It is I

An Authentication System for a Reconfigurable Radio

Arya Abraham

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Dr. Peter M. Athanas, Chair
Dr. Mark T. Jones
Dr. Joseph G. Tront

August 2, 2002
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia

Keywords: configurable computing, FPGA, security, authentication, biometrics,
differentiated service

It is I

An Authentication System for a Reconfigurable Radio

Arya Abraham

Abstract

The security of a radio system hinges on its ability to effectively authenticate a user. This work proposes a two-factor authentication scheme using a token and a biometric. The users' access rights are determined during authentication and the users are served only those channels of data that they are privileged to receive. The strengths and the weaknesses of the implementation in reconfigurable hardware are identified. The capabilities of the scheme are put into perspective by comparing it to a high-end authentication system and by evaluating the use of standardized APIs and low-end authentication devices. Modifications to the system are suggested to improve the level of security the scheme provides. Finally, a baseline study is carried out to measure the data processing performance of a radio developed in reconfigurable hardware, which uses the proposed authentication scheme.

To my parents

None of this would have been possible without their inspiration

Acknowledgements

While working on my research, many people have supported me with motivation, inspiration, direction and technical help. Please forgive me if I fall short of expressing my gratitude to all of them.

I thank Dr. Peter M. Athanas, my primary advisor, for helping me find the path I wished to tread and providing me with the opportunity to work on the projects that I needed to widen my horizons. I am indebted to him for his insightful advice and the encouraging word whenever I faltered. I thank Dr. Mark T. Jones and Dr. Joseph G. Tront for their support and motivation. I sincerely appreciate their willingness to serve on my graduate committee.

The people at the Configurable Computing Lab form one of the smartest technical communities at Virginia Tech. They have always been willing to help me out with ideas and advice. I express my gratitude to each one of them for providing a cordial and a highly motivating atmosphere at the Lab.

Special thanks goes to Scott Harper, Ryan Fong, Jason Zimmerman and Christian Schneider for their help. Scott, Ryan, Jason and Christian have contributed to the development of the radio mentioned in this thesis. In addition, Scott patiently read every draft of this thesis, suggesting changes that have helped improve the content. I also thank Dr. Jae Hong Park for the technical help he has rendered.

I convey my gratitude to all my friends in Blacksburg, who have been extremely supportive throughout my stay here. My life at Blacksburg would not have been the same without them. Special mention must be made of Kiran and Vandana. They were there to listen patiently to my ideas whenever I needed an opinion.

Finally, I thank my parents for being such a tremendous source of support and inspiration. They have borne all my attempts to 'repair' household gadgets with patience and have been there to hear me out and give me unprejudiced advice every time I needed it.

Arya Abraham

August 2002

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security is a key concern of system design in today's world. The goal in securing a system is to control access to the services provided by that system. A two pronged effort is required in this direction. First, the services exported by the system as well as the details of the implementation need to be protected from unauthorized access. Secondly, a user authentication scheme with a high level of confidence needs to be developed.

A unique combination of reconfigurable logic and biometric authentication is presented in this thesis. A user is authenticated with a token and a biometric. As part of the authentication process, the privileges of the user are retrieved from the token. This is used to select a configuration for the reconfigurable logic, which implements a radio in hardware. When this configuration is programmed into the reprogrammable logic device, a data processing channel is set up which provides the user with the level of information that he is entitled to receive.

The use of this combination of reconfigurable logic and authentication system is expected to protect the data as well as the details of the hardware design.

1.1 Motivation

The work presented in this thesis is driven by the need to implement access control in a reconfigurable hardware radio using biometric authentication. User access rights are determined during authentication. The radio is then reconfigured to tailor its data processing capabilities to the specific privilege level of the user.

1.2 Contribution

The contributions of this thesis towards incorporating secure authentication in a reconfigurable radio are listed below.

- Develop a user authentication scheme using a biometric and a token, which can determine the authenticity of the user with a high level of confidence and provide the system with the data that is required to gauge the privileges of that user.
- Make a comparative study of this biometric authentication system to authentication systems and components available in the market.
- Analyze the user authentication scheme proposed, on a link by link basis, to determine the possible strengths and weaknesses of each functional link. A solution is proposed to overcome each weakness that is identified.
- Implement a secure, re-configurable radio that uses the proposed biometric authentication scheme.
- Measure the baseline data processing capability of the radio, which is secured by this authentication scheme, on the basis of throughput, latency and reliability.

1.3 Thesis Organization

Chapter 2 begins by presenting a background of authentication techniques and their implementation in hardware. It contains an account of the research that has been carried out on the use of reconfigurable logic in system design.

Designing the authentication system is the main goal in this thesis. In order to use the authentication unit however, a working radio was required. The high-level system design of the entire radio is presented in Chapter 3. It outlines what is required for user authentication and a secure radio system, and analyzes the level of security afforded by the radio and the authentication unit.

The details of the secure radio implementation are presented in Chapter 4. The biometric authentication system that was develop here, is compared to the SecurID system developed by RSA Security, a leading provider of authentication systems in the market. The capabilities of the proposed system are put into perpective by comparing it to other commercial components. The results of the analysis are presented in Chapter 5. An analysis of the proposed authentication scheme, as it is implemented in reconfigurable hardware, is carried out and the results are described in Chapter 5. The strengths and the weaknesses of the scheme are identified and suggestions are made to improve the level of security attained. The steps taken to measure the performance of the post-authentication secure channel is outlined in Chapter 6. Performance is measured in terms of throughput, latency and reliability. Finally, the conclusions drawn from the results and possible extensions to this research are mentioned in Chapter 7.

Chapter 2

Background

The concepts used in this research are detailed in this chapter. User authentication is implemented using some combination of a password, a token and biometrics. Using a token for authentication requires that a secure communications channel be set up between the device and the token. The implementation of security measures in hardware relieves the system of the some of the burden of computing the cipher. It also protects the algorithm from an intruder. Reconfigurable computing hides the implementation of the algorithm in a programmable bitstream. This affords a low-cost upgrade for the hardware by allowing the bitstream to be changed when required.

2.1 User Authentication

A fundamental requirement of any secure system is the authentication of a valid user to the system. Access control, which is a set of methods that place constraints on the use of the system by the validated user are put into play only after the user is authenticated. Strong access control is meaningless without strong authentication.

Traditionally [1] an effective authentication system is a cumulative check of something the person has, something the person knows and something the person is. The first can be satisfied using a password or Personal Identification Number (PIN) challenge. Making the user insert a token that only he might be carrying, e.g. a smart card, can make the second check. Comparing some biometric measure of the person with a known value carries out the third check. This is a simplistic view of an authentication system. Other techniques are used in addition to the above to positively identify a user, such as encryption, challenge-response and session-id. They are dealt with later in this chapter.

2.1.1 Password Based Authentication

Password challenges have been the authentication method of choice in computer systems since their inception. The wide acceptance of passwords can be attributed, mainly, to the simplicity of the method and the ease with which it can be implemented. Specifically a password authentication system does not need expensive hardware like a token-based or a biometric system would require. In recent times the computational power available to the individual has increased many-fold, leaving password authentication systems more susceptible to sophisticated attacks [2], viz. dictionary based attacks, Internet worms, brute force attacks, hash based attacks and so on.

The validity of a password [1] is limited to the measures implemented by the user and the authentication system to ensure its secrecy. A password may be shared amongst co-workers or even written down somewhere visible as an aid to memory. An effort must be made by the user to prevent compromising a password. Similarly the authentication system has to protect the user password that is stored on the system for comparison. Since 1973, major operating systems that use authentication, such as UNIX, have incorporated hashing algorithms to protect passwords on the system. The methods used to protect the password have evolved

rapidly to keep up with the evolution of hacking techniques and the tremendous increase in computational power.

Transmission of the password from the user to the authenticating machine can be a loophole in an otherwise secure system. A sniffer on the network may pick up a password during transmission. Even if the password is encrypted before it is transmitted, there is a distinct period before encryption when it remains clear-text. This leaves a window in time during which a password might be compromised.

The probability of an attack succeeding increases with the weakness of the password. A password that is made up of dictionary words or one that can be pronounced easily falls into the category of weak passwords. Forcing a randomly generated password on the user increases the chances that is will be written down to aid memory, increasing the probability of a compromise.

The trend is to move to an authentication system that uses a combination of a password challenge with either one or both of the other two methods of authentication: token based authentication, biometric verification. The use of passwords has not been dropped completely because it is inexpensive and it allows the user to change the object that is checked if he perceives a security breach.

2.1.2 Token Based Authentication

Tokens range in complexity from a simple hand turning key to a smart card. The acceptability of smart cards as tokens is on the rise for the following reasons: they are not easy to replicate, they are easy to program for a specific application, they protect the base secret and can be made to destroy information they carry if tampering is detected [2]. In a smart card authentication process the unit that initiates a data transfer from the card and subse-

quently reads that information is know as the *terminal* . A smart card contains memory to store the base keys securely and a small processing unit to either perform encryption and decryption or to generate a one-time key from the base keys. Some smart cards are designed with specialized hardware [3, 4, 5, 6] for cryptography, random number generation and now even biometric authentication.

Smart cards and similar devices can be used in a token based authentication system in many ways. The token can provide a simple PIN to the system such as the id-cards used in low security applications. A generic token can be used to set up a secure communication link with the terminal [7] to allow the user to authenticate himself with a weaker secret such as a password or PIN. In the latest breed of smart cards, the smart card sends some information to verify the user such as a password, a PIN or even biometric information to the terminal in addition to setting up a secure link. This information is used to authenticate the user. The terminal need have no knowledge of the user before the smart card is inserted. For the protection of the user, it is imperative that the data transferred between the terminal and the token be encrypted. This encryption is also used to authenticate [8] the token to the terminal so that the terminal may be sure that the data it is receiving is coming from an authentic token. Authentication using encryption is described in detail in Section 2.1.4.

Java is fast emerging as the language of choice for smart card applications. A standard for Java based processors on the smart cards has been developed by Sun [9]. The standard describes a framework that presents a common design environment to developers of smart card application, for cards that run a variant of Java byte code. The set of Java classes available to the programmer is a subset of the standard Java distribution. Multiple Java applets can reside on the same card. The Application Programming Interface (API) provides access to cryptographic functions as well as reliable data transfer functions. Since Java is not machine dependent, the Java code can be recompiled for a smart card with different specifications with minimal change.

Using a token for authentication is not without its drawbacks. A token may be lost or misplaced. Using a token in conjunction with a PIN or password challenge can circumvent a compromise in security in case the token is misplaced. This will work as long as the PIN or the password has been kept securely and not shared with a malicious user who tries to use the stolen key. The level of security attained is much higher when a token is used in conjunction with a biometric verification or a password challenge, since only the authorized user can activate the system using this token.

A procedure for distribution of keys to both the token and the terminal is required, since the terminal has to know the key to decrypt the data sent by the token. In normal practice the smart card generates a one-time secret using the base secret for that authentication attempt. The base secrets should not fall into the wrong hands in case the token is lost. Keys are stored within the token. Hence the token should be designed to restrict access [10] to the private keys stored within it. If it were to be pried open, it should destroy the contents of the memory where the keys are stored. Apart from being physically pried open, smart cards are susceptible to a variety of non-invasive attacks [11, 12] ranging from power surges to protocol failures to playback of the data stream depending on the resource available to the hacker.

The *Java I-Button* [13], manufactured by Dallas Semiconductor, is a Java Virtual Machine (JVM) that communicates with a host using a proprietary API. The API is an extension of the JDK 1.1 standard that encapsulates all the functions of the Java processor on the button into Java classes. The I-Button is enclosed in a rugged, tamper-resistant, button-sized steel case. With 134kB of SRAM the I-Button can store multiple Java applets and information for user authentication, such as keys and biometric signatures. The SRAM is designed to erase itself if the case is pried open. A cryptographic co-processor speeds up encryption, decryption and key generation. Strong random numbers are generated within the I-Button for cryptography. Although it does not look like one, the I-Button is very much a smart

card. It can store user information, encrypt and decrypt data and communicate with a host in the same way a smart card would communicate with a terminal. Unlike most standard smart cards, the I-Button has a lot of memory and has a high level of tamper resistance.

The use of a token for authentication is considered stronger than the use of passwords. This assessment can be misleading since the use of a token has to complement the use of one of the other two methods. An authentication with a token is effectively a two-stage process. *Stage one:* authenticating the token to the system and transferring user data. *Stage two:* authenticating the user to the system using either a password challenge or a biometric verification.

2.1.3 Biometric Authentication

The quest for an authentication system that verifies a person based on something he is has led to the use of biometrics in authentication system. Historically, fingerprint matching in forensic investigations is the first known record of the use of biometrics. The technique has evolved over the years to a full-blown industry dealing with security. This industry has its own standards and standards committees. Fingerprinting is not the only biometric measure in use. Smith [2] refers to 14 different types of biometrics that have been used. Biometrics fall into two classes: *physical* and *behavioral*. Literature [2, 5, 14] identifies fingerprint scans, retinal scans and facial scans as the major physical biometrics, and signature scans and voiceprints as major behavioral biometrics. It is impossible to compare biometrics as a whole to any other method of authentication since there are so many different biometrics that may be used, each with its own idiosyncrasies and applications.

Figure 2.1 shows the block level flowchart of the operations in a biometric authentication system. The operations shown in the dotted box on the left are usually performed within

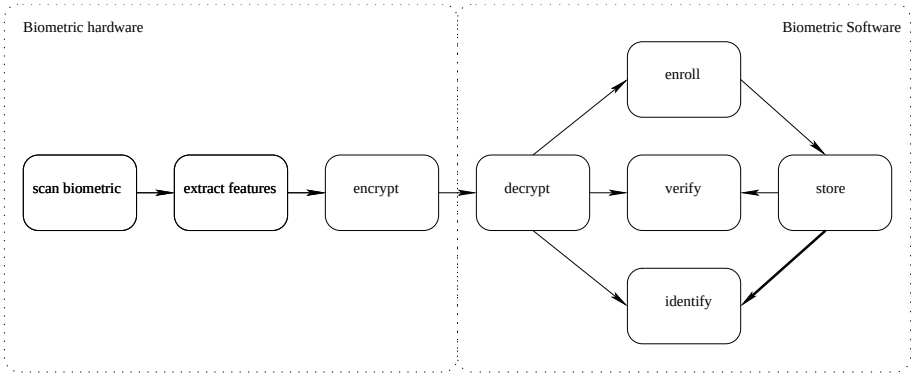

Figure 2.1: The flowchart of operations in a biometric system.

the biometric sensor unit. (Some recent applications like biometric enabled desktop PC login, trade off security for affordability by transferring the biometric signature across the boundary to the processor in the desktop PC.) The biometric is scanned and the *biometric signature* of the person is obtained by *feature extraction*. The dotted lines define security boundaries, described in [2], within which data is secure from prying eyes. Any data that flows across a security boundary has to be encrypted. Usually a biometric authentication system is divided into two units: the hardware and the software. The hardware reads the biometric from the scanner and performs very basic processing capabilities such as feature extraction. The software, on the other hand actually performs the authentication tasks, viz. *enrollment, verification, identification*. During *enrollment*, the biometric signature read from the person is stored and probably indexed. Multiple signatures of the same person may be recorded. The choice of recording multiple signatures depends on the authentication method chosen. Storage methods vary by implementation. In some applications, all the signatures are stored in a centralized database. In other applications, a person's signature is stored on his own personal smart card. *Verification* is an authentication method in which a person's biometric signature is compared to the signature in store. It is rare that the two signatures will be exactly the same. The system assigns a numeric value to how close the two signatures

are. If this value lies above a certain threshold, a match has been made and the identity of the user is verified. *Identification* is used to see if the user is a member of a set of users whose signatures are in a specific database. For this the biometric signature, that was extracted, is compared to each signature in the database. A set of all matches that cross a threshold is created and the closest match in that set is returned as the identity of the user. Sometimes the threshold is not crossed for a close enough match. In that event, the user is said to be not identifiable in the database.

