A METHODOLOGY FOR A VERIFIABLE SOFTWARE PLATFORM TO SECURE SOFTWARE DEFINED AND COGNITIVE RADIOS

Thomas W. Rondeau (Virginia Tech, Blacksburg, VA, USA; trondeau@vt.edu); Timothy M. Bielawa (Computer Sciences Corporation; tbielawa@vt.edu); David Maldonado, (Virginia Tech, Blacksburg, VA, USA; davidm@vt.edu), Michael Hsiao (hsiao@vt.edu), and Charles W. Bostian (bostian@vt.edu)

ABSTRACT

Software defined radios (SDR) introduce many new challenges, one of which is the proper development, maintenance, and distribution of the core software. As with any software venture, SDR requires industry, government, and the independent development community to work together to produce an environment that fosters software development and innovation. SDR differs from other areas of software development by the long history of radio regulatory requirements that must be satisfied. In this paper, we propose a methodology to bring to the SDR world the same level of development and innovation that has made other software ventures a success. The verification platform we propose allows software developments to guarantee regulatory compliance even when faced with the challenges of open source software and cognitive radio regulation.

The system itself first verifies that the software meets regulations and sends back to the developer the object code along with a security key that grants access to the radio for download. The system's security policy relies on standard industry encryption and authentication schemes. Therefore, it requires no new developments but rather the application of existing methodology for this application.

1. INTRODUCTION

Over the past few years, much work has gone into the concept of software defined radio (SDR) downloads – that is, getting code that defines a new or modified air interface into the radio platform [1-3]. This work has been focused on the reliable, efficient, and secure downloading of new software from the company of origin, primarily though an over-the-air (OTA) interface. Major concerns are guaranteeing the integrity of the download and the protection of the software itself from prying eyes.

There is another software issue that has been largely overlooked, mostly because of the immaturity of the SDR field: open source software (OSS) and radio software developed by individuals. Here, the concern is not so much the security of their source code (although it could be); instead, the concern is with the probable reaction of the regulators. In its recent Report and Order on cognitive radio, the Federal Communications Commission (FCC) [4] discusses open approaches like the GNU Radio and exempts them from the rules governing interference regulations because of their low output power and amateur-band only operation. Extrapolating from here, it is easy to envision a future where the available open source software easily allows operations of the radios outside of the amateur bands, causing great potential for interference to existing services.

Accompanying SDR is the development of cognitive radios (CR), which allow for new and possibly unknown waveform development in real-time operation [5, 6]. While the FCC is interested in supporting this, as stated in their Report and Order, it does not yet understand how to control the radios to ensure compatibility with the regulations.

Chapin [7] presents a strong argument for how successful efforts in new software technology require wide-spread support and development tools for adoption and design of best practices. There is no better way to advance SDR and CR technology than to foster the creative spirit of the academic and development communities through encouraged experimentation and supported development systems like open source software.

Here, we introduce a new method of SDR and CR software development along with a methodology and framework for assuring compliance and securely downloading the software to these radios. The paper first presents an analysis of similar software download systems and concepts and then gives an overview of automatic software verification methods that could be employed. Section 4 provides the system architecture, and Section 5 provides a cryptographic analysis of the security issues. In Section 6, we propose a system of development classes for SDRs and CRs to support development and regulations on different levels. Section 7 concludes with thoughts on the development and use of this system.

2. CURRENT RELEVANT TECHNOLOGIES

2.1 Over-the-Air downloading

Although many methods exist for software upgrades, over-the-air downloads represents the most attractive and challenging one. In these situations, downloads and upgrades are performed under the care of an individual user,

A METHODOLOGY FOR A VERIFIABLE SOFTWARE PLATFORM TO SECURE SOFTWARE DEFINED AND COGNITIVE RADIOS

Thomas W. Rondeau, Timothy M. Bielawa
David Maldonado, Michael Hsiao, and Charles W. Bostian

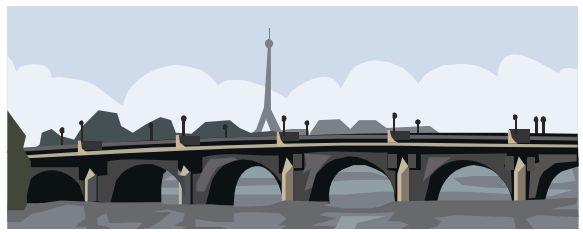
Center for Wireless Telecommunications
466 Whittemore Hall
Blacksburg, VA 24061
Virginia Tech

11/14/2005

Two problems facing the future of SDR and CR

1) Bridging the gap:

