, which is in general undecidable.
Chapter 10

Conclusions

This chapter highlights the lessons learned from my study of security and configuration issues of software defined radio, and summarizes solutions I explored to address relevant problems.

10.1 Conclusions

The principal defining characteristic of SDR is that the most important operational parameters can be configured when it is in use rather than when it is manufactured. This enables highly flexible handheld devices that can switch from one network technology to another to suit a particular application or environment. However, reconfigurability makes attacks against the SDR terminal and network easier and perhaps more prevalent. As SDR technology gains widespread usage, it is inevitable that third party component support, and flexible configuration schemes will emerge.

In this thesis we presented a secure configuration framework that addresses wide variety of issues related to radio reconfigurability. We conducted extensive survey of SDR security issues, and formulated a list of security requirements for our work. Detecting unauthorized modifications of SDR configuration software, preventing a hacker from changing the device configuration, ensuring that radio emissions are limited to those of regulations, allowing installation of only certified software, facilitating radio
to ascertain its configuration information to service providers—these are some of the security requirements.

Keeping these requirements in mind, we developed an automated configuration methodology that decouples waveform design from its implementation. Given a expert database of blueprints on how to compose a particular type of radio, our system automatically builds a deployable composition of digital signal processing modules. Ensuring validity and safety of such composable radios turns out to be a very hard problem. We narrowed the scope of configurations profiles so that we can make better predictions about the properties of the composite part. We require that software components and configuration profiles come pre-certified. We modeled the notion of composable certification, and were able to make some assertions about the conformity of the composite radio device.

By requiring more specific configuration profiles, we were able to achieve more efficient mapping of a configuration profile into a deployable DSP flowgraph. Methodology for verifying software certification and checking compatibility of a configuration for the host radio are discussed. We presented performance evaluation of the composition algorithm, and described our experiment of deploying it on GNU Radio software platform and USRP hardware platform.

We addressed the issue of protecting integrity and confidentiality of downloaded software. Instead of using traditional secure communication techniques, we designed a protocol that utilizes Trusted Platform Module’s (TPM) functionalities. In order to ensure conformity of an SDR terminal and provide an effective way to for the terminal to ascertain its current state to a remote party, we make an assumption that all SDR terminals come equipped with a TPM. We informally analyzed the secure download protocol for security properties.
We discussed how trusted computing can be adapted for SDR to provide a trusted radio platform. We outlined a secure boot sequence that uses integrity primitives specified for Mobile Trusted Module. This boot sequence has a property where the verifier of the platform integrity need not check all attestation certificates, but just the initial one.

### 10.2 Summary of Contributions

The main contribution of this thesis is a framework for secure configuration of software defined radios. The specific contributions are:

1. Technique for automatically configuration a radio terminal given a configuration profile.
2. Graph mapping model for composing DSP components.
3. Specified portable format for configuration profile.
5. Protocol for secure download of radio software.
6. Methodology for conformance validation and compatibility check.
7. Application of trusted computing concepts to build a trusted SDR platform.
8. Secure boot sequence for trusted SDR platform.
9. Methodology for remote attestation.
To summarize, I claim my thesis provides a secure configuration framework to automatically configure waveform components of a software radio and ensures validity and integrity of the radio configuration.

10.3 Future Research

In this section we discuss possible directions for future work to address SDR configuration issues.

One interesting problem to explore in the future is a dynamic configuration scheme where device configuration dynamically adapts to changing network conditions. The configuration should correspond to the capabilities, preferences, and dynamic properties (load, radio link properties) of the currently used network. For example, in a noisy communication environment, the configuration may choose to use baseband software modules that perform better signal filtering at the expense of more power consumption. Or when multiple radio access technologies are available in the region, the terminal may switch to the one with less power demand. In this scenario, the terminal will continue using the same service, as the service providers need not be associated with one single network operator. Note, our work addresses SDR configuration initiated by some actor, i.e. on-demand scheme, not dynamic.

In our we compose DSP software components based on a configuration profile. The flip side of this, generating a configuration profile from an existing radio configuration has some uses. The benefit of developing such a tool is making configurations of legacy devices portable. This tool would parse the source code of a deployable configuration (likely to be in high-level languages such as C++, Python) and generates an XML template of the configuration profile.
Another interesting future work would be to explore various ways to reason about composable properties. Modeling and asserting a composite property of composable components is very challenging. Given properties of individual components, and the blueprint of interconnecting them, how to derive characteristics of the composite part? Composability of security properties is especially of interest to the authors.
References


[33] Federal Communications Commission, “terminal equipment approval requirement.”


Author’s Biography

Suvda Myagmar was born in the northwestern region of Mongolia, and grew up in Ulaanbaatar, the capital of Mongolia. She graduated from the Concord University, West Virginia, USA in 2001 with a Bachelor of Science degree in Computer Science. She was ranked the top of the graduating class and delivered a valedictory address at the graduation ceremony. During her undergraduate studies, she worked as a software engineer at VTLS Inc for three semesters.

Suvda obtained her Master of Science degree in Computer Science from the University of Illinois at Urbana-Champaign in August 2005 and her Doctor of Philosophy degree in Computer Science in February 2008 from the same university. During her graduate studies, Suvda interned at Motorola, IBM Almaden Research Center, and Hewlett-Packard Labs performing research on wireless networks, social computing, and systems security respectively.

Her research interests include security and configuration of wireless devices, and the use of data mining techniques in bug detection and social computing.