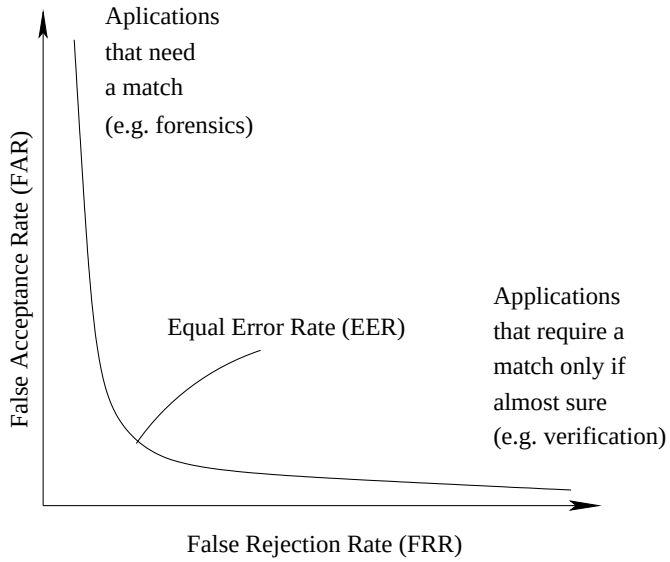

Figure 2.2: The receiver operating characteristics (ROC) curve of a biometric system showing the relationship between the FAR, FRR and the EER.

Any biometric system has a set of parameters [2] that characterize the accuracy of the operation of the device. The accuracy of the system is a trade-off that depends of the level of the threshold the system is willing to accept in a comparison operation. Experiments that count the number of errors made while authenticating a valid user is the *false rejection rate (FRR)* . This represents the number of times a valid user was rejected. Similarly, the *false acceptance rate (FAR)* is a count of the number of times an invalid user was accepted by the

system. Decreasing one will result in an increase of the other. Plotting *FAR* against *FRR* illustrates the relationship. The curve obtained, shown in Figure 2.2, is called the *receiver operating characteristics curve (ROC)* of the system. In most systems it is imperative to reduce the *FAR* to a minimum as long as the *FRR* remains within bounds. The scenario that a valid user is rejected often is preferred to one where an invalid user is authenticated. A reasonable starting point in arriving at an optimal setting of the *FAR and FRR* is the point on the graph known as the *equal error rate (EER)*. At this point the *FAR equals the FRR*. The actual operating point may be moved to the left or the right on the curve until acceptable levels of accuracy are reached. A low *EER* generally indicates that the system can deliver strong security with minimal false rejections. Wayman [15] has developed a detailed model for the analysis of the accuracy of a generic biometric system that takes into account five parameters of the system.

Fingerprint scans have been used as a biometric since the inception of biometric authentication. The method has its advantages and its disadvantages [14]. The technology for fingerprint scanning, verification and identification has matured over the years. The industry that supports the technology is stable. A company that invests in fingerprint-scan based authentication can be sure that their investment will not become obsolete overnight. Although an iris scan or a retinal scan produces much greater variety in the biometric signature, the technology behind a fingerprint scan is robust enough to ensure accurate verification close to 100% of the times it is used. The signature file that is created after the scan is between 200 to 500 bytes for most systems. This allows more than one signature to be stored on a smart card. Fingerprint scanning technology has its drawbacks. It tends to deteriorate with time and has difficulty authenticating a scarred or dirty finger. Human resistance to having their fingerprint scanned is often attributed to the fact that fingerprint scanning is often associated with forensic sciences. In spite of these drawbacks it continues to be the technology of choice since it is relatively robust as well as cheap.

Fingerprint scanning hardware is available in three flavors [14]: *optical scanners, capacitive solid-state scanners and ultra-sound scanners.* Ultra-sound scanners are extremely tolerant of dirt and dust between the scanner and the finger, but they are expensive to manufacture. Capacitive scanners produce excellent results in a clean environment and can even check that the subject placed on the scanner is alive. They are susceptible to damage from electro-static discharge, which is quite common in carpeted rooms or dry climates. Optical scanners cannot test a subject for life. Dirt and grime between the finger and the scanner bed can cause the authentication to fail. In spite of these drawbacks optical scanning is the technology of choice for AFIS. Optical scanners are resilient to electro-static discharge. They produce extremely high quality images, of the order of 500dpi, which assists the feature extraction process immensely.

The Secugen FDA01 is a versatile fingerprint-scanning authentication unit designed for embedded systems application [16]. The unit comprises an optical sensor and a processing unit to compare fingerprints. The optical sensor is resistant to wear and electro-static discharge. A latent print removal coating on the plate ensures that latent prints do not adhere to the plate. The sensor can capture fingerprint images at 500dpi, which is the industry standard. A proprietary matching algorithm on the processing unit produces match results with an FRR of 0.1% and an FAR of 0.01%. The FRR and the FAR of this device compare favorably to the results obtained by Wayman [17]. The processing unit communicates with the system that requests the authentication through a serial port. A serial command API is used to communicate with the processing unit. The processing unit can be configured to perform enrollment, authentication and identification.

Privacy is a concern most people have when confronted with a system that reads some physiological characteristic of their person [2, 14]. The collection of vast amounts of biometric data indexed with the person's identity, in some way or the other, brings up questions as to whether the information will be used ethically. Woodard [18] studies the concerns and

the possible solutions to the problem through decentralization of the database. The feature extraction of a biometric reading yields a signature that is, to a large extent, characteristic of the person. Usually this is a one-way process. The actual physiological characteristic cannot be reproduced using this information. This fact can provide a degree of security to the person, since an attack on a system using his signature can only be perpetrated by a playback of the signature rather than the reproduction of the physiological characteristic.

In addition to the usual applications, biometric authentication systems have been used in border control, licensing, national identification cards and prison management [18]. It is slowly finding application in the military where security is critical.

2.1.4 Cryptography in Authentication

In a seminal paper, Roger and Schroeder [19] illustrate how authentication can be achieved with encryption. Their study is not linked with any specific cryptographic technology available at the time the paper was published. They claim that if A and B are two principals on a network (A trying to authenticate B), then for the authentication to work, the following is required: *the message that A sends B must be comprehensible only to B and it must be evident to B that the message originated in A.* For this to work, A and B must have some shared secret, be it the each other's public key, a common symmetric key or the public key of a key distribution server. Gifford [8] observes that encryption can be used to assign capabilities, access control and information flow control by encapsulating data in encrypted objects. By creating multiple decryption keys based on a principal key, the authenticator can restrict access to some classes of users. A user can obtain a part or all of the capabilities associated with the object depending on the keys he possesses. Burrows and Abadi [20] cite three logical postulates that are required for the use of encryption in authentication. The system should have a mechanism to reject messages that were generated in a previous

authentication request since they are not pertinent to the current operation. The source of an encrypted message should be able to discard messages that it has put on the network. There should be enough redundancy in the message so that the receiver can tell whether it has the right key or not.

Encryption is used for authentication in most smart card based systems today. The encryption used may be *symmetric key algorithms* such as *DES, 3DES, AES* [21] or a *public key algorithm* such as *RSA* [22]. Symmetric key encryption algorithms are simpler to implement and require smaller keys than the public key algorithms for the same level of security. A study by Lenstra and Verheul [23] determined that a 72-bit symmetric key requires the same amount of computation power to crack as a 1028-bit public key. It has been determined, on the basis of the computational power required to crack the key, that keys of the aforementioned sizes are safe to use until 2002.

The *RSA* encryption scheme was proposed in 1978 [22] and later patented in 1982. Prior to RSA most cryptographic schemes utilized a symmetric key algorithm. Symmetric key algorithms have the disadvantage that both principals in the communication have to possess the same key. The transfer of keys from one location to the other was a security risk. RSA was the first public key algorithm that caught on in the cryptography community. By using a pair of keys, one confidential and the other public knowledge, the algorithm serves to make the distribution of keys, encryption and authentication, a simpler and more secure process. The generation of the key pairs is described in [21, 22]. A key comprises a modulus and an exponent. The modulus is same for both keys in the pair. The exponents differ. The basic operation in the encryption and the decryption process is the modulo-exponentiation, the exponent changing between encryption and decryption operation. The size of the modulus limits the size of the largest message than can be encrypted by the operation. If M is the message, e the encrypting exponent, d the decrypting exponent, n the modulus and M_e the encrypted message the encryption and the decryption operations are given in (2.1 and 2.2)

respectively.

$$M_e = M^e mod(n) \tag{2.1}$$

$$M = M_e^d mod(n) \tag{2.2}$$

The most common attack on RSA is factorizations of the modulus [21]. Since the public exponent and the modulus are known, the missing link is the private exponent. It is possible to factorize the modulus to arrive at the private exponent, but the operation is not trivial for large key sizes. A particular size is said to be strong for a key if the factorizations of the key take too long to be practical, given the computation capabilities of the day. An estimate of strong key sizes for an asymmetric encryption scheme like RSA is calculated in [23]. A 1024-bit key is considered strong enough for applications today [24] but larger keys will be required as the years progress. A redeeming feature of RSA is that the level of security does not depend on the size of the exponents used. Usually one exponent is set to 65537, 17 or 3 [21] to make computation simpler. It is recommended that a random number be placed within the encrypted message to prevent attacks on the system.

Public Key Cryptography System (PKCS) is an authentication scheme that uses RSA to transfer data in a secure manner [21]. The public key is distributed to the public domain while the private key is kept a highly guarded secret by the owner of the RSA key pair. Any message that is sent to the owner of the key pair is encrypted using the exponent and the modulus in the public key (2.1). The message can only be decrypted using the associated private key (Equation 2.2). To secure a communications channel, the key pair is generated and disseminated, with a level of confidence, to the principals at the ends of a channel. Data can be sent to the owner of the key encrypted to the strength of the RSA cipher. The

reverse, encrypting the data with the private key (by the owner) is known as digital signing. The portion of the data signed can be restored using the public key, (Equation 2.1). In our design, we have used this property of the RSA key pair to transfer data securely in both directions of a channel using a single key-pair.

To eliminate the risk of a security attack by playback [25] of a previously generated message, the n-byte block that is encrypted comprises a two byte tag (the first being 0), followed by eight bytes of random number and $n - 10$ bytes of message data. The random bytes ensure that even if the message is the same, the cipher text is different each time.

The PKCS standard [25] defines the implementation of four cryptographic primitives – encryption, decryption, signature and verification. In addition it describes multiple schemes for encryption and signing. The schemes are a combination of the primitives and other techniques to achieve a particular security goal. RSA is used as the basic encryption technique. RSA is combined with hashing functions in some schemes.

The generation of strong random numbers is a requirement for most cryptographic systems [26]. On a computer, it is impossible to generate true random numbers. Pseudo-random numbers are generated instead, using algorithms that are seeded with a value that changes each time the system starts up, e.g. system time. Seeding the random number generator ensures that the stream of pseudo-random numbers do not repeat every time the system is started. In hardware, pseudo-random numbers are generated using a Linear Feedback Shift Register (LFSR) [27]. In general, an LFSR generates a sequence of bits. The type of sequence generated depends entirely on the feedback taps in the design. Since there is a finite set of states that the register in the LFSR can take, it is imperative to ensure that the register cycles through every possible state, thereby generating a maximal sequence. The set of taps required for maximal sequence generation can be derived [27].

Maximal sequence LFSRs are used to generate Pseudo Random Number (PN) sequences.

The following properties have to be satisfied to classify an LFSR generated random sequence as a PN sequence [26].

- The number of zeros and ones should be as equal as possible per period.
- Half the runs in a period have length 1, one-quarter have length 2, ... , $1/2^i$ have length i. Moreover, for any length, half the runs are blocks and the other half gaps. (A block is a subsequence of the form ...011110... and a gap is one of the form ...10000001.... Either form is called a run.)
- The out-of-phase auto correlation $AC(k)$ has the same value for all k.

The three conditions listed above can be met with a proper selection of the feedback taps of the LFSR. Further, for use in a cryptographic system, the sequence should be very long. The length of a maximal sequence is $2n-1$ where n is the number of flip-flops in the LFSR. Special care must be taken to ensure that starting from a zero state will not set the LFSR to a stuck state.

2.2 Reconfigurable Computing

The implementation of an algorithm in a conventional system is usually tackled by using either custom logic or a processor. Until recently, using custom logic usually meant designing and fabricating an Application Specific Integrated Circuit (ASIC). An ASIC exploits the benefits of the advances in integrated circuit technology. This translates to the fastest or the most compact implementation of the logic. On the flip side, the entire design cycle takes very long and the ASIC, once fabricated, cannot be modified to perform a different function. The other option is to design the logic in software and embed it in a processing unit. This

method produces a system that can be reprogrammed at will, but the speed of the system is reduced, since the hardware implementation is always faster. Configurable logic bridges the gap between the software and the hardware implementation. It provides the flexibility of software and the speed of hardware.

A reconfigurable logic device is an integrated circuit that can be programmed to implement custom logic. The device is usually a regular array of configurable logic blocks embedded in a fabric of routing resources that is also configurable. Writing bits to configuration points program the configurable resources on the device. A configuration is a stream of bits, which, when written to a programmable logic device, maps the custom logic to the resources on the device. It is created from the hardware description of the logic to be implemented, using tools provided by the vendor or developer. The configuration points on the device are either memory units (usually SRAM) or anti-fuses. The configuration in an SRAM based device is volatile. It retains programming information until it is powered down. An anti-fuse, on the other hand, is a one-time programmable configuration point on the device. It has to be blown to make a contact between two layers in the integrated circuit.

Reconfiguration of a programmable device may be *static* or *dynamic.* When the programmable device is configured before the algorithm is run, it is known as static reconfiguration. Static reconfiguration is used for rapid prototyping to test logic before designing an ASIC. Dynamic re-configuration (run-time reconfiguration) involves changing the configuration, or parts of the configuration of the system, while the system is running. Dynamic reconfiguration may be triggered by a host or it may be data driven.

The first known configurable computing system was implemented by Estrin [28]. The machine was a hybrid of a general purpose processing element and configurable logic devices. The configurable logic devices were connected to the processing element as peripherals on a bus. The Field Programmable Gate Array (FPGA) introduced by Xilinx in 1984 is a

reconfigurable logic device. PRISM [29] introduced the concept of tightly linking the reconfigurable logic (FPGA) with the processing element to increase the data bandwidth between them. The program to be run on the processor was evaluated during compilation to extract functions that could be implemented in hardware. Compilation resulted in two programs, one for the processor and one to configure the logic. The program utilized the function implemented in hardware as if it were an extension of the capabilities of the processor. PRISM introduced instruction set metamorphism to configurable computing.

The Gate Array Reconfigurable Processor (GARP) [30] is another example of a run time reconfigurable system. The reconfigurable logic, in this case, is on the same die as the processor. It is arranged as arrays of logic blocks that have a direct path to data memory. As a processing unit, the reconfigurable logic in GARP was built as a control-flow architecture. In direct contrast, wormhole run-time reconfiguration [31] is implemented as a data-flow architecture. Data flowing through the system re-configures the programmable logic. The Stallion [32] is an implementation of a wormhole run-time reconfigurable architecture. Each packet of data that flows into the device contains a header with configuration information. The header in the data packet reconfigures the hardware, creating a custom logic path that processes the data that follows.

Virtual hardware is a term that describes the ability of a reconfigurable device to store multiple device configurations, which can be swapped within nanoseconds at run time. Each configuration in such a device is known as a context. Applications that use context switching take advantage of the logic multiplying capabilities of dynamically programmable logic to increase the effective density of logic on the chip. Some of the applications of context switching are described by Puttegowda [33].

Reconfigurable logic has wide application today. Processing intensive encryption algorithms can be off loaded to a hardware co-processor, which is optimized for the operation. This

application is dealt with later (Section 2.3). Reconfigurable information appliances [34] form a new category of devices that have reconfigurable logic embedded beside the processor. Depending on the data being served, the appliance can download and program the reconfigurable logic with the configuration for the codec required to decode the data. The lifetime of the appliance is extended because the appliance can keep up with evolving technology. Some appliances may be designed to load the initial configuration from an external device or a network. For a networked application it is essential that both the server and the configuration be validated. A corrupted or modified configuration could destroy the system.

2.3 Implementing Secure Hardware

Security is a key application in computer systems design in a world that is rapidly being connected by high speed networks. Most of the research is aimed at securing communications between entities on the network and protecting system access. Securing communications involves authenticating the entities to each other (Section 2.1) and encrypting the actual data communicated. Key sizes for both asymmetric and symmetric encryption algorithms keep increasing to offset the computational power available to potential hackers. Implementing the encryption engine in hardware is becoming a popular alternative to software implementation. An encryption engine implemented in hardware either serves as a co-processor or as an embedded unit in a custom architecture. Most of the common encryption engines have been implemented in an FPGA, e.g. 3DES [35], RSA [36] and Blowfish [37]. Each implementation shows an improvement in speed over the respective implementation in software.