Hardware Engineering



Software Engineering

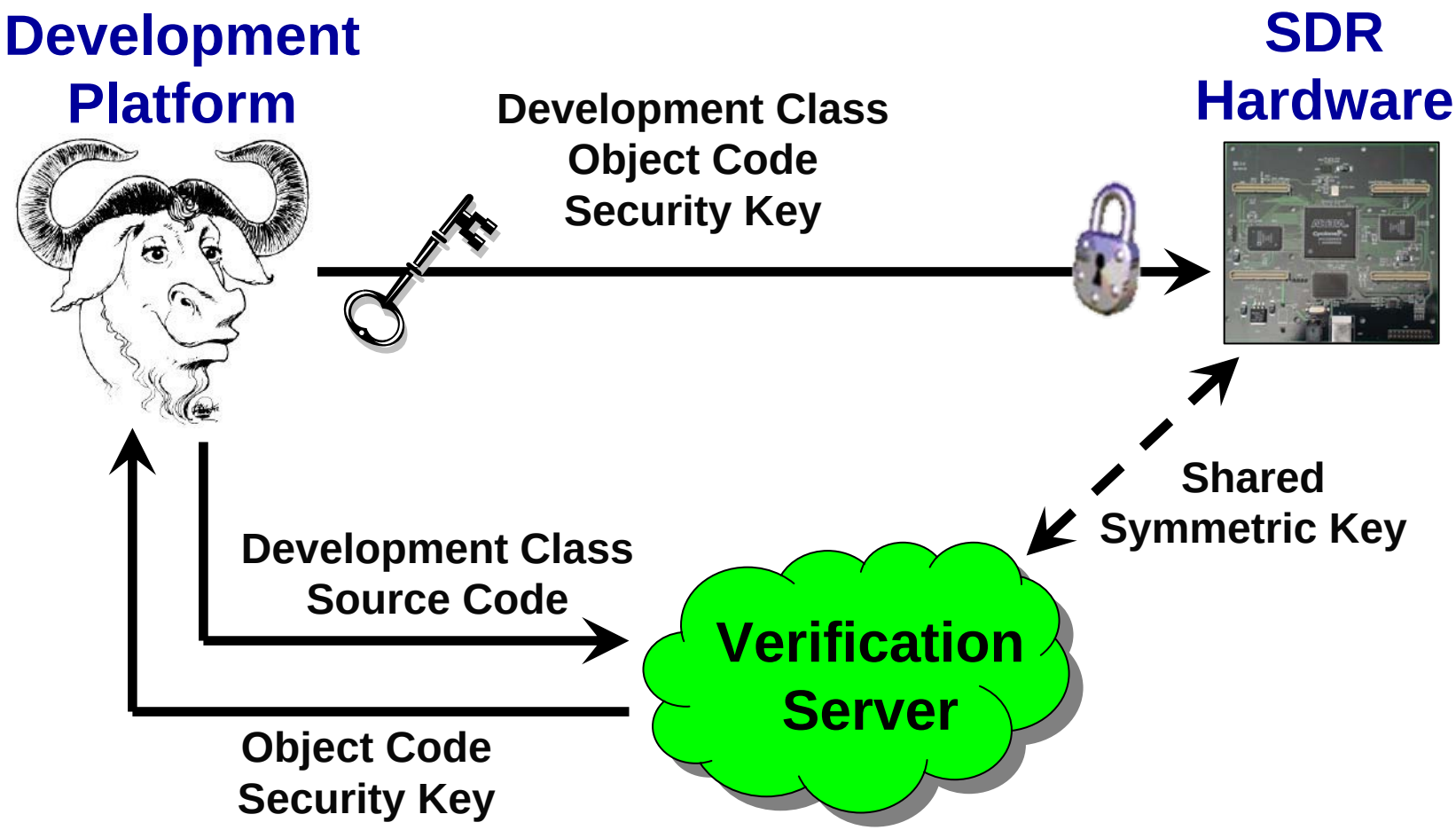
2) Support and encourage creativity, innovation, and development in government, industry, and academia.

Ada

vs.

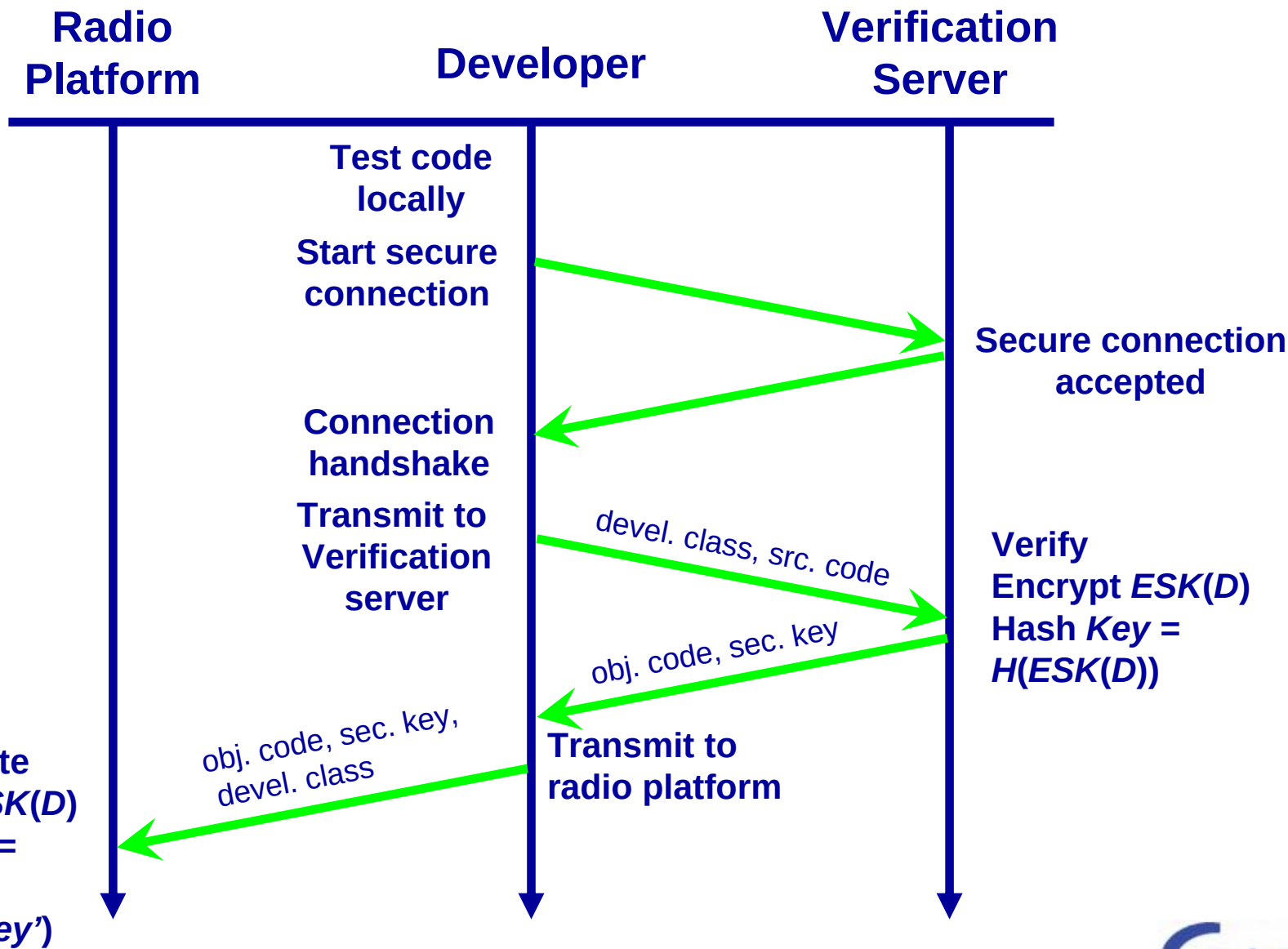
python

Overview of System Operation



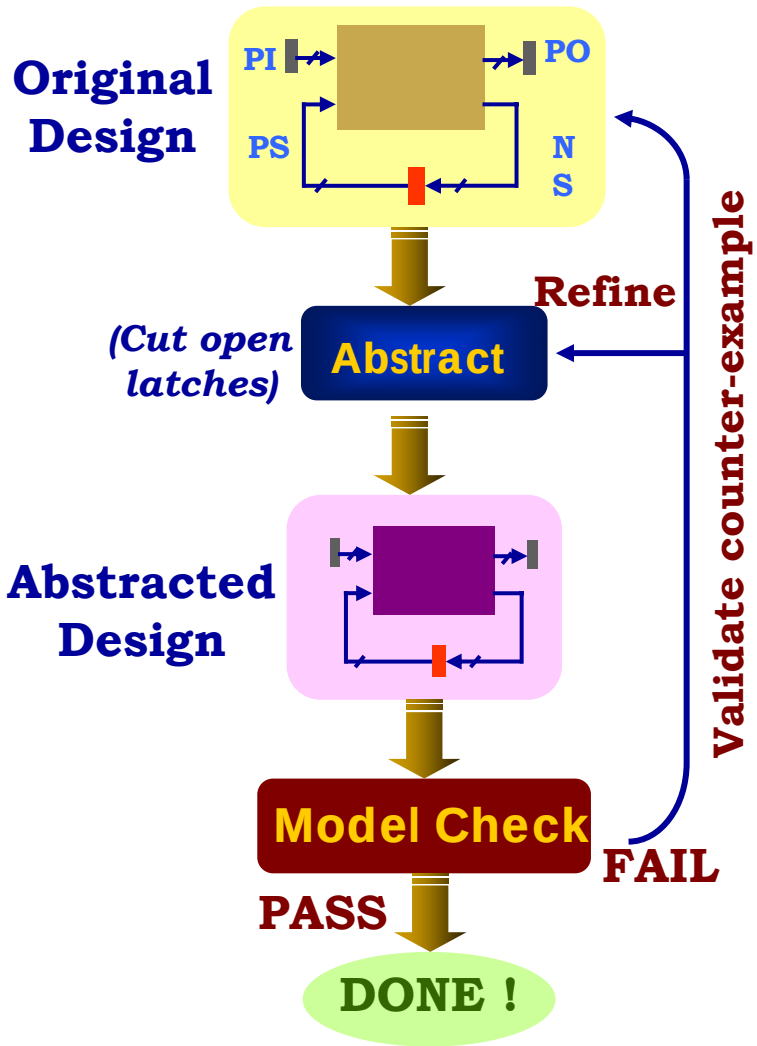
Software is verified, and a secure key is used to protect downloads to the radio.

Timing Diagram



Software Verification

- Apply unbounded and/or bounded model checking to underlying software
- Check property violations
  - Frequency bounds
  - Power output
  - Authorization
- If property holds in abstracted design, it definitely holds in original



Security Key

Plaintext

Plaintext contents of the Object File.