The encryption of data and the authentication of the peer is not the only application of hardware towards building a secure system. Implementation of security in almost all systems that are available relies on authentication to switch the system from an idle state to a

running state. It is often forgotten that such a system may give away enough information to a malicious users to enable them to reverse engineer the system. Snooping signals on the bus will reveal the bus protocols used. Blythe [38] shows that even an ASIC may be reverse engineered using a process that involves selective etching and a scanning electron microscope. Given time and the resources, with this information the malicious user may be able to develop a system that performs the same tasks without the authentication.

The system logic may be implemented in an FPGA to avoid using an ASIC. Within the system, an FPGA is a black box and the signals within it cannot be easily probed. Further since the system is only configured when it is running the technique described in [38] to reverse engineer the logic cannot be used without physically accessing the system in the presence of an authenticated user. The security problems associated with bus snooping and layer-by-layer reverse engineering go away only to be replaced with the security hole created by a set of bit-streams that have to be loaded into the FPGA at run time. The format of the bit-streams used for most FPGA are proprietary and un-documented but this has not deterred the reverse engineering of the bit-stream in the past [39].

The option of using an encrypted bitstream to configure an FPGA is available on the VirtexII FPGA released by Xilinx [40]. The FPGA can be programmed with up to 9 56-bit 3DES keys. Once the keys are programmed, the device retains the keys internally until powered down. If the FPGA is programmed with an encrypted bitstream, the bitstream will be decrypted by the 3DES decryption engine within the FPGA. This protects the design encapsulated in the bitstream from reverse engineering because decrypting the bitstream has to precede reverse engineering. The hardware is not protected from malicious bitstreams since a clear-text bitstream can also be used to program the device.

The application developed in this thesis uses user authentication to provide access control to an encrypted stream of data. Reconfigurable hardware is programmed with an encrypted

bitstream to prevent reverse engineering of the system, adding a additional level of security. The system may be upgraded as technology advances by downloading an up-to-date encrypted bitstream from the data stream.

2.4 Commercial Authentication Schemes

User authentication is a high volume market. Almost every system administrator wants to ensure that his system is protected from unauthorized access. This is especially true if the system is connected to the Internet. There are so many user authentication schemes in the market that it will be impossible to list them all here. This section will contain a description of a few biometric-enabled authentication devices. As of today, all high-end biometric capable authentication solutions are extremely expensive and the details of the systems are generally not available. For the most part, these systems have biometric readers in terminals that extract a user's biometric signature and send it to a centralized server for verification as well as access right determination. The security of the channels of communication and the user input to the system cannot be analyzed since not much information about these systems is made open knowledge.

Quite a few hardware devices are available to the home/office user that incorporate a biometric scanner. These devices allow the user to login to a Personal Computer (PC) without having to enter a password. In most cases the only component of the authentication scheme that is implemented in hardware is the biometric scanner. Once the signature is scanned it is sent to the PC for identification. The software available on the PCs for these applications is usually very platform specific. Most of these units are gimmick devices which cater to the inquisitive casual user. They do not use very secure channels of communication with the PC. Very often a weak key or no key at all is used. Protection of the database of users is

either weak or not available at all. Since most of these devices are bought by individuals for their personal use, the biometric verification and identification algorithm implemented does not support more than a handful of users. An authentication system that is built in hardware must make use of one of the many Original Equipment Manufacturer (OEM) biometric authentication units that are available today, such as the Secugen FDA01 [16].

The RSA SecurID [41, 42] is an authentication system that uses two factor identification. Each user is given a unique token and is supposed to remember a pass code. The system has a centralized authentication server, which contains the pass code and the id of the token allotted to each user. The token is a device that generates a unique one-time id every minute. The id is based on the current system time as well as a 64-bit symmetric key that is programmed into the token. Each token has an identification number, which is used by the server to determine the 64-bit key embedded in that token. A combination of the pass code appended to the number generated by the token is sent to the authentication server for user verification. The server looks up its database, extracts the symmetric key associated with the user's token and checks the number that was sent along with the pass code. If the number generated using the symmetric key matches the number sent by the user, the server knows that the user is authentic. The server accounts for delays in network transmission when it checks the token generated one-time id. If the user id does not match the id calculated at the server, the server re-calculates the id using a value for time that is one to five minutes old.

Access agents control the access of system resources. They are available for all desktop machines (user login) and embedded in many networking hardware devices to protect access to that hardware. A system administrator can control the access privileges of each user by making the necessary changes to the user's profile at the server. The centralized server is designed for salability. The number of potential users runs up to one million and up to a thousand access agents can be controlled by one server. Up to twenty servers can be

networked together to either scale the system, or provide redundancy [42].

2.5 Support for Communications

At the physical layer, a communications network is unreliable. Data that is transfered over it may be dropped, routed to the wrong destination or corrupted. Corrupted data needs to be identified, so that it is not used by the receiver. The *CRC-CCITT* is a Cyclic Redundancy Check proposed by the ITU and used in X.25 communication networks [43]. A CRC is a value computed from the data by the transmitter. The value computed may be 16-bits or 32-bits long depending on the method of calculation. Before transmiting a packet of data, the transmitter calculates the CRC for the packet and attaches it to the end of the packet. When the packet is received, the receiver calculates the CRC value for the whole packet (including the CRC value attached by the host) using the same method. If the CRC value generated at the receiver is zero, the packet is said to have passed the CRC. This does not mean that the packet is free from all types of errors. Depending on the method used, a CRC can catch certain types of errors. The 16-bit CRC-CCITT detects the following errors.

- All 1-bit and 2-bit errors.
- All errors of odd number of bits.
- All error bursts of 16 bits or less.
- 99.997% of 17-bit burst errors.
- 99.998% of 18-bit or longer burst errors.

The calculation of the 16-bit CRC-CCITT is shown in (2.3). A message of length n is represented as a polynomial of degree $n-1$ and denoted by MP. The characteristic polynomial

of the 16-bit CRC-CCITT is $x^{16} + x^{12} + x^5 + x^1$. The CRC calculated is the remainder from the division of characteristic polynomial into the message polynomial (MP).

$$CRC - CCITT16 = \frac{MP}{x^{16} + x^{12} + x^5 + x^1} \tag{2.3}$$

Chapter 3

A System Overview

The focus of this thesis is on a unique authentication and identification technique for a secure radio platform. A three-pronged approach is used to both secure communications through the radio, and to secure the radio itself. First the data are encrypted with a standard encryption engine. Secondly, with the necessary precautions, configurable hardware protects the radio from attempts to reverse engineer the system. The final component in the three-pronged approach is user authentication. The implementation of the radio in reconfigurable hardware is only as safe as the configuration used to program it, which is secured by ensuring that the user provides the system with the keys needed to decrypt the configuration. This chapter provides a system-level overview of the radio, showing how multiple levels of security can be attained.

3.1 The Secure Radio

The *radio* described in this study is a reconfigurable system that sends a stream of data onto and receives a stream of data from the airwaves. To provide data stream security as well

as hardware security, user authentication and data encryption/decryption are built into the radio. The functions that the radio performs are broadly divided into three categories: *user authentication, hardware reconfiguration* and *data stream processing.*

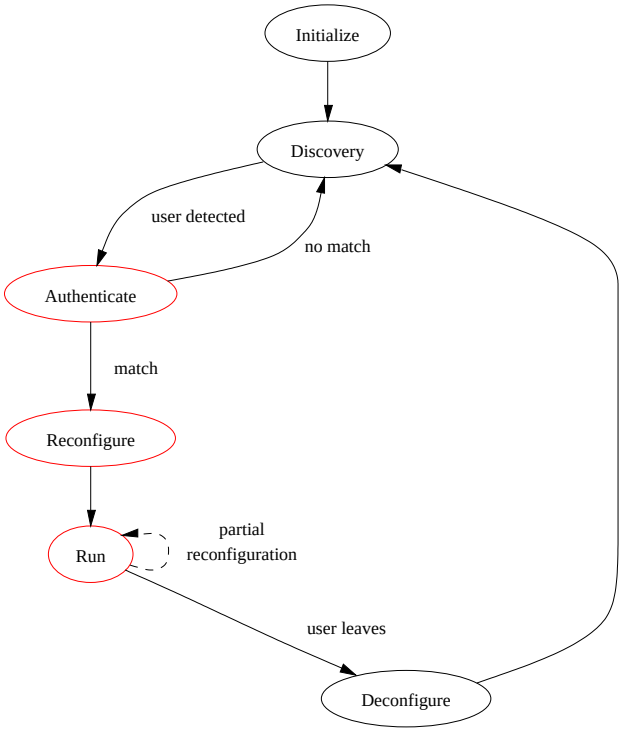

Figure 3.1: Flow chart for the operation of the Secure Radio.

Figure 3.1 shows the sequence of operations followed in the running of the radio. When the radio is powered up, it is first initialized. The peripheral hardware connected to the system is reset. It is brought to a known initial state. A discovery application is loaded into the system. This involves loading a configuration into the reconfigurable hardware and starting the peripherals that facilitate the discovery process. When a potential user inserts a token, the discovery process initiates authentication.

Authentication involves a comprehensive set of tests to verify that the user is authorized

to use the radio. Once the user is verified, a set of keys is retrieved from the token carried by the user. These keys are sent to the reconfiguration subsystem to select and decrypt a configuration for the radio. If the set of keys retrieved is valid, the radio is configured and a data path is set up. The radio can now be used be used as a communication device. The data path decrypts incoming data and sends it to a client. The client, too, can transmit data through the radio. When the user is done using the radio, they remove their token. This triggers the system to stop the radio and reconfigure the discovery and authentication configuration. In discovery state, the system waits until another token is inserted. The cycle repeats until the radio is powered down.

The discovery and authentication configuration is required to start up the system and put it into a state in which it can authenticate users. Therefore, this configuration is not encrypted. If it were to be reverse engineered, a potential intruder could gain complete control over the authentication process. The intruder would be able to snoop into the data received from the token. This data will reveal the keys used to decrypt other configurations that are loaded into the system after authentication. This is a limitation of the system that will be overcome when reconfigurable hardware which has the ability to load an encrypted configuration is used. The Xilinx Virtex2 [40] is an example of a reconfigurable device that has native support for encrypted configurations. For this to be usable, the decryption keys must be loaded into the device before it is shipped. The keys that are stored in the device must be held in memory perpetually by providing the device with a battery power source. If the power to the device ever fails, the keys will be wiped out and the device will not function with the encrypted configurations provided.

The block diagram of the radio system is shown in Figure 3.2. Each block in the figure represents a category of radio functions mentioned earlier. The shaded region covers the components of the system that are secure. Lines that cross the boundary indicate data transfers that have to be secured. The *Authentication Unit* performs discovery as well as

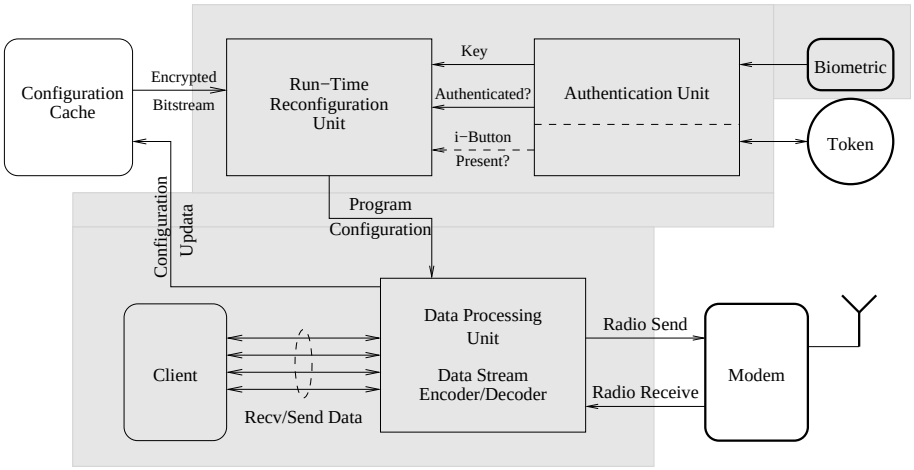

Figure 3.2: System block diagram of the Secure Radio showing the individual subsystems, their interaction within the system and their interaction with the environment.

authentication. A token and a biometric scan are required to authenticate a user. If a user is authorized to use the system, the keys extracted during the authentication process are sent to the *Reconfiguration Unit*. An encrypted configuration is read into the *Reconfiguration Unit* from the cache memory and decrypted with the set of keys obtained. The resulting configuration is the *Data Processing Unit*. If the user is authorized to use the configuration, it is loaded into the system and the radio is ready to transmit and receive data. The *Data Processing Unit* decrypts data received from the radio modem and sends it to the client. It is also possible to send data in the opposite direction, from the client to the radio modem. The *client* is an abstraction of a user of the data-stream. The client could be as simple as a gateway to a local network of computers, or as complex as a hand-held device connected directly to the radio.

Security is built into the radio in three layers: *authentication, reconfiguration* and *data processing.* Figure 3.3 shows the interaction of the three layers in a block diagram. Each block in Figure 3.3 forms a key subsystem of the radio. These subsystems are described in

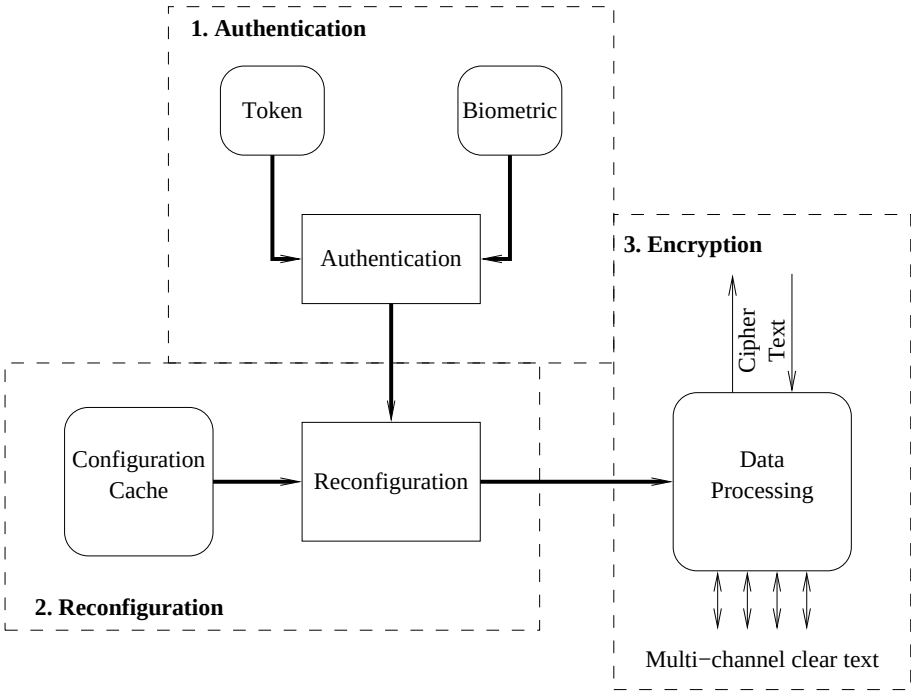

Figure 3.3: Security features built into the Radio

detail in the sections that follow.

3.2 The Authentication Subsystem

The security of the radio hinges on efficient user authentication. A *user class* is defined as a group of users with the same access privileges. Each user class has at least one configuration associated with it. The user class is determined during authentication. If authentication succeeds, the radio is configured with the correct configuration by the reconfiguration subsystem (Section 3.4) to set up the data path (Section 3.3). The radio remains in operation from the time the user is authenticated until the time he removes his token. The removal of the token indicates that the user is done using the radio. The discovery and authentication

configuration is then re-loaded and the system lies idle until the next user inserts a token. Figure 3.4 shows the block diagram of the *Authentication* subsystem.

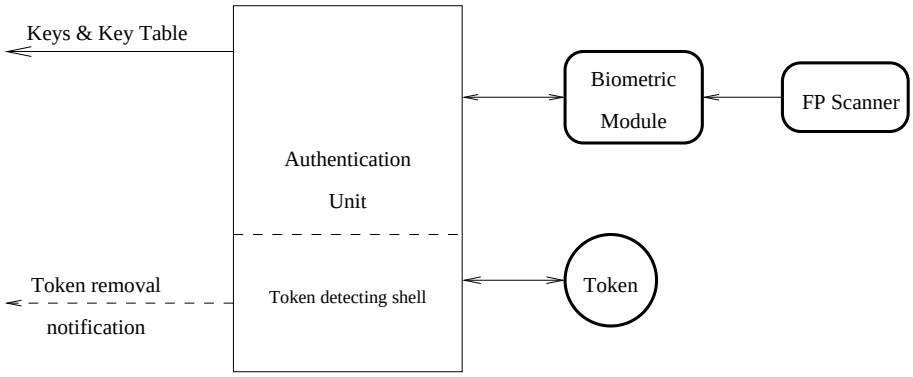

Figure 3.4: Block diagram of the *Authentication* Subsystem.