Encrypt

Ciphertext

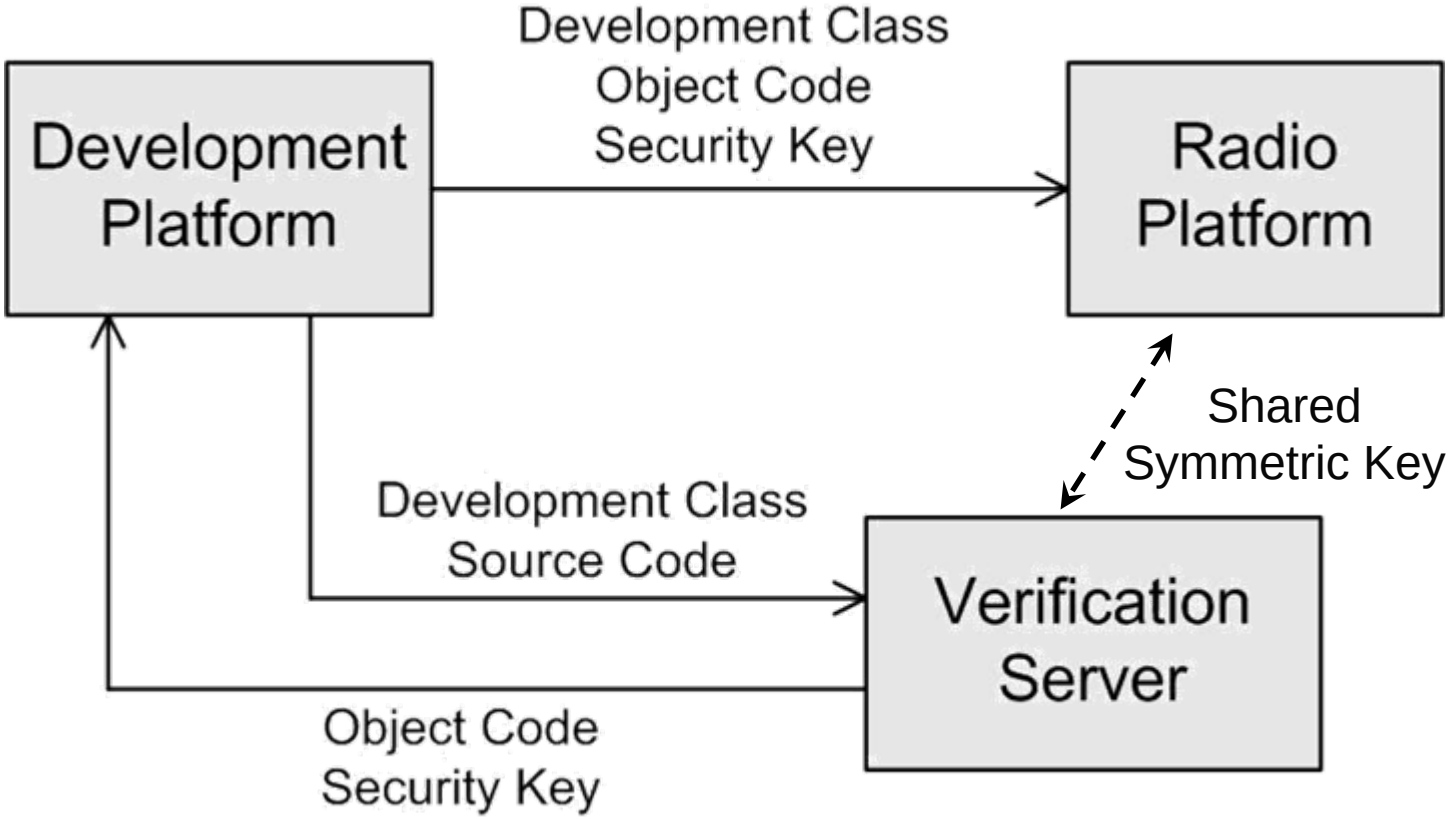
Œ
□ □ □ □ x□ □ ‡NP□™`ÉF&
4ý^h□ }Ý—º□ °æ‹—
'□ □ ,È¤;Ì½ŽÑ"□ WùrÏ"ñ†
£˜□ Û®□ RàN□ □ Õø!Ÿab
(j'i□ Ñ—‡f5¬"€Ü¹cÈ□

Hash

7e3377644e2392651d5cd5d93ce49b4a

- **Encryption links the key to the object code.**
- **Hashing protects against decryption attacks**
- **Final key is small and secure**

System Diagram



The radio platform contains its own symmetric key to recreate the HMAC.

If this HMAC matches the security key, the code is authenticated.

Development Classes

- Provide a set of *development classes* that developers can use to certify their radios.
- Development classes are licenses the FCC grants developers
- Symmetric keys are associated with a particular development classes
- Could be free or priced depending on level. Access to verification servers could be priced by these classes.

Partial Class List

- **Amateur**
- **Cognitive Radio**
- **Part 15 / ISM**
- **Cell phone**
- **GPS receiver**
- **Public Safety**
- **Experimental**

Security Issues

- Cryptographic-Based Attacks
  - *man-in-the-middle*
  - *birthday*
  - *plaintext*
- Hardware-Based Attacks
  - *Power*
  - *Timing*
  - *Logic*

Enigma

Alice

Bombe

Eve

Bob

Acknowledgement

National Institute of Justice
The Research, Development, and Evaluation Agency of the U.S. Department of Justice

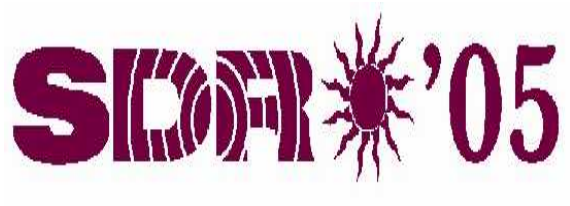
This project is supported by Award No. 2005-IJ-CX-K017 awarded by the National Institute of Justice, Office of Justice Programs, US Department of Justice. The opinions, findings, and conclusions or recommendations expressed in this publication/program/exhibition are those of the author(s) and do not necessarily reflect the views of the Department of Justice.

National Science Foundation
WHERE DISCOVERIES BEGIN

This material is based upon work supported by the National Science Foundation under Grants No. 9983463, DGE-9987586, and CNS-0519959. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Thank you!

Thomas W. Rondeau
Timothy M. Bielawa
David Maldonado
Michael Hsiao
Charles W. Bostian



Contact:

trondeau@vt.edu

Center for Wireless Telecommunications
Virginia Tech
466 Whittemore Hall MC 0111
Blacksburg, VA 24061

not a licensed authority. In general terms, OTA downloads can happen in two ways: originated by the network server (push) or by a unit request (pull). Most current work in the OTA download area assumes the push model, which lowers the burden of authenticating the devices. Within a given network, the authentication process is easier as the network server knows which units will require a given update/download, and the service that it provides as an authentication mechanism is already in place. On the other hand, when a pull request is generated, the server may have no knowledge of the intent of the device and how the download would affect any given service. Therefore, the authentication mechanism needed must contain additional information beyond the one that is currently provided.

Signaling is required to request or initiate a download (by either the server or terminal). The signaling will have to cover the aspects of security (authorization and authentication) as well as negotiation of the download. Several approaches have been presented that explain in detail successful ways to achieve that [1-3].

Security in OTA downloads is one of the largest concerns in implementing this technology, and a lot of work has gone into the development secure methods to safely transport the information between the developer and the radio as well as ensuring proper authentication [1-3]. These systems assume a closed-source development structure, which has different challenges than this system is faced with.

2.2 Other Software Download Systems

Without naming any particular technology, we can separate relevant methods into two main areas: proprietary technologies that secure and protect the interest of the original developer, and technologies that distribute open source software for general application, e.g., software published under the General Public License (GLP).

The first group protects the intellectual property of the original developers and manufacturers, prevents access to too much information, and possibly offers a high-level development kit supporting limited applications. These systems are often built on proprietary processors and operating systems. Downloads are restricted to certain areas of operation within the system, and other areas require special permissions. These closed systems offer some flexibility, but often at a high cost. Physically, these systems are manufactured to be secure; the systems are not publicly well documented and are physically compact and obscure. While such systems would offer security guarantees to regulators, they discourage the innovation so critical in the development of SDR and CR techniques and communities.

Some closed technologies are supported on open platforms that use standard parts, which are easily reverse engineered. While they are secured to some extent, there is a history and culture of “tinkerers” that hack such systems.

On the other side, open source methods grant access to most of the system, allow individuals to change and distribute newer versions of software or build new systems incorporating parts of or whole projects. The Internet has grown out of this concept, and many of the standard tools used today have come directly from this culture, like the GNU project and Linux.

The innovation the market has seen from the open source concept is now being brought, for the first time, into a world with regulatory limits and protections. In neither the closed nor open systems do we yet see the crossover between the freedom and development of the OSS community and the protections required by the government.

3. SOFTWARE VERIFICATION

The verification system we propose is a general architecture for safely conveying verified software from the developer to the radio platform, but we do not provide any particular means of performing the verification. While verification could be done by an individual human auditor, it could also be done, more efficiently, with advanced and automatic software verification techniques. In this section, we briefly summarize the concept of software verification and argue for the validity of this approach [8].

Software testing has traditionally been performed via simulation, program analysis, theorem proving, or model checking. While simulation is easy to understand, it is not scalable to large systems. In addition, coverage metrics such as statement and condition coverage are not readily mapped to how the bugs may have been covered.

In testing the software embedded in SDRs and CRs, success in validation has an additional dimension of making sure that the waveforms, frequencies, etc., produced by the embedded software not only comply with the FCC, but also do not make demands beyond the radio hardware capabilities. This allows the testing of SDRs to target specific goals or properties. Thus, model checking becomes a very suitable technique.

Model checking by definition is the systematic exploration of the state space of the underlying system. Properties can readily be mapped to states and/or transitions in the state diagram. Software model checking essentially views the software as a transition system and applies the concept of model checking onto it. Unlike hardware systems, software can have significantly more variables. However, most variables, especially local variables, do not change in value and may be dormant when considering a given program trace. This can be advantageous when model checking the software. Aggressive abstraction can be performed before model checking is applied.

Given the abstracted model of the software on the cognitive radio, along with the set of properties that we need to check for, the testing follows the standard model checking

flow, and results of the model-check either uncovers any counter-examples to which certain properties are not upheld or declares the properties are upheld in the software.

We note that the software testing paradigms are still evolving, including software model checking. Advances in testing technology will further our goals of testing the embedded software in SDRs. Nevertheless, we believe our proposed flow for verifying the software in SDRs is valid and can be widely deployed for SDR servers.

4. THE SYSTEM ARCHITECTURE

The system discussed in this paper is a platform model to ensure SDR code is properly verified before being downloaded and used on a radio. It requires a system that both properly verifies the software and provides a secure method to get the code from the developer to the radio that prevents the download of unverified code; in this paper, we focus on the later issue. This system requires no new work in cryptography and authentication; we only apply these techniques in a way conducive to both the developers and the regulators.