The *authentication scheme* relies on a token and a biometric to verify the user. The token contains a set of key table entries and a biometric signature of the person authorized to use the key. Both the radio and the token verify the authenticity of the other. If the radio accepts data coming from any token without validating it first, the system is open to malicious attacks. The radio receives keys, user class information and a biometric from the token. The wrong set of key and user class information could be sent to the radio. It could even be fooled into authenticating an unauthorized user. Similarly if a token assumes that the source of the challenge that it received is an authentic radio, it will respond even if the system that challenged it is not authentic. An attacker could use this information to either crack the encryption scheme used or to launch a replay attack [21]. Either scenario results in a security hole.

The system uses *encryption* to authenticate the token to the radio and vice versa. The token and the *Authentication Unit* have a shared secret. If the *Authentication Unit* can decrypt a communication, then it is assumed that the communication originated in the token. Similarly, if the token can decrypt a communication, then it is assumed that the communication

originated in the *Authentication Unit*. Strong encryption keys are used to ensure that the encrypted communication is not compromised [21]. Symmetric key encryption schemes require a key at least 78 bits long to be considered safe (Section 3.5). A key size of 1024 bits is considered safe for applications using public key encryption [21, 23]. The public key algorithm requires a larger key-size than a symmetric key algorithm because the public key is publicly available, making it susceptible to factorization attacks [21]. In the peculiar case where a public key algorithm is used, but the keys are kept secret, the modelling of possible attacks on the system is similar to the case where a symmetric key is used. Since factorizing the key is not possible if the keys are kept secret, a brute force attack will have to be used. Key sizes smaller than 1024 bits, closer in size to those used in the symmetric key algorithms, should be safe.

After the token has been authenticated, the *Authentication Unit* must verify that the user is authorized to use the token. The user is verified by running a biometric verification of her fingerprint. Since a large number of users may be authorized to use the radio, the storage of the each user's biometric signature in the radio is not feasible. The memory requirement would be too great for the system. Further, if even one radio is compromised, the biometric signature of every possible user would be compromised. To circumvent this, the user's biometric signature can be stored on the token assigned to her. From a security perspective, storing the biometric on the token is valuable, since the signature is never transferred out of the token in clear text and is always in the possession of the user [18].

Figure 3.5 and Figure 3.6 show flowcharts of the steps involved in authentication. The *Authentication Unit* sends the token a challenge containing a nonce. A *nonce* is a one-time session identifier determined at the start of the authentication process. The nonce is embedded in each packet that is sent. It must be some number that changes every time authentication is run, e.g. a random number. The token receives the challenge packet and validates it. If the packet is not valid the token does not respond to the challenge. If it is,

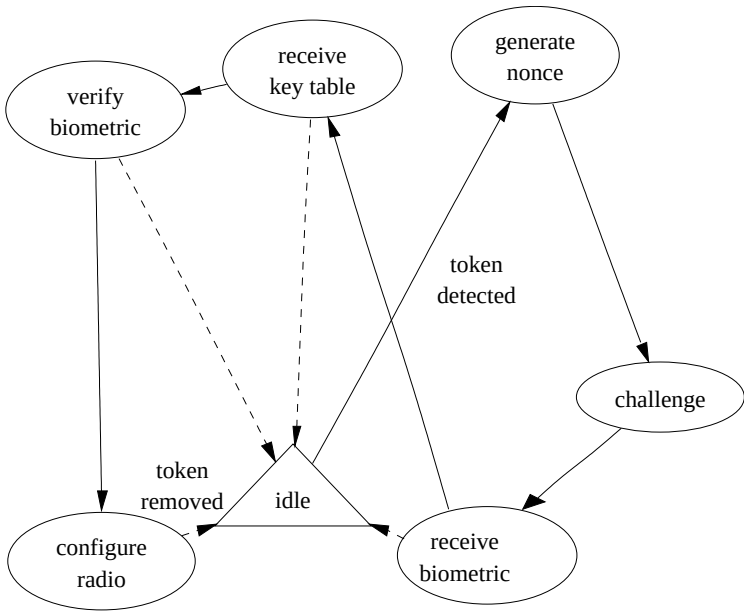

Figure 3.5: Flowchart of the steps performed for authentication in the *Authentication Unit*. Dashed lines show the route taken if validation/verification failed at a given step.

the token extracts the nonce from the packet. The nonce will be embedded in all further communication with the *Authentication Unit*. The token responds to the challenge sent by *Authentication Unit* with a set of packets that contain a biometric signature of the authorized user and a key-table to be used for reconfiguration. The *Authentication Unit* validates each packet that it receives from the token. If valid, the data in the packet is assembled to form a biometric signature. After receiving all packets that comprise the biometric signature, the *Authentication Unit* assembles the key-table from data that arrives in the subsequent packets. The key-table is written to the *Reconfiguration* subsystem. It is used to decide which radio configuration to load and which key to use to decrypt the configuration.

The biometric signature assembled by the *Authentication Unit* is sent to the attached biometric module for verification. The module scans the fingerprint of the user and compares it to the biometric signature sent by the *Authentication Unit*. If the user is verified, the *Recon-*

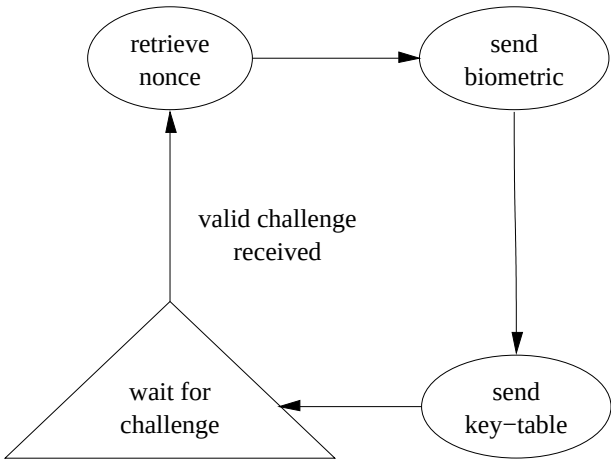

Figure 3.6: Flowchart of the steps performed for authentication in the token.

figuration Unit is enabled and the radio is configured and started up. If at any stage in the authentication process a received packet or a biometric cannot be verified, the *Authentication Unit* will go back into the idle state and wait for the next user.

While the radio is running, a shell of the discovery application is retained. The logic in this shell waits for notification that the token has been removed. The removal of the token signals that the radio is not required anymore. The reconfigurable hardware is re-programmed with the full authentication and discovery configuration and it waits for the next user. There is no trace of a radio in hardware when it is not being used by an authenticated user.

3.3 The Data Processing Subsystem

The path that a stream of data takes as it passes through the radio, when it travels between the client and the air-waves, is called the *data path.* The data path is a bi-directional stream. Figure 3.7 shows a block diagram of the data path. Data that originates at the client flows

through the radio and is broadcast on the air. Similarly, a message that has been broadcast by another radio or a central *base station* is received and sent to the client. The stream of data that flows through the radio is referred to as the *data-stream.* In normal operation, the radio processes two data-streams: one broadcast to a peer, and the other received from a peer.

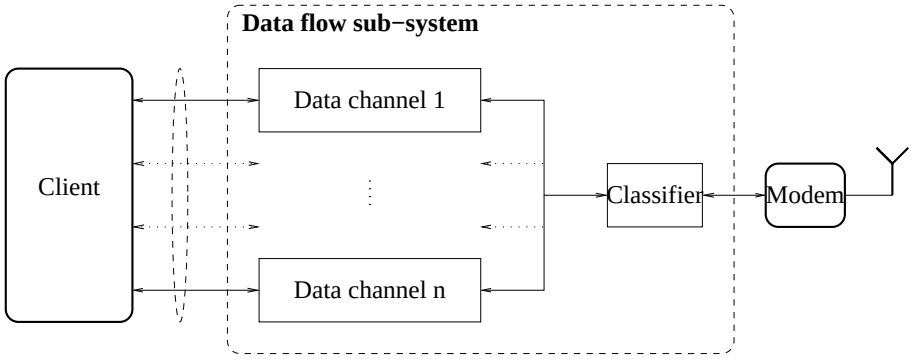

Figure 3.7: Block diagram of the Data Processing Subsystem.

The *client* in this subsystem is the infrastructure to which the radio is connected. The radio serves as a link between this client and clients attached to other radios within range. The client may be a single user terminal that has the ability to serve data and interpret received data. The client may also be a network of machines that use the radio as a gateway to another network. The functionality of the radio does not differ in either case. The radio provides a secure link between the clients.

Methods for securing communication through encryption are described in [21]. The data-stream is encrypted to keep data secure over the air. Encryption also protects the data from tampering. Checks are put in place to ensure that invalid data is not processed by the radio and presented to the client as valid data. A CRC can be calculated for data that is transmitted. The receiver compares the received CRC value with one that it generates. Data that does not pass the check is considered invalid and is dropped by the radio. The

client is not informed of data that is dropped.

Data are received and transmitted in packets. A header is added to each packet that is transmitted by the radio. This header contains system specific information that is used by the receiver to collect and interpret the data it receives. The receiver strips the header off the packet before it is relayed to the client. Since the header contains information that is critical to the operation of the radio, it must be checked for integrity before it is processed. A header checksum is a simple, yet efficient way to implement this check.

A wireless link is fallible. Packets of data may be dropped or corrupted over the air. The radio can be setup for reliable communication over the air. The transmitter requests an acknowledgement for every packet that it sends. After sending a packet it waits for the acknowledgement. If it does not receive an acknowledgement within a preset time, it assumes that the packet was either lost or corrupted. The last packet is then re-sent. A reliable link that is set up in this manner is slow. The transmitter must wait for an acknowledgement between the transmission of two packets. The wait comprises the round trip time and the processing time at either end of the communication link. Therefore acknowledgements can be disabled if the application does not requires it. Further, the throughput in certain classes of applications is adversely affected if acknowledgements are enabled. The scenario where there is one base station broadcasting data to multiple receivers is one such application which will not perform well.

Another security feature of the radio is the provision for differentiated service. *Differentiated service* is a term used to describe a data dissemination scheme in which different users are given access to portions of the data-stream depending on their access privileges. Each user's access privilege is determined during authentication (Section 3.2). Differentiated service is implemented by associating each packet of data with a virtual channel. The channel that a packet is assigned to determines the key used to encrypt it, which is also the key that will

be required to decrypt it.

Each data packet can be encrypted with a different key and possibly a different encryption engine. The number of unique keys supported by the radio can be increased to the extent allowed by the communications protocol and the size of the reconfigurable devices used. This can be achieved by replicating portions of the hardware. Effectively, this is akin to creating multiple channels through the radio. Each *channel* processes the data flowing through it separately from the other channels in the radio. A single wireless link is multiplexed for the transmission/reception of different types of data, each processed individually through a dedicated channel. The *classifier* is a hardware unit in the data path of the radio that time multiplexes the wireless link. It receives a single stream of data from the radio modem and feeds it into the appropriate hardware channel. Fields in the header indicate which channel to use to process each packet. For data flowing in the opposite direction, the classifier combines multiple streams of data into a single stream that is relayed to the radio modem.

It is not required that the data flowing through every channel of the radio be encrypted. The data in a channel may be processed in some other manner. One channel is dedicated to receiving hardware updates. This channel is used to transmit updated bit-streams from a central broadcasting unit to each radio in the field. Since this updated bit-stream is stored locally, it must be stored encrypted. Decrypting the bit-stream within the radio is an overhead that can be avoided, hence the channel is not encrypted/decrypted in the usual way. Bit-streams transferred over this channel must be pre-encrypted by the client that transmitted it. An updated bit-stream is used to change the keys needed for authentication, add features/processing elements to the radio, update features of the radio, change the cryptographic engine or change the keys used to encrypt and decrypt data in each channel.

3.4 The Reconfiguration Subsystem

The radio is built on a reconfigurable hardware platform to provide the flexibility of run-time reconfiguration as well as to hide the implementation of the radio. The ability to reconfigure the hardware while the radio is running is used to control access to the data stream received by the radio modem. Once a user is authenticated (Section 3.2), the reconfigurable hardware is programmed with an appropriate configuration. The chosen configuration depends on the user class of the authorized user. This is decided on the basis of the data retrieved from the token during authentication.

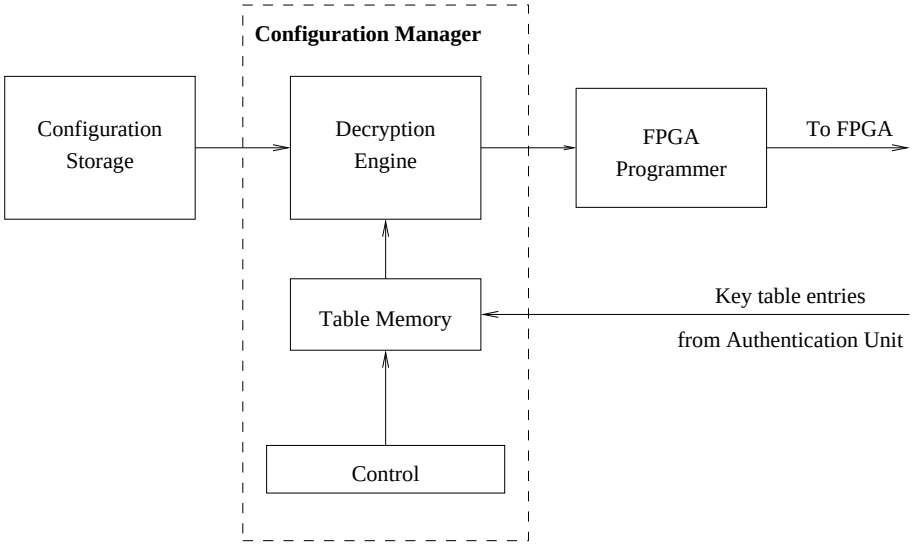

Figure 3.8: Block diagram of Reconfiguration subsystem.

Figure 3.8 shows the block diagram of the reconfiguration subsystem. The *Configuration Manager* is a key component of this subsystem. All the functions that are required to select and decrypt a configuration are grouped together to form the configuration manager. It comprises the *Decryption Engine*, the *Table Memory* and the *Control*.

Every configuration that is available to the radio is stored in the *Configuration Store*. Every

user need not have access to all the configurations in the store. The radio allows the items in the store to be updated while it is running. Updates received from the air are be written directly to the store. The client is allowed to have configurations that are specific to its requirements. It sends the radio a configuration that it has cached locally. Every configuration in the store is encrypted. This protects the configuration from tampering and unauthorized access. Since the key used to encrypt the configuration is secret, even to the client, the client must obtain the configuration from a central distributing authority.

Once a user is authenticated, the token sends the radio the biometric signature of the user followed by a key-table. The *key-table* contains an entry for each configuration that a user can access. Each entry in the key-table contains an index into the configuration in memory and the key to decrypt the configuration. There are other fields in the key-table entry that are not filled in until the table is in the system. Key-table entries must be considered classified information. They contain information that could be used to decrypt and hence reverse engineer the configurations. They must never be accessible as clear-text outside the system. Therefore, they are encrypted by the token before being sent to the *Authentication Unit* and it is stored in memory within the configurable logic. The memory used for key-table storage is called the *Table Memory*.

Before a configurable logic device can be programmed with a configuration, the encrypted configuration must be decrypted. A standard cryptographic algorithm with a symmetric key is used. The key used to decrypt the encrypted configuration is retrieved from the token during authentication. A valid key is needed to process the encrypted configuration. Decrypting the encrypted configuration with an invalid key will result in a corrupt configuration which will not be used for programming.

Once a configuration is selected and decrypted, it must be programmed into the reconfigurable device. The *FPGA Programmer* is the logic that programs the device. The details of

the programmer depend on the reconfigurable devices used. A different programmer will be needed for devices procured from different vendors or for different devices sold by the same vendor.