The system design (see Figure 1) operates simply. Following the timing diagram of Figure 2, the developer begins by first locally testing and verifying the code before offloading it to the verification server. The developer and server perform standard authentication and establish a secure connection to transmit the information. The developer sends the source code as well as an indication of the development class under which the code will be verified.

The verification server takes the code and performs the verification methods on it. Once verified, the server creates a Keyed-hashed Message Authentication Code (HMAC) security key [9, 10]. Both the object code and the security key are transmitted back to the developer.

The developer then sends the object code, security key, and development class to the radio. The radio uses the development class to determine which private key it must use in the creation of its own HMAC. The object code is then encrypted and hashed, and the HMAC is compared with the HMAC of the security key to test if there is a difference. If there is no difference, the code is accepted. If there is a difference, an error is returned to the developer.

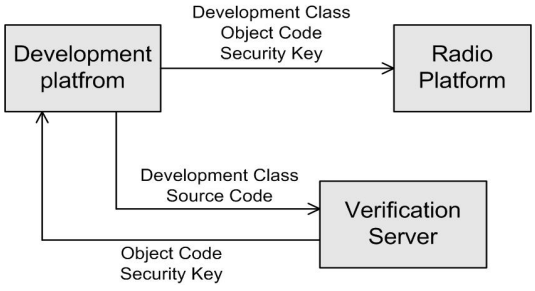


Figure 1. Verification ystem block diagram.

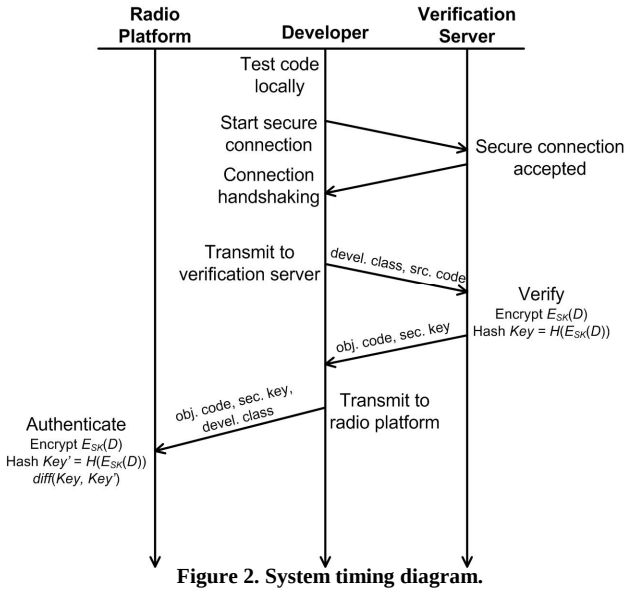


Figure 2. System timing diagram.

We will now look closer at all the ideas developed for the system operation.

Source Code

The source code written by the developer may be open source, proprietary, or in between; the same principles of the system's operation apply regardless. The code should be verified as much as possible by the developer before transmission to the verification server to ease the burden and time for both the server and developer. If desired, the connection between the developer and server can be secured using a SSL/TLS security policy [11].

Verification Server

The verification server assumes an analogous function to the FCC in verifying proper regulatory compliance of the software. Evaluation can be done in a similar way as current compliance testing is done, or it could be done in a more sophisticated and automatic way using the software verification theory discussed previously. The verification servers could be controlled completely by the regulatory body (i.e., the FCC) or on trusted third-party servers to offload some processing time. Third-party servers offer economic incentives to the administrative entities through a per-use charge depending on the type and number of radio applications supported and the level of sophistication used in their verification methods. Independent companies could also obtain authorization to establish their own trusted servers to allow complete in-house verification of all SDR and CR applications created.

Object Code

The object code is returned by the verification server after successful compilation and verification. The terms object code and compilation are generalized for many code

structures, even if no code is compiled (such as with some Java or Python implementations). Compilation on the server is important to have the proper ties between the object code and the security key, as is discussed in the next section. The server must therefore support a number of standard languages to support different development strategies. The list of languages can be constrained to some reasonable number of common and well-accepted languages (e.g., C, C++, Java, and Python to start with).

Security Key

The verification server returns two items: the object code and the security key. The security key is what allows the SDR to permit the user to download new code. Of course, the radio must know that the code being downloaded is code that was properly verified, and so the security key must include enough information to ensure this. This requires that the security key is somehow directly tied to the object code.

It is because of the relationship between the security key and the object code that we recommend the code is always compiled on the verification server and not left to each developer to compile the software. Different compilers and configurations will result in different object code, which will destroy the relationship between the security key and the server's object code.

To create the security key, the verification server must have enough information about the object code to tie the two together; however, the key must be sufficiently unique that the user can not create his own security key and sufficiently complex such that there is no one-to-one relationship between the object code and the security key.

A common way of authenticating software is to create a hash, or message digest, of the software [12, 13], which is a one-way fingerprint of the contents of the hashed software (one-way in that there is no (known) mathematical method of taking the fingerprint and knowing the message that created it [14]). However, this by itself is a weak authentication method if we are trying to prevent only verified code from being downloaded to the radio.

Another method that could be used is to encrypt the object code with a private key and only the SDR would be able to decrypt it using a public key (or using a symmetric algorithm). However, this is very weak form of encryption since the user could compile his own code and, despite possible differences between the two compilers, would have a reasonable facsimile if not the exact copy of the object code created by the server. This allows an easy plaintext attack on the encryption, enabling the extraction of the key originally used to encrypt the object code and allow the user to create his own encrypted object code.