3.5 System Security

The previous sections dealt with an overview of the system from the standpoint of implementing a secure radio. In this section, the points where the system can be potentially compromised are identified and the methods used to thwart attacks on the system at these points are described. Figure 3.2 shows an overview of the system with the secure regions in the shaded box. Three communications lines cross this box. The first is the communication channel between the *Authentication Unit* and the token. Securing this channel of communication is described in more detail later in this section. The second communication channel that crosses the boundary of the shaded region is the communication between the Configuration Store and the radio. Configuration data that crosses this boundary is always encrypted and is hence secure. The third channel that crosses the boundary is the flow of user data into the system from the radio modem and out of the system to the radio modem. Since this data is always encrypted, the security risk is limited by the encryption method used and the size of the keys used.

Figure 3.9 shows the levels of security that are implemented in the system in the form of a pyramid. User *Authentication* is at the base of the pyramid while Data Security is at the apex. A compromise in the User *Authentication* process will result in the compromise of the keys used to encrypt/decrypt configurations. If a device configuration is compromised, the method for processing data that flows through the radio can be reverse engineered at will. Hence data transfer, too, is compromised. The compromise of a process at a lower level

in the pyramid results in the compromise of all processes at higher levels in the pyramid. Security of the User *Authentication* process, which is at the base of the pyramid, is essential to the security of both the configuration and the data. Configuration and data are secured by encrypting individual data streams. The authentication process, on the other hand is complex and involves multiple devices and modules in the system, some external and others internal. This section mentions the methods by which security may be compromised in the *Authentication Unit* and how it is designed to counter them.

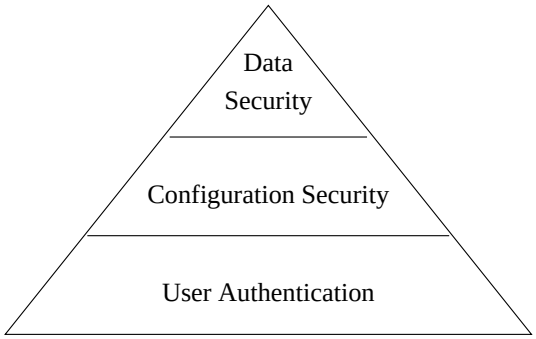

Figure 3.9: Pyramid of Security for the Radio

Possible threats to the authentication protocol can be classified under two headings – host compromise and communication compromise [21, 44]. *Host compromise* includes all attacks on the hosts that are trying to authenticate each other. In this system the two hosts are the *Authentication Unit* on the FPGA and the *token*. The *Authentication Unit* is immune to a host attack since no external unit can take control of the execution of the protocol within the FPGA. This is because the configuration is encrypted and the *Reconfiguration Unit* is embedded in the hardware. Methods of compromising the data on a token are described in [2, 10, 11, 12]. A token that is designed to thwart host attacks is chosen for this system. The token must be tamper-resistant and the signals on data busses internal to the token must be protected from external snooping.

Communication compromise includes all attacks on the communication exchanged by the two authenticating hosts. The authentication protocol requires data to be transferred between the two hosts. As in any secure system, the information that is exchanged must be encrypted to protect it from a passive listener on the network. A *passive listener* reads the data that is exchanged between the two hosts and tries to decipher the data from what is known about the authentication protocol. A security scheme must never rely on an algorithm or a protocol for data security [21]. The security must lie, entirely, in the keys used. It must be assumed that the authentication protocol is known to everybody, including the passive listener. The strength of the keys used in the cipher determine the security of the data transferred between the two hosts. Schneier [21] shows that if the keys used in the encryption scheme are secret, the strength of the cipher depends on the time required to mount a brute force attack on the cipher-text. This is proportional to the number of iterations required to run an exhaustive search of the key space, and inverses proportional to the speed of computing and the time taken to perform one encryption/decryption (3.1).

$$strength \propto \frac{iterations}{computation\_power \times computation\_speed} \tag{3.1}$$

For a key of length n, the average number of operations required to determine the key is given by half the total number of possible keys in the key space (3.2).

$$iterations = 2^{n-1} \tag{3.2}$$

The computation power available to an individual/group is increasing in leaps and bounds. Moore's Law of Computing suggests that the computing power doubles every 18 months. Lenstra [23] devised a method to calculate the number of MIPS that is considered infeasible to attain given the technology in a particular year. Using this and Moore's Law, Lenstra

calculated the key size required to attain the same level of security in a given year, as that afforded by the Data Encryption Standard (DES) in 1982. The values in Table 3.1 are calculated by applying this model to a range of key sizes. The estimates of the year that are tabulated against key sizes 128-bits and larger in the table are optimistic since they assume that no new method will be discovered in the interim to break the key and that Moore's Law will hold over the entire period. Despite the optimistic estimates, the model clearly shows that a 128-bit or larger key will not be threatened for some time to come. Therefore the selection of a key with size 128-bits or larger will clearly be a strong key for the application described in this thesis.

Key-size	*Year*	*Key-size*	*Year*	*Key-size*	*Year*
56	1982	78	2010	128	2079
63	1990	86	2020	256	2176
70	2000	96	2030	512	2370

Table 3.1: Estimating the year until which a specific key-size, in bits, will give security comparable to DES in 1982. Values calculated using Lenstra's model.

An authentication protocol is potentially susceptible to a man-in-the-middle attack, a replay attack and a chosen-cipher text attack [21]. Public key authentication protocols are prone to man-in-the-middle attacks. In such an attack, shown in Figure 3.10, Alice sends Bob her public key and Bob sends Alice his public key. Chris receives the message sent by both and sends his own public key to both Alice and Bob. Alice and Bob begin to communicate using the public keys they received, each assuming that the key they received was sent by the other. In the meantime, Chris decrypts each message (since it was encrypted with his public key) and then sends it along to the recipient, encrypted with the recipients public key. In this way Chris decrypts and has the potential to modify each message sent between the Alice and Bob. An asymmetric key system is prone to a man-in-the-middle attack only

if the one key in each key pair is made public. In the radio described in this thesis, the keys are always kept private. Hence there is no 'public' key available for a man in the middle attack.

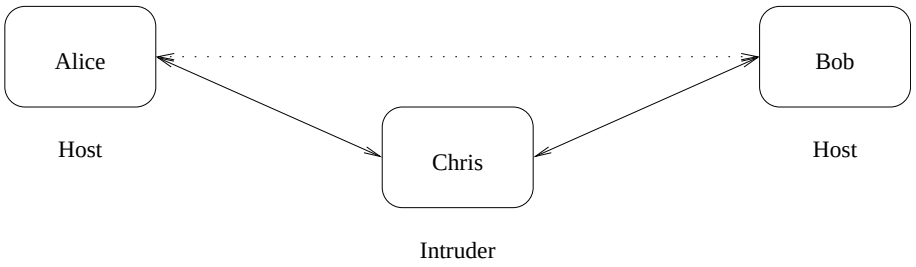

Figure 3.10: Illustration of the Man-in-the-middle attack.

In a replay attack the intruder stores valid messages that are exchanged by two valid hosts. These messages are played back to a host by the intruder at a later time with the hope of gaining entry into the system. The only way to prevent an attack of this kind is to pad the message that is passed between the hosts with a one-time session id. This session id may be generated from a time-stamp, but this requires both hosts to have synchronous clocks. Another way to create a one-time session id is to have one host generate a random number from a maximal sequence Linear Feedback Shift Register (LFSR) (Section 2.1.4). The LFSR generates a non-repeating sequence of bits that are shifted together to form a pseudo-random number. The duration after which the number generated by the LFSR repeats will depend on the size of the LFSR. The sequence generated has a period 2^{n-1}, where n is the number of bits in the sequence. The period after which the sequence will repeat for different n is listed in Table 3.2. The period is calculated in years with the assumption that an n-bit maximal length free running LFSR is clocked at 40MHz. A maximal length LFSR with a size greater than 54 bits has a period in tens and hundreds of years. A 64-bit LFSR will only repeat after seventy-three centuries. Using a maximal length LFSR of size greater than 58 bits to generate a session id will ensure a session id that will be unique for at least a hundred years.

n	*Years*	*n*	*Years*	*n*	*Years*
48	0.11	54	7.14	60	456.67
50	0.45	56	28.54	62	1826.69
52	1.78	58	114.17	64	7306.78

Table 3.2: Estimating the periodicity of the sequence generated by a maximal length LFSR of size n.

Since the LFSR is a finite state machine, a hacker can try to compromise the system by restarting the machine and replaying the message at a time that matches the time it was originally sent. This effort is thwarted by using a biometric for authentication. Since a biometric signature is encapsulated in the messages sent during authentication, the user who was authenticating himself to the system when the messages were captured, must be present for the system to authenticate his biometric. It is assumed that the module that performs biometric authentication communicates with the *Authentication Unit* over a secure channel that can thwart a replay attack. Biometric verification modules than can be customized are available in the market. The same scheme that is used to secure communication with the token should be implemented for communication with the biometric module. Therefore the system cannot compromised by restarting it without the valid user being present.

Another form of attack on an authentication protocol is the chosen cipher text attack. A malicious intruder sends either the *Authentication Unit* or the token a random message and analyses the response to break the cipher. This form of attack will not succeed on the either host, since both hosts look for a predetermined header in every packet of data that is received. The *Authentication Unit* will stop processing packets that it receives after the first invalid header is encountered. The token, on the other hand, will process each packet that it receives, and determine whether it is a valid challenge by checking the header. If the packet

is not a valid challenge it will not respond. If a replay attack is mounted on the token, the token will recognize it as a valid challenge and respond with the same packet every time. This will not reveal any new information to the attacker.

The features of the authentication scheme described in this section make the scheme secure to known forms of attack. This does not mean that the system is invincible. A technique for deciphering encrypted messages may be found that is computationally simpler than those available today, leaving the system more vulnerable to attack than it was originally designed to be.

3.6 Summary

In this chapter, an system level overview of the radio was presented. The radio combines the use of reconfigurable logic and a biometric authentication system to provide an extra level of security. No details of the actual implementation were revealed. The system was broadly divided into three subsystems – *Authentication*, *Data Processing* and *Reconfiguration*. The system was analyzed to determine the channels of data that need to be secured. Possible attacks on these channels were described. By choosing the right key-size and implementing an authentication system that thwarts replay attacks, it was shown that the system could be made extremely secure.

Chapter 4

Implementing the Radio

Chapter 3 outlined the radio without delving into the details of the actual implementation. The details of the implementation of the radio on the SLAAC1V reconfigurable architecture are described in this chapter. First, the SLAAC1V architecture is explained to help the reader comprehend design decisions made later in this chapter.

4.1 The SLAAC-1V Reconfigurable Platform

The *SLAAC-1V* reconfigurable platform [45] was developed by the Information Sciences Institute (East) of the University of Southern California. It is comprised of three Xilinx Virtex XCV1000 FPGAs along with hardware interconnection networks, logic to program the FPGAs, a Peripheral Component Interconnect (PCI) bus master interface to connect it with a host.

Figure 4.1 shows a system-level block diagram of the SLAAC-1V board. The three Virtex XCV1000 parts are named *X0*, *X1* and *X2*. They are connected to each other via a systolic

bus and a crossbar bus, each bi-directional and 72 bits wide. *X0* contains the PCI interface through which a host communicates with the board to program the board as well as to read and write data to the memory banks on the board.

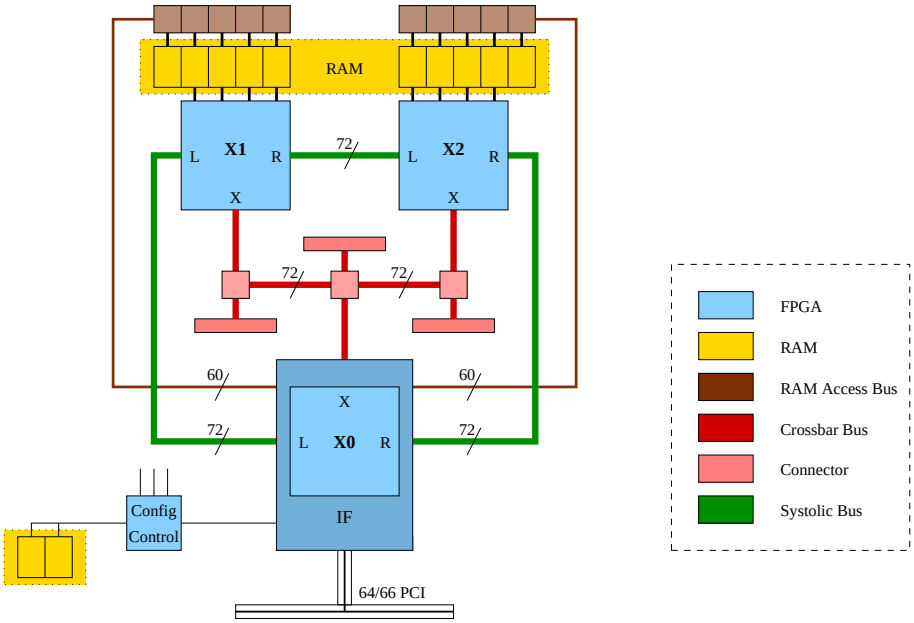

Figure 4.1: The SLAAC1V.

The board has twelve memory banks; four are connected to *X1*, four to *X2*, two to *X0* and two to the Configuration Controller. Any one of the memory banks connected to *X1* and *X2* can be used by *X0*, by swapping it with an *X0* memory. The capacity of each memory bank is one Megabyte, resulting in a total memory capacity of twelve Megabytes on the board.

The Configuration Controller programs the three Xilinx parts with configurations stored in the memory banks attached to it. The configuration bitstream for each Virtex1000 part is 766052 bytes in size. Therefore, each memory bank can store only one complete bitstream.

X1 and *X2* interconnect lines (72-bit wide) connected to the crossbar bus can be redirected to Input/Output (I/O) connecters on the board. External logic and devices are connected

to the SLAAC-1V board through these I/O connecters.

	X0	*X1*	*X2*
Before Authentication	Authentication Reconfiguration PCI bus-master		Token I/O Biometric I/O Client I/O
After Authentication	Authentication Reconfiguration PCI bus-master	Data Processing	Data Channeling Radio Modem I/O Token I/O Client I/O

Table 4.1: The functions embedded in X0, X1 and X2 before and after authentication.

The radio that has been developed is mapped onto all three FPGAs on the board (Table 4.1). The design is partitioned to allow some parts of the system to be reconfigured without affecting other parts. The *Reconfiguration subsystem*, the Authentication subsystem and the PCI bus-master are required for every stage in the operation of the radio. The modules that comprise these subsystems are loaded into *X0*. The *Reconfiguration subsystem* will re-program *X1* and *X2*, when required, to change the functionality of the radio. The entire *Authentication subsystem* is needed when a user is being authenticated. After authentication, only a portion of the Authentication subsystem is needed to check for the removal of the token. The entire subsystem, however, is retained to authenticate the next user. The biometric authentication module, the network connection to the client and the token are modules that are external to the reconfigurable logic. They communicate with the radio through the I/O port attached to *X2*. Before a user is authenticated, *X1* is not programmed with any logic and *X2* contains only the modules that interface the system with the external devices required for authentication, viz. the client connection, the connection to the token and the connection to the biometric module. After a user is authenticated, the data pro-

cessing elements of the *Data Processing subsystem* are programmed into *X1*. By replacing the configuration in *X1* the data processing algorithm of the radio can be changed. The rest of the *Data Processing subsystem*, which is primarily concerned with channeling the data, is loaded into *X2*, along with the I/O interfaces to the token, the client and the radio modem.

4.2 The Authentication Subsystem

This section deals with the implementation of the *Authentication subsystem* on the SLAAC-1V platform. This is outlined in Section 3.2. A detailed picture of the *Authentication subsystem* is shown in Figure 4.2. The TINI, the iButton, the FDA01 and the fingerprint scanner are devices that are external to the reconfigurable hardware.

When the radio is idle, the reconfigurable hardware in *X1* and *X2* is programmed for authentication. The user starts the authentication process by inserting the token. When the *Authentication Unit* is informed of the presence of the token, it begins the authentication run. The subsystem detects the presence of the token, authenticates the token, authenticates the user, retrieves the key table used by the *Reconfiguration* subsystem, starts up the radio and then waits for the removal of the token (Figure 3.5).