A much more secure method combines both of these functions and is referred to as a Keyed-hashed Message Authentication Code (HMAC) that has been used in network applications for years [9, 10]. The HMAC first encrypts the object code with a symmetric key; the ciphertext returned from the encryption process is then hashed using a valid hashing function such as SHA-256 [12] or MD5 [13]. This operation is shown in equation 1. It is important that the encryption algorithm, the hash algorithm, and the symmetric key be replaceable if any of them turn out to be weak or broken in the future.

$$Key = H(E_{SK}(D)) \tag{1}$$

In this equation, D is the object code, E_{SK} is the encryption algorithm using a secret key, and H is the hash algorithm. This method ensures that the key is a small fingerprint that is uniquely tied to the object code, but it prevents anyone from either figuring out the ciphertext (due to the one-way property of the hash) and prevents the user from creating his own key due to the use of the secret key that is shared by the verification server and the radio.

The security key could come in the form of a certificate that includes information about the server, the developer, the verification process, and the HMAC. The certificate security key helps establish accountability and trust.

Radio Platform

Of course, the whole system requires a radio platform to operate the software. Software defined radios are the most likely platform to be used, not only for SDR code, but also as the enabling platform of cognitive radios.

When the developer downloads the object code, he or she must also download the security key returned from the server. In order to authenticate the object code, the same process occurs inside the SDR as it did in the verification server. Equation 1 is used on the plaintext object code to create an HMAC. If the HMAC calculated here matches the HMAC of the security key, the SDR has authenticated the object code as coming from a valid verification server, and so the object code is accepted.

The SDR itself therefore requires a standard operating environment to perform the proper authentication measures. While this system requires widespread cooperation and deployment, the trade-off allows both freedom of development as well as regulatory compliance guarantees that do not currently exist.

5. SECURITY ANALYSIS

In this section, we review some of the most common cryptographic attacks and security risks to systems. Typically, developers and companies have little desire or incentive to break the regulations if they are trying to sell a product. It is for this reason that we suggested the in-house establishment of a verification server with proper rules governing the company's responsibilities and accountability.

For these cases where software developers are trying to, in good faith, create new SDR and CR technologies, the verification server acts such that it protects the developers from making a regulatory mistake and wasting their efforts.

Conversely, there is always a small percentage of the community that will want to tinker or break any system set before them. While they may not necessarily be malicious about it, our intention is to protect the integrity of the verification method its role as much as possible. While we can never guarantee a perfectly secure system, we can ensure security against most attacks and establish methods that will allow correction of problems as they arise.

5.1 Standard Cryptographic Attacks

Unlike OTA downloads, this verification methodology is not subject to a *man-in-the-middle attack*. While this is a concern in the transmission of the code to and from the verification server, standard and trusted methods of authentication and privacy can be employed here. If such an attack were to occur, it would not damage the integrity of the authentication method between the developer and the SDR.

By using accepted standards for both encryption and hashing, we ensure proper key and hash lengths to prevent a *birthday attack* from being problematic. If designed properly, the encryption and hashing algorithms are also replaceable in case they show any weaknesses in the future.

This paper has already addressed a *plaintext attack*. Even with the encrypted object code (a known plaintext), we have hashed it using an HMAC, so there is no known method (aside from brute-force) that will tie the known plaintext to the security key [15].

5.2 Hardware Attacks

Other attacks on security are when there is physical access to the secured system. Both timing and power attacks exploit the behavior of the system when performing the security operations. Timing and power use can directly indicate the type of operation being performed, which can lead to discovering the secret keys used [16, 17]. There are many known and developing methods to effectively combat such attacks, and any implementation of this proposed methodology should apply some measure of protection.

Another issue is bypassing the security operations altogether. If the security and authentication is performed in a different chip on the radio hardware than the actual processing of the code, there are methods to remove or bypass the security chip. To counter this, tamper-proof hardware has been used to ensure that both the security and processing are done on-board the same chip or that an attempt to remove or bypass the security chip would destroy the system board [18]. It would be easier to create a new system than hack this one.

6. SOCIAL ENGINEERING THE PROBLEM AWAY

While we can never guarantee a completely secure system (i.e., brute-force) we can at least provide one that prevents most technical problems. On the other hand, an environment that embraces research and development can help mitigate the desire to break the system. Here, we argue for a system of *development classes*, which are licenses by the FCC to perform certain radio operations on the SDR/CR platform.

A developer applies for a set of development classes for which he wishes to develop radio applications. Each development class is associated with a secret key, which is the key used in the HMAC authentication process. A key for each development class is pushed to all verification servers, and when a user is authorized to develop under a particular class or set of classes, the associated keys are pushed on to the SDR device using an OTA download mechanism.

Now, when a developer is attempting to verify his code, he also indicates his desired development class to the verification server. The HMAC uses the associated secret key to create the security key that is returned to the user. Now, when the developer downloads the verified code to the radio platform, the radio must have the same secret key to properly authenticate it.

Development classes should cover all areas of possible desired development, and different development classes could have different prices. Two important development classes are for amateur radio and general research, both of which should be free of charge (with proper identification of a ham license for the amateur class).

The amateur radio class offers a great set of frequencies and flexibility for research and development and playing with different waveforms and protocols [19]. Conversely, we argue for the introduction of a basic, open-ended development class that would allow a developer to create any waveforms and protocol where only the output power is restricted to some low power (< 0 dBm).