The token used in this system is a Dallas Semiconductior *iButton* [13]. The iButton is a smart card that has a cryptographic co-processor and runs Java. Java support for cryptography on the iButton is limited to a fast modular exponentiation operation. This was a key factor in deciding which encryption to use for authentication. The iButton is programmed with a key-table and the biometric signature of the user authorized to use that specific iButton. The *key-table* is used by the *Reconfiguration Unit* to determine which bit-stream to program into the hardware.

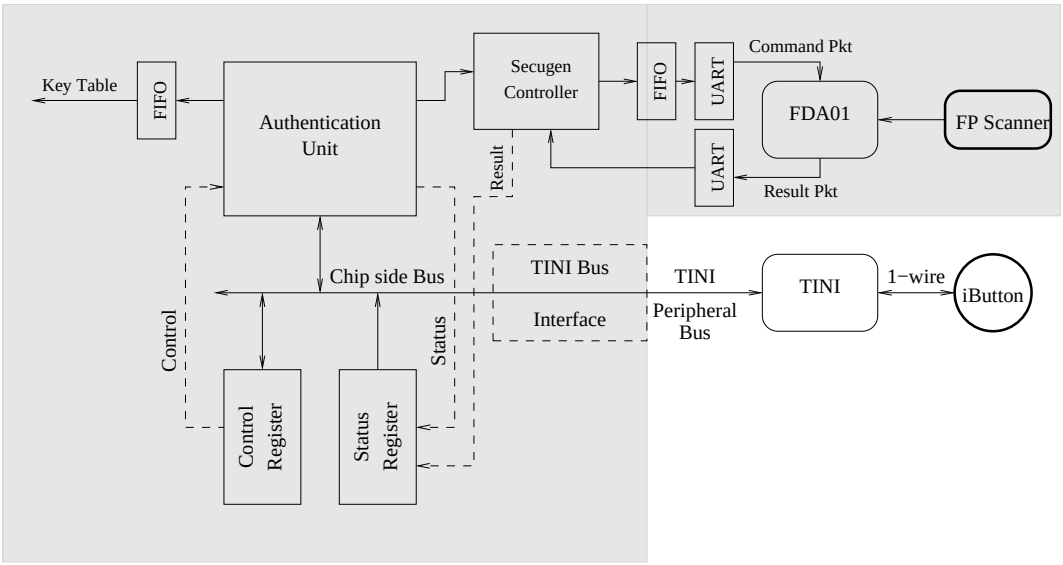

Figure 4.2: Implementation of the Authentication subsystem. (Modules outside the shaded region are not secure.)

The RSA algorithm [22] is used to secure all communication between the iButton and the reconfigurable hardware. An RSA key-pair is generated before the system is deployed [21, 22]. With the crypt analysis techniques available to a potential intruder today, public-key algorithms based on RSA require key sizes upwards of 1024 bits to ensure security. Although the algorithm used here is called the public key algorithm, neither key in the key-pair is actually made public in this system. Therefore, a key 86 bits in size is considered sufficient to maintain the level of security until 2020 (see Section 3.5). In this application, all communication between the reconfigurable hardware and the token is secured with a 512-bit key. The key size can be changed if the need arises, by upgrading the bit-stream used to program *X0*. The upgraded bit-stream will contain logic that uses a key of a different size. The method for upgrading the bit-stream is mentioned in Section 3.3 and the hardware implementation is described in Section 4.3.

The RSA key-pair comprises two keys, commonly called the public key and the private key. In this section one key will be called the 'public' key to distinguish it from the other, which is the 'private' key. The public key is used by the *Authentication Unit* and the private key by used by the iButton. Modular exponentiation is the basic operation in the RSA algorithm (Section 2.1.4). In RSA key-pairs used for the Public Key Cryptography Specification #1 (PKCS#1) [25], the exponent in the public key is usually 65537. The speed of the unit implemented in the FPGA varies linearly with the size of the exponent. Using the smaller exponent in the FPGA results in a faster cryptographic operation in the FPGA. The exponent in the private key is a very large number. The key with the larger exponent is assigned to the iButton since it has a dedicated co-processor for cryptography. This is done to speed up the execution of hardware implementation of RSA.

The iButton communicates with a host through a 1-wire network [13]. The *1-wire network* is a proprietary serial networking standard developed by Dallas Semiconductor that allows devices to receive power from the same lines that they use to communicate data. A network of 1-wire enabled devices can be set up on a 1-wire network. Each 1-wire enabled devices has a unique 1-wire network addresses. Using this address, a master device can communicate with a specific slave device on the net. The iButton is a 1-wire slave device. Communicating with the iButton requires a master that interfaces with the 1-wire network. Since it was not considered feasible to develop an interface to communicate with the iButton on the FPGA, an external device is used as a bridge to relay communications between the logic in *X2* and the iButton. The *Tiny Internet Interface (TINI)* [46] is a 1-wire master device that performs the task of communicating with the iButton and relaying the data to X2 and vice-versa. It is a Java computer on a single board, the size of a standard Single Inline Memory Module (SIMM). The TINI signals the reconfigurable logic in *X2* when an iButton either joins the 1-wire network or leaves the 1-wire network. The former event starts authentication, while the latter event triggers the radio to shut down if it is running.

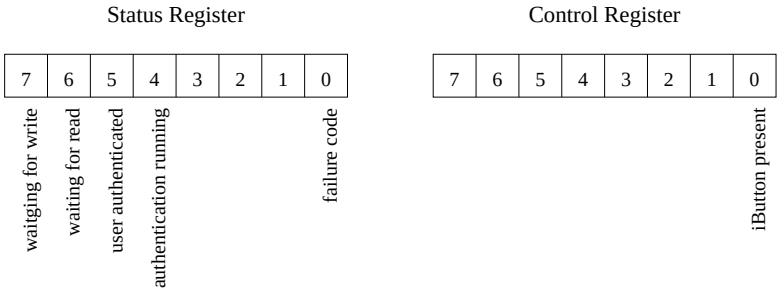

Figure 4.3: Relevant fields in the Status Register and the Control Register.

In addition to the externally connected modules, the *Authentication subsystem* comprises an *Authentication Unit*, a *TINI Bus Interface*, a *Status Register*, a *Control Register*, a *Secugen Controller* and two UARTs. The *Tini Bus Interface* and the two UARTs are implemented in *X2*. The remaining modules are implemented in *X0*. The TINI communicates with the *Authentication unit* through the *TINI Bus Interface*. The TINI Bus Interface makes each of the devices on the *Chip Side Bus* a peripheral device for the TINI. The TINI can read from the *Status Register* and read as well as write to the *Control Register* and the *Authentication Unit* using regular external bus I/O. Since each device behaves as a peripheral for the TINI, the design of the software that runs on the TINI is greatly simplified.

The *Authentication Unit* controls the authentication process. It generates a nonce, reads packets of data from the TINI, decrypts the packet, decides whether the packet is valid and then writes valid data out to either the *Secugen Controller* or the key-table FIFO. The TINI reads the *Status Register* to determine the state of the *Authentication Unit*, which reveals the progress made in authenticating the user and the result of running the authentication. A field in the *Control Register* is set by the TINI to indicate when an iButton is present. This field is checked by the radio to start the process of authenticating a user and to determine when to stop the radio. Figure 4.3 shows the format of both the *Control Register* and the

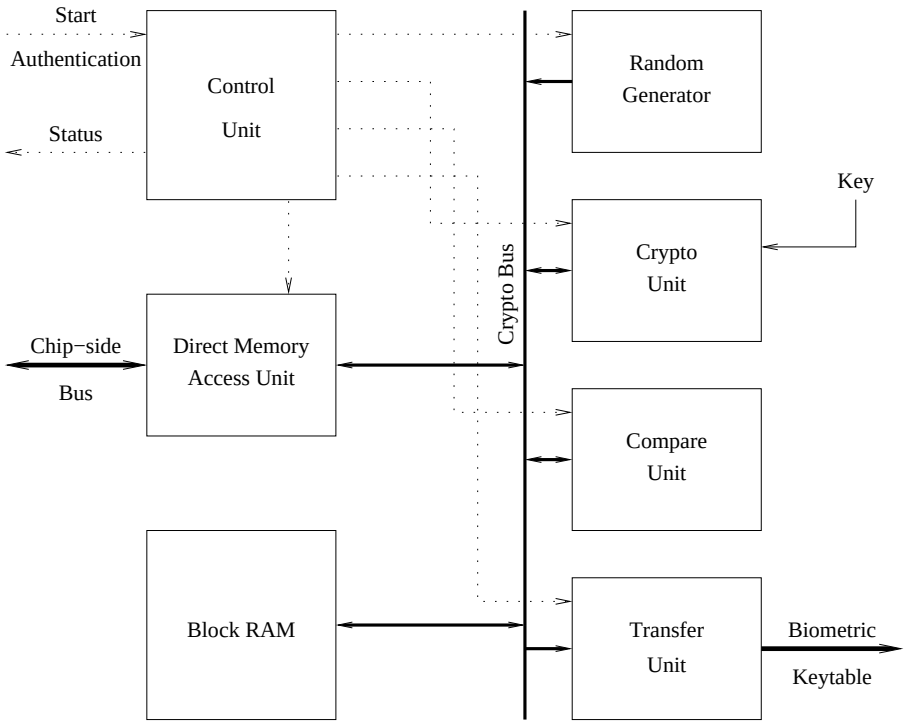

Figure 4.4: Details of the Authentication Unit.

Status Register.

The *Authentication Unit* is composed of several sub-components, each having a specific task. Figure 4.4 shows the sub-components of the *Authentication Unit*. Through the center of the figure runs a bus, which is used by the components to communicate data, address and control information. A fixed sequence of steps needs to be performed by the *Authentication Unit* to authenticate a user. Figure 4.5 shows a flow chart of these steps. Each node in the figure represents a task that must be performed. The paths are colour coded to distinguish between operations that comprise a sequence of tasks. Each node is labeled to show both the module that performs the task and a brief description of the task. The tasks along the path shown in red generate a challenge that is sent to the iButton. Each data packet that is received

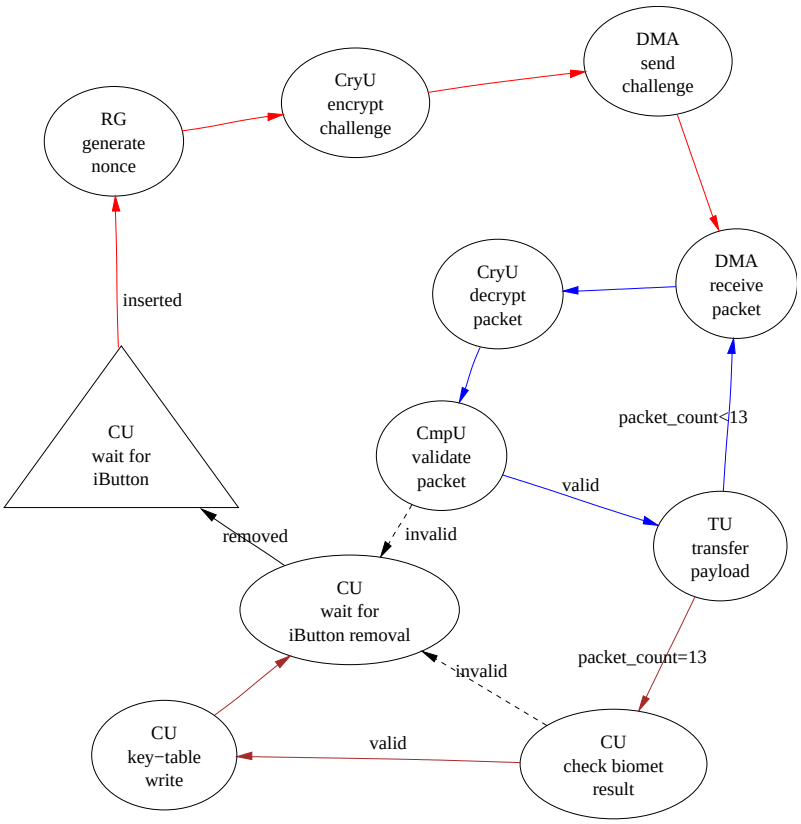

Figure 4.5: Flowchart of the key steps performed in the Authentication Unit for an authentication run. *(CU: Control Unit, RG: Random Generator, CryU: Crypto Unit, DMA: DMA Unit, CmpU: Compare Unit, TU: Transfer Unit)*

from the iButton is processed by the tasks shown in the blue path. Since multiple packets are received from the iButton, this path is traversed until the required number of packets are processed. The biometric and the key-table are assembled from the payload in these packets. Once the biometric and the key-table are assembled, the biometric is verified. The results of the verification are checked in the task shown in the brown path. The *Authentication Unit* then enters a state in which it waits until the iButton is removed. The authentication run will fail if either the packet cannot be validated or the biometric cannot be verified. The *Control Unit* switches between modules to carry out the necessary steps in the right

order. A field in the Control Register is set when an iButton is inserted. The authentication run will begin when this field is set. The result of the run is available through the *Status Register*. The result will be held in the register until the iButton is removed. If the iButton is removed before the authentication run is completed, the Control Unit will return to the state in which it waits for an iButton to be inserted.

The first step in authentication is the generation of a 64-bit nonce using a free-running maximal length LFSR. A 64-bit LFSR is implemented in the *Random Generator*. The size of the LFSR is chosen to ensure that the period of the sequence that it generates is extremely long. The LFSR is free-running and is clocked at 20MHz. Therefore, the sequence is guaranteed not to repeat for over 140 centuries (Table 3.2). The LFSR is not a perfert random generator. One of the biggest weaknesses of using an LFSR to generate a random number is that the sequence of random bits generated is repeated every time the system is reset. If the start time for the sampling of the sixty four bits of the nonce, can be made indeterminate, then this weakness of the LFSR will not pose a problem in the authentication unit.

The *Block RAM* is a scratch pad used to store packet data temporarily. It uses one unit of Block RAM in *X0*. The unit is partitioned into three regions. Figure 4.6 shows how the memory is mapped. Most modules in the *Authentication Unit* read data from one of the three regions, process the data and then store it back in the scratch pad. The functional modules that read and/or write to each region are listed in the figure. The direction of the arrow drawn against each entry in the figure shows whether the data are read from the region or written to the region.

At the heart of the *Authentication Unit* is the *Crypto Unit*. It is a hardware implementation of the RSA modular exponentiation operation. For each authentication run, this unit is invoked once to encrypt the data being sent to the iButton, and thirteen times to decrypt

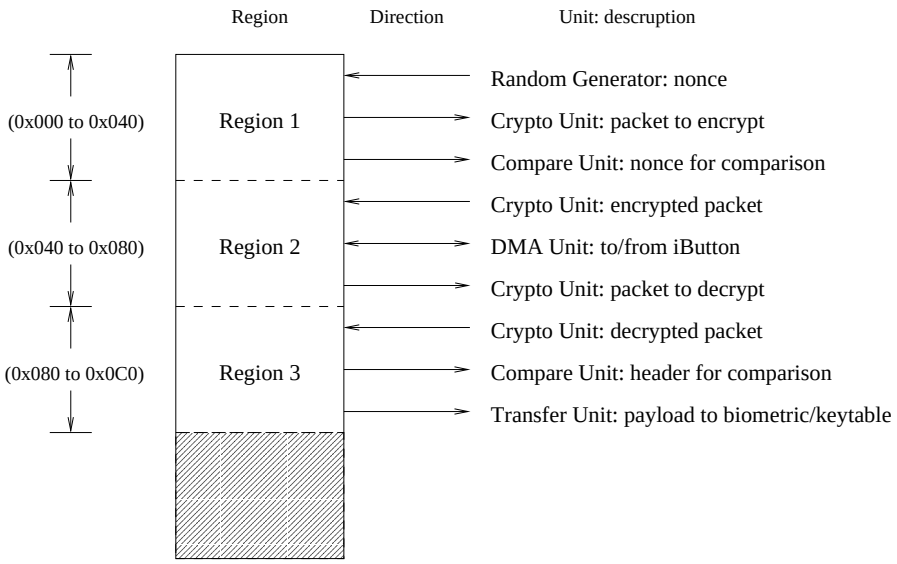

Figure 4.6: The memory map of the Block RAM used as a scratchpad for encryption/decryption.

data received from the iButton. The size of the key which is used is parameterized in the design. For this system, a 512-bit key is used. Data are read from the *BlockRAM Unit*, encrypted/decrypted and then written back to the appropriate region.

Data must be exchanged with the iButton for authentication. The TINI reads encrypted packets from the *Authentication Unit* and relays them to the iButton. All data that must be transferred are stored temporarily in the *BlockRAM Unit*. The address and control signals are generated by the *DMA Unit* to read data from the *BlockRAM Unit* and put the data onto the *Chip Side Bus*. The TINI reads the data through the *TINI Bus Interface*. Working in the other direction, data from the iButton that is destined for the FPGA is written to the *Chipside Bus* by the TINI. The *DMA Unit* generates the signals necessary to write the data into the *BlockRAM Unit*. The *DMA Unit* will always read from and write to Region 2 of the *BlockRAM Unit*. Data are never stored in clear text in this region. Therefore, even if

the TINI was to be controlled by a malicious intruder, unsecured data will not be read from the FPGA through this interface.

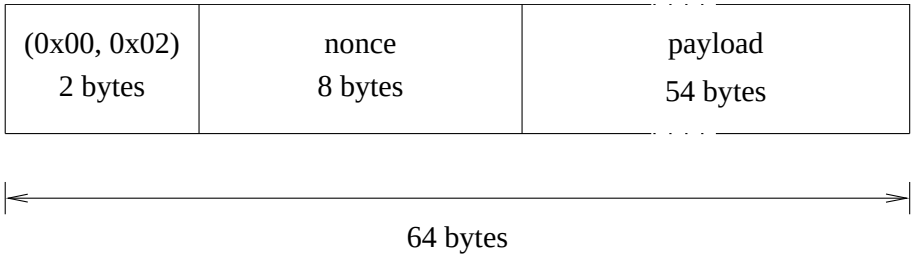

Figure 4.7: The structure of a packet of data transferred between the iButton and the Authentication Unit during authentication.

As a security measure, data that are received by the *Authentication Unit* must be validated before it is used. Two checks are carried out on the decrypted packet to ensure that the data received is valid. Both checks are made on the packet header. The format of the packet received from the iButton is shown in Figure 4.7. The header of the packet is a modification of the header used in the PKCS#1 authentication algorithm. The first check is a sanity check. The first two bytes of the packet are checked for the values $0x00$ and $0x02$ respectively. If either of these is not found, then it is assumed that the packet received was either corrupted in transfer or that the packet was generated by a malicious intruder. The latter is usually the case. Either way the packet is not processed any further and the *Control Unit* is notified of the failure. The second check compares the nonce in the received packet with the session-id created by the *Random Generator* for this authentication run. The second through the tenth bytes of the packet contain the nonce. The *Compare Unit* reads the session-id from region 1 in the *BlockRAM Unit* and compares it with the nonce in the packet. If the packet passes this test, it is considered to have arrived, unmodified, from a valid source (in this case the iButton). If either check fails, the *Control Unit* is notified of the failure and the authentication run is halted.

After each packet is validated, the payload of each packet is written to modules outside the *Authentication Unit* by the *Transfer Unit*. The biometric signature is written to the *Secugen Unit* which assembles the biometric signature into an FDA01 command for the biometric verification. A header that contains the command is attached to the signature and is written out to a FIFO. Until the entire signature is assembled the data in the FIFO are not released to the UART . It is imperative to delay the flow of data to the biometric module until after the entire signature is collected. If an invalid packet is received by the *Authentication Unit* before the biometric signature is fully assembled, the authentication run will halt. If the data received in the previous packets were not delayed, a portion of the command will have been written to the biometric verification module. The module will remain in an undefined state and will not be able to process the next command. The biometric verification module will need to be reset before it can be used again to authenticate another user.

The *Secugen FDA01* is the biometric module used in this system [16, 47]. It implements fingerprint verification and identification software on an embedded ARM processor. An optical fingerprint scanner is connected to the module. The biometric signature assembled by the *Authentication Unit* is embedded in the command required to run verification and then written to the FDA01 . The module scans the fingerprint of the user and compares it to the fingerprint signature sent by the *Authentication Unit*. The *Authentication Unit* is informed whether or not the user has been verified.

The key-table is assembled from the packets received by the *Authentication Unit* after the biometric signature has been processed. The key-table is written to a FIFO and is not made available to the *Reconfiguration Unit* until the biometric signature has been verified and the entire key-table is assembled.

Figure 4.8 illustrates the protocol used to transfer data between the *Authentication Unit* and the iButton during authentication. The first step in the authentication process is the

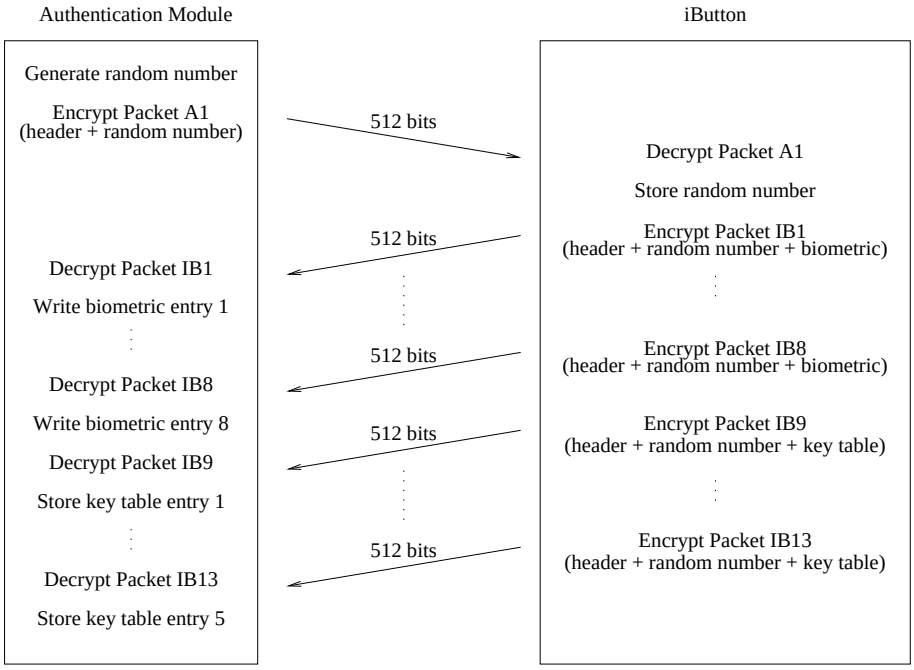

Figure 4.8: Data transfer protocol for authentication.

challenge made to the iButton by the *Authentication Unit*. The challenge contains the nonce that is to be used for the rest of the authentication run. If the challenge is validated by the iButton, it will attempt to transfer thirteen packets containing information required for authenticating the user. Data are transferred between the *Authentication Unit* and the iButton in packets. The size of the packet is limited to the size of the encryption/decryption key used in the *Crypto Unit*. The size of a packet of data in this system is limited to 512 bits, since the encryption key is 512 bits long. Both the biometric signature and the key-table are larger than 512 bits. Therefore, they will straddle multiple packets. The key-table, which is 256 bytes long straddles five packets. The biometric signature, which is 400 bytes long, is transferred in eight consecutive packets. The structure of the packet is shown in Figure 4.7. The first eight packets contain the biometric signature of the authorized user. The last five packets in the set contain key-table entries. The Authentication unit validates

each packet when it arrives. If all the packets are valid, the data in the first eight packet are assembled into a biometric signature. After receiving all the packets that contain the biometric signature, the *Authentication Unit* assembles the key-table for the user from data that arrive in the next five packets.

Software on the TINI, written in Java, checks for the presence of the iButton on the 1-wire network. When an iButton is detected, the field in the *Control Register* is updated to convey this information to the *Authentication Unit*, which, then, initiates an authentication run. The challenge generate by the *Authentication Unit* is relayed to the iButton. The response packets generated by the iButton, which contain the key table and the biometric signature are relayed back to the Authentication Unit. After all the response packets have been relayed, the TINI lies idle until the iButton is removed. The removal of the iButton is conveyed to the radio through the *Control Register*. The TINI goes back to checking for the presence of an iButton.

A user is deemed authentic if all packets received from the iButton are valid and the biometric signature matched the user's fingerprint. When the user is verified, the key-table is written to the *Reconfiguration Unit*, which retrieves the necessary configuration bit-streams and re-programs *X1* and *X2*. This will set up the data channels on which the user is authorized to receive data.

The next section describes the *Data Processing* subsystem.

4.3 The Data Processing Subsystem

The radio that has been implemented contains the hardware to support three unique methods for the encryption and the decryption of data flowing through it. Data packets received

by the radio are piped into one of four channels. Each channel is allowed to have a unique encryption/decryption method. One channel is reserved for unencrypted data. It is assumed that data transferred over this channel has been pre-encrypted by the device system that transmitted it. This channel is used to transmit updated bit-streams from a central broadcasting unit to each radio in the field. An updated bit-stream is used to change the keys needed for authentication, add features/processing elements to the radio, update features of the radio, change the cryptographic engine or change the keys used to encrypt and decrypt data in each channel. Section 3.3 contains a system-level description of the *Data Processing subsystem*. This section describes the implementation. On the SLAAC-1V, this subsystem is implemented in *X1* and *X2*.

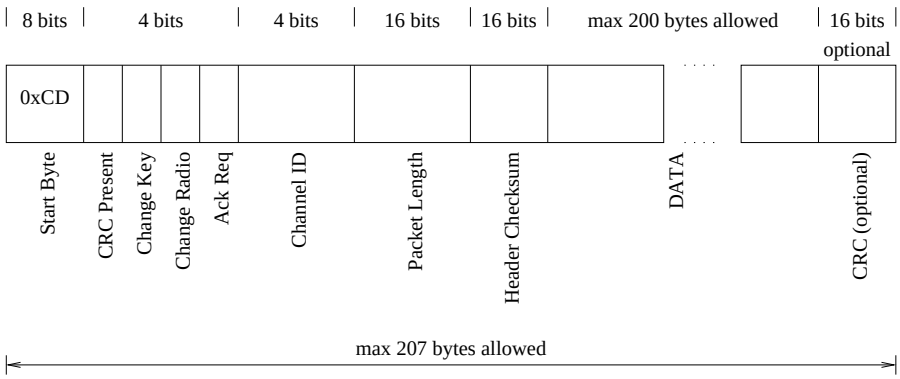

Figure 4.9: The radio data packet. All values that span more than one byte are stored as Little Endian.

Data flows through the radio in packets. A *header* is attached to each packet that is sent over the air by the radio. The details of a packet including the header are shown in Figure 4.9. The header contains a start byte, control information, the channel assigned to the packet, the length of the data encapsulated and a checksum for the header. A Cyclic Redundancy Check (CRC) is used to validate the payload of the packet. The standard 2-byte CRC-CCITT16 [43] is used. The result of the CRC is appended to the packet. The header provides the channel

routing information and supports methods that test the packet for integrity. Some fields in the header are reserved to convey control information for future versions of the system. The header is used to determine the contents of the packet. Other than user data, the packet may contain new encryption/decryption keys or an updated bit-stream to reconfigure the radio dynamically. An Internet Protocol (IP) checksum [48] is calculated for the packet header. It is placed in the packet immediately after the header. The checksum is used to determine whether or not the header for the packet is valid. The payload is allowed to contain a 65536 bytes of data, but is constrained to 200 bytes for this application, to prevent overflowing the FIFOs in the radio. This is a limitation of the system, but it can be circumvented by modifying the data transfer protocol used to transfer data out of the FPGA.

The *Data Processing subsystem* can be broadly divided into three parts. The first, shown in Figure 4.10, is the interface between the client and the reconfigurable logic. This portion of the subsystem is implemented entirely in *X2*. The radio is connected to the client through a 10Mbps Ethernet link provided by the *Z-World RabbitCore RCM2100* (Rabbit) [49]. The Rabbit is a C-programmable processing unit with a TCP/IP stack, an Ethernet connector and parallel I/O lines. It is built around the *Rabbit 2000 Microprocessor*, which has an 8051 core, manufactured by Rabbit Semiconductor [50].

The radio feeds data to the Rabbit through the *Rabbit Interface*. The interface multiplexes data from four channels onto the single link with the Rabbit. The Rabbit encapsulates this data in a UDP packet and sends it to the client. The access privileges of the user class may not allow the client to receive data from all the channels. The disabling of certain channels in the radio is supposed to be transparent to the client. Therefore, neither the radio nor the Rabbit will attach any information to the UDP packet to indicate through which channel the data was received.

The *Rabbit Interface* has one input for each channel of data that it is expected to relay to

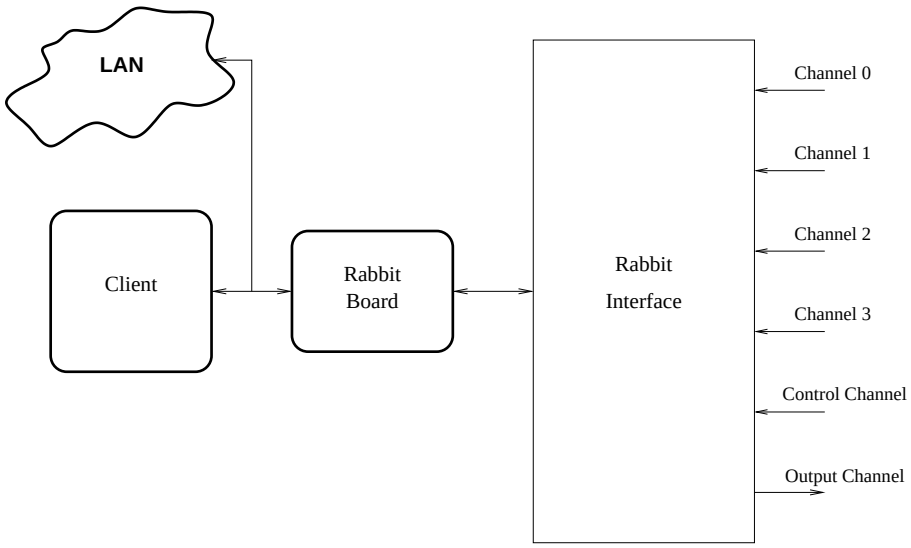

Figure 4.10: Block diagram of Data Processing subsystem (1 of 3) showing the interface between the radio and the client.

the Rabbit. A *Control Channel* provides the control information needed to multiplex the link with the Rabbit. Since data packets will arrive back-to-back in the data channels, the interface must be told how many bytes must be read from a channel to assemble a packet. Further, sometimes the data that is received is corrupted while it is transmitted over the air. Wireless networks have a higher error rate than wire line networks. These packets are discarded by the Rabbit. The length of the packet, the channel number and the result of a CRC check on the packet is the control information that is retrieved from the *Control Channel*.

The Rabbit receives data from the client on one of four consecutively numbered User Datagram Protocol (UDP) ports. Each port corresponds to a channel of data. The data received is padded with control information(channel id, whether acknowledgements are desired and the length of the packet) and sent to *X2* via *Rabbit Interface*. The control information is used to create the packet header that is attached to the packet before it is transmitted on

the air.

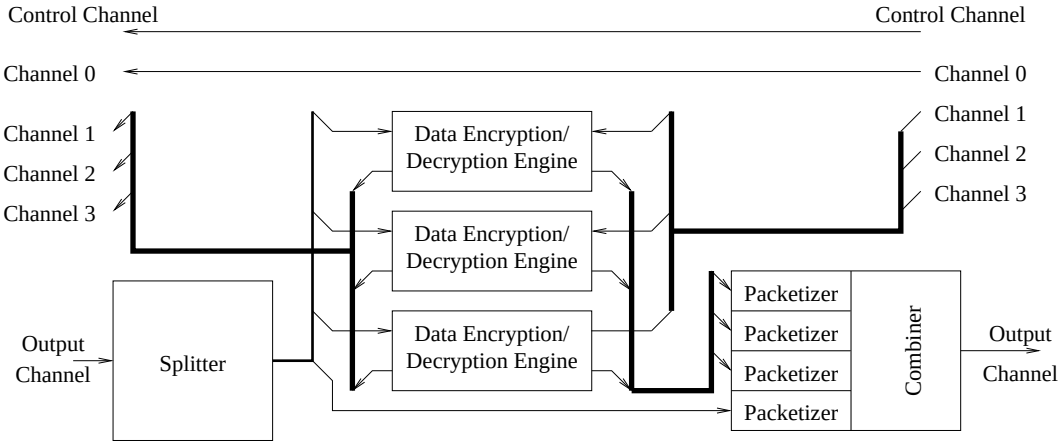

Figure 4.11: Block diagram of Data Processing subsystem (2 of 3) showing the encryption/decryption units on each channel.

All data that is sent to the radio by the client is encrypted before it is transmitted. Figure 4.11 shows the modules in the subsystem that encrypt and decrypt the data. There are three *Encryption/Decryption Engines*. Each engine processes one channel of data.

The Rabbit attaches a header to every packet that is received from the client. This header is a subset of the header shown in Figure 4.9. One field in the header determines the channel that is supposed to process the data. Since the packets are received back-to-back in a single stream, each packet must be assembled, and sent into the encryption unit for that channel. The packet is stripped of the header attached by the Rabbit before it is encrypted. Relevant portions of the header are saved to construct the header that is attached to the packet after the packet is encrypted. The *Splitter* assembles a packet, determines the channel associated with the packet and feeds the packet into the appropriate *Encryption/Decryption Engine*.

The encrypted data from each channel at the output side of the *Encryption/Decryption Engine* is collected by the *Packetizer*. The *Packetizer* prepares the packets for transmission

over the air by attaching a header to each packet as shown in Figure 4.9. The header contains information about which channel was used to process the packet. The output channel to the radio modem is multiplexed with packetized data from all four channel by the *Combiner*.

The fourth channel shown in the figure (Channel 0) is not encrypted/decrypted. This channel is used to carry information that is pre-encrypted, like a new configuration. Data sent through this channel should be accepted by the receiver regardless of how the other channels are processed by the configuration that is running when the data are received. Data transmitted in this channel will still need to be split into packets that adhere to the format shown in Figure 4.9.

Data flowing through the system in the other direction, from the radio modem to the client is classified prior to it flowing into the Encryption/Decryption Engines. The process of classification involves separating the single stream of modem traffic into the channels on which they were sent. Each channel is decrypted by the engine for that channel and the data flows out to the client through the Rabbit Interface. The *Control Channel* contains packet header information that is used by the Rabbit to assemble the packet before it is sent to the client.

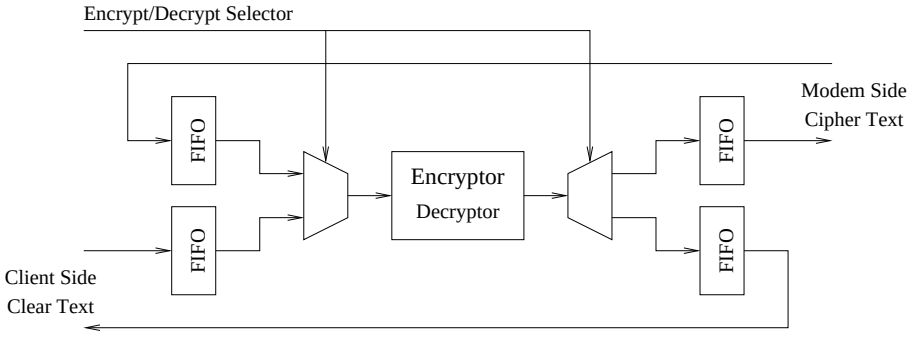

Figure 4.12: Block diagram of an Encryption/Decryption Unit.

An XOR encryption scheme is used on all encrypted channels in this prototype. This scheme

is symmetric in its operation. It is implemented in the *Encryption/Decryption Engine*. If the direction of the data being fed into the engine is reversed, then, with a few changes in the settings of the engine, the operation performed on the data is inverted. The *Encryption/Decryption Engine* is built using combinational logic that performs an XOR operation on the data. Figure 4.12 shows a detailed view of the internals of the *Encryption/Decryption Engine*. XOR encryption is used in this system to demonstrate the capabilities of the radio. A single *encryptor/decryptor* is used to encrypt data before they are transmitted and to decrypt data that are received by the radio. The signal *Encrypt/Decrypt Selector* switches the engine to perform either an encryption or a decryption depending whether data is being transmitted or received on that channel. FIFOs are used to buffer data so that received data are not lost while data to be transmitted is being processed and vice-versa. The *Encryption/Decryption Engine* for all the channels are implemented in *X1*. The scheme used to secure the data can be changed by loading a new configuration into *X1*.

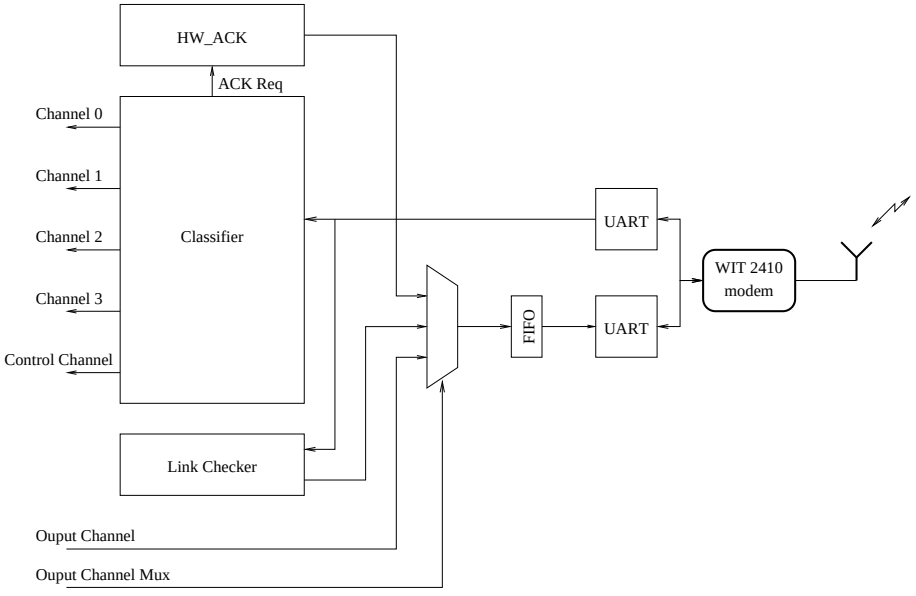

Figure 4.13: Block diagram of Data Processing subsystem (3 of 3) showing the classifier, link checker and the radio modem.

Figure 4.13 shows the details of the interface with the radio modem. The *Cirronet WIT2410 Radio Modem* is used by the system to transmit and receive data over the airwaves [51]. The WIT2410 is a 2.4 GHz spread spectrum frequency hopping wireless transceiver. The configurable logic in the radio communicates with the WIT2410 over a serial link. The WIT2410 can be set to run as a base or a remote. To be able to send data to another radio or receive data from another radio, a connection must be established between the two radios. A base unit is required to establish a connection. Two remote units will not be able to communicate directly with each other. Therefore, in any network that is set up using the WIT2410, there must be one base. The WIT2410 can be set to return the header used in the wireless transmission along with the payload. The header is needed if the application wants to catch control packets that are sent between two units to indicate when a connection has been made (or broken). In this application, the control information is not required by the radio to determine when a connection has been established.

On startup the radio is set to transmit a data packet with a request for an acknowledgement. When an acknowledgement is received from the other end of the link, a connection is known to have been established. If an acknowledgement is not received within a set period of time, the WIT2410 is turned off and then turned back on again. In a noisy environment these two steps may have to be repeated until a connection is established. The *Link Checker* performs this task of establishing a link before data are sent over the air.

The WIT2410 is connected to the radio through an asynchronous serial link. Packets captured by the WIT2410 are sent to a *Universal Asynchronous Receiver-Transmitter (UART)* in the radio. Data received by the radio through the UART must be processed to check the integrity of the data and to extract channel routing information from the header. In the radio, the *Classifier* performs these tasks. In addition, the *Classifier* detects whether a hardware acknowledge has been requested. If an acknowledgement has been requested, the *ACK_REQ* line is asserted and the *HW_ACK* sends the acknowledgement to the transmit-

ting radio. After the header is processed, the classifier writes the packet to a FIFO in the correct channel.

The *Classifier* is responsible for validating the packet and detecting if an acknowledgement is requested. A field in the header of the packet (Figure 4.9) is set by the transmitter to request acknowledgement of the packet. If the transmitter does not receive a requested acknowledgement within a preset time, it will re-transmit the packet. Using acknowledgements reduces the data throughput since the transmitting radio lies idle waiting for the acknowledgement to arrive. The receiver is able to generate an acknowledgement faster in hardware than in software. Therefore, the drop in throughput is less significant if a hardware acknowledge is used. The *Classifier* validates the header by calculating a checksum. If the header is valid it begins to receive packet data and forward it to downstream components. Once the entire packet is received, the *Classifier* calculates and verifies the CRC for the packet. If the check returns negative, hardware downstream from the classifier is instructed to drop the packet. A notification of the lost packet may be sent to the client. If the transmitting radio is set up to broadcast data to multiple receivers, the receivers should not be required to acknowledge the reception of packets, since the transmitting radio will have to wait till it receives every acknowledgement before it can continue. Further, receivers that have already acknowledged the receipt of the packet should discard the a retransmitted packet.

The data path in the radio is potentially bi-directional. The stream of data that are sent to the radio modem for transmission can originate in either the *Packetizer* or the *HW_ACK*. Only one of the two sources can be used at a time. Therefore, bi-directional communication of data is allowed only if hardware acknowledgements are turned off. Data on the multiplexed output stream is sent to the output UART through a FIFO. A FIFO is used here to prevent data loss since the speed at which the UART sends data is much slower than the speed at which the Rabbit can relay data from the client into the system. The Rabbit is notified by the *Rabbit Interface* when the FIFOs in the output data path have filled up. The software

on the Rabbit is designed to buffer packets of data as they are received from the client. The size of the buffer can be set at compile time. The larger the buffer, the less chance there is of losing bursts of high volume data that arrive at the Rabbit in a short time.

The *Data Processing Subsystem* is physically present in the system only after a valid user has been authenticated. As soon as the user removes his token, the components of this subsystem are removed from the reconfigurable hardware. The subsystem that facilitates the setting up and the removal of the components of the *Data Processing Subsystem* is the *Reconfiguration subsystem*, which is discussed in the following section.

4.4 The Reconfiguration Subsystem

The *Reconfiguration subsystem* is essential for the implementation of a secure radio (Section 3.4). After a user has been authenticated by the *Authentication subsystem*, the *Data Processing Subsystem* is set up in *X1* and *X2*. This involves programming these two FPGAs with the required configurations. This section describes how the configurations are stored, retrieved and programmed into the FPGAs. Figure 4.14 shows the details of the *Reconfiguration subsystem*.

At the heart of the *Reconfiguration subsystem* is the *Key Table*. The *Key Table* is a unit of blockRAM on *X0* which contains information that is used to decide which configurations to load into *X1* and *X2* and to maintain a record of what is currently running on both these FPGAs. The format of an entry in the Key Table is shown in Figure 4.15. Each entry in the Key Table is 128 bits wide. The *tag* is used to identify the configuration. Control information is stored in each entry to indicated whether the entry is *valid*, whether it is a *partial* bitstream, whether it is available locally and whether the entry points to the *default* configuration that must be loaded at startup. The location of the configuration in memory

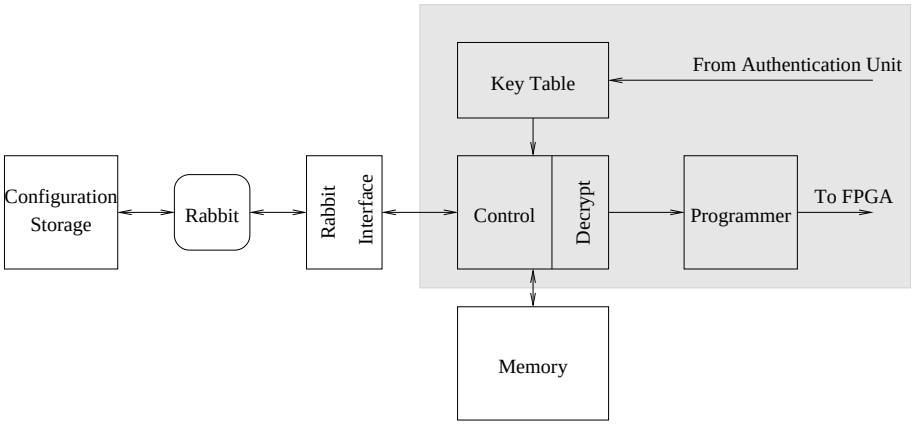

Figure 4.14: Block diagram of Reconfiguration subsystem. Everything outside the shaded area is considered insecure.

is indicated by the *start* address and the *stop* address entries. Every configuration is stored encrypted. The table entry contains the *key* that should be used to decrypt the configuration. The *Key Table* is loaded into the system after a user is authenticated. It is retrieved from the token used to authenticate the user. The entries in the *Key Table* determine the access rights of the user.

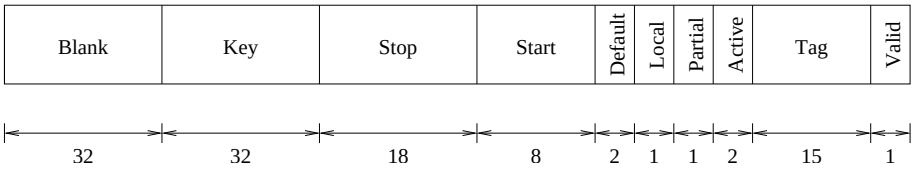

Figure 4.15: A Key Table entry.

Configuration bitstreams are indexed by a tag. This tag must be unique for every configuration available to users of the radio. A 31-bit tag allows approximately 2 billion configurations, each with a unique tag. Since it is not possible to store every possible configuration in hardware, a scheme has been developed that allows the *Control Unit* to request encrypted configurations from an external server. The *Configuration Storage* is a server that stores

configurations in a database. The database of configurations is indexed with the unique tag. When the *Configuration Store* receives a request for a configuration, it checks the database to see if the configuration is available. If the requested configuration is available, it is sent to the *Control Unit* via the Rabbit. The Rabbit is the link between the *Control Unit* and the *Configuration Store*.

The *Control Unit* comprises the logic required to parse the *Key Table*, generate requests to load the required configurations, decrypt the configurations and program the FPGAs on the board with the appropriate decrypted configurations. The Blowfish [21] encryption scheme is used to protect the configurations. When a user has been authenticated and *Key Table* has been loaded by the *Authentication Unit*, the *Control Unit* is signalled that the *Key Table* has been changed. The *Control Unit* parses the *Key Table* and generates requests to the *Configuration Store* for configurations that are not in memory. Configuration data that is returned by the *Configuration Store* is written to the memory location indicated in the *Key Table* entry for that configuration. Once all the configurations are loaded into local memory, the *Control Unit* reprograms *X1* and *X2* to set up the *Data Processing Subsystem* in the radio. A clear-text bitstream is required to reprogram either FPGA. The encrypted bitstream is read from memory, decrypted in a Blowfish engine embedded in the *Control Unit*, and written to the *Programmer*. At any time during the running of the radio, the Control Unit can be signalled to swap in a new configuration using one of the entries in the *Key Table*.

4.5 Summary

This chapter described the implementation of the secure radio on the SLAAC1-V reconfigurable platform. The entire system is divided into three subsystems: the *Authentication*

subsystem, the *Data Processing Subsystem* and the *Reconfiguration subsystem*. Each subsystem comprises modules which are orchestrated to perform the necessary functions of that subsystem. This chapter also described and justified the key design decisions that were made during the implementation of the radio. In the next chapter, the authentication scheme that has been impemented in reconfigurable hardware is analyzed for its strengths and weaknesses.

Chapter 5

Analysis and Results – Authentication

The security of the proposed radio hinges on effective user authentication. The market for user authentication using biometrics is slowly maturing as an increasing number of applications are found that could use biometrics. The market for high-end ultra-security devices is fueled by the need for securing access to the resources of large corporations and even the government. These devices are tailored to meet the specifications of the customer. Therefore, the specifications and the capabilities of these devices are generally kept under wraps and are not available to the public.

Traditionally low cost authentication was synonymous with either smart card authentication or passwords authentication. Since the cost of deploying biometric technology is falling by the day, the market for low-end biometric devices is maturing. This market is taking the biometric concept to the casual home/office user. These low-end devices are not as secure as the high-end custom systems mentioned earlier.

Standardization results in greater acceptability among consumers and hence greater sales volumes for the manufacturers. The tendency for manufacturers in an industry to join

hands in developing a common standard makes sound economic sense. A common standard for an interface between the device and the software gives the users the ability to match their software with any conforming hardware. This allows users to select from a wide range of devices depending on their requirements. Manufacturers can be sure that existing applications will support their devices if they conform to the standard. The JavaCard [