This concept has the potential to stimulate growth and experimentation as well as discourage abuse. The tinkerers who will play with anything, irregardless of the restrictions, will still be able to play with no added cost to them or the threat of introducing a major problem for the regulators. There will still be those who will try to break the system, because there always are, but we can limit this to a small percentage and let them have their fun.

The biggest flaw in this system is the threat of exposing a secret key. If one of these were to get out, the information would quickly spread over the Internet. Since almost all cryptographers advocate the use of open standards for security, once the key is known, its application in a known cipher would be trivial. To avoid this problem, we suggest a new set of keys be generated every so often (on the order of a month) and automatically pushed to all registered radio

platforms. Now, the security risk is in the OTA download mechanism used to push the set of keys; this must be secured to maintain the integrity of the entire system.

7. CONCLUSIONS

The system presented offers a way to allow development and innovation while adhering to the necessary restrictions placed on wireless communications. While there is still much work needed to realize this system as well as adoption by industry, government, and the development community, it offers a solution that is both secure and flexible.

One interesting area is the software verification system that is still a major research undertaking. This topic has great potential in the future of SDR and CR work, but in the meantime, the verification platform discussed in this paper will work with any system or independent auditor that guarantees regulatory compliance.

We have focused on the SDR/CR development community, but we feel it has application in other areas, too. SDRs and CRs are the first place to look to for a system like this because of the consequence that innovative designs may have on regulations, yet the same use applies in other, non-regulatory areas where open development is greatly beneficial. Intel has recently supported the efforts of open source software to develop applications on TinyOS for their Mote wireless sensors [20] exactly because the support of an independent development community would build new and creative tools and applications that might otherwise never be seen. Likewise, other areas might wish to see such creative innovation, but the nature of the product introduces safety and liability risks. This verification platform would uphold those restrictions and maintain compliance with certain bounds placed on the system by the original manufacturer.

While some in the software community might decry this solution, claiming that any restriction is too much, the intent is to satisfy both sides: the developers who need their freedom, and the regulators who need to uphold their laws.

7. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under awards *9983463*, *DGE-9987586*, and *CNS-0519959*.

9. REFERENCES

[1] M. Dillinger and R. Becher, "Decentralized software distribution for SDR terminals," *IEEE Trans. Wireless Communications*, vol. 9, pp. 20-25, 2002.

[2] SDR Forum, "Overview and Definition of Radio Software Download for RF Reconfiguration in a Technical and Regulatory Context," SDR Forum, 2002.

[3] L. B. Michael, J. Mihaljevic, S. Haruyama, and a. R. Kohno, "Security Issues for Software Defined Radio: Design of a Secure Download System," *IEICE Trans. Communications*, vol. E82, 2002.

[4] FCC, "Facilitating Opportunities for Flexible, Efficient, and Reliable Spectrum Use Employing Cognitve Radio Techniques," R&O: FCC 05-57, March 11, 2005.

[5] S. Haykin, "Cognitive Radio: Brain-Empowered Wireless Communications," *IEEE Trans. Selected Areas in Communications*, vol. 23, pp. 201-220, 2005.

[6] T. W. Rondeau, B. Le, C. J. Rieser, and C. W. Bostian, "Cognitive Radios with Genetic Algorithms: Intelligent Control of Software Defined Radios," in *SDR Forum Technical Conference*, 2004, pp. C-3-C-8.

[7] J. M. Chapin, "The Future of JTRS and its SCA: Lessons from Ada," *COTS Journal Online*, 2004.

[8] T. Ball and S. K. Rajamani, "Automatically validating temporal safety properties of interfaces," in *Proc. SPIN Workshop on Model Checking of Software*, 2001, pp. 103-122.

[9] H. Krawczyk, M. Bellare, and R. Canetti, "IETF RFC 2104: HMAC: Keyed-Hashing for Message Authentication," 1997.

[10] W. Stallings, "The HMAC Algorithm," *Dr. Dobb's Journal*, vol. 24, pp. 46, 1999.

[11] T. Dierks and C. Allen, "IETF RFC 2246: The TLS Protocol: Version 1.0," 1999.

[12] F. I. P. S. P. 180-2, "Secure Hash Standard," 2002.

[13] R. L. Rivest, "IETF RFC 1321: The MD5 Message-Digest Algorithm," 1992.

[14] B. Schneier, "One-Way Hash Functions," *Dr. Dobb's Journal*, vol. 16, pp. 148-151, 1991.

[15] B. Schneier, *Applied Cryptography*, 2nd ed., New York: John Wiley & Sons, 1996.

[16] E. English and S. Hamilton, "Network security under siege: the timing attack," *Computer*, vol. 29, pp. 95-97, 1996.

[17] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks," *IEEE Trans. Computers*, vol. 51, pp. 541-552, 2002.

[18] A. Raghunathan, S. Ravi, S. Hattangady, and J. J. Quisquater, "Securing mobile appliances: new challenges for the system designer," in *Proc. Design, Automation and Test*, 2003, pp. 176-181.

[19] J. Miller, "The "Ham and SDR Sandwich": Innovation and Enforcement Issues for Free and Open-Source Software on Software-Defined Radio Devices," *33rd TPRC*, 2005.

[20] R. M. Kling, "Intel Motes: Advanced Sensor Network Platforms and Applications," *IEEE Proc. IMS*, 2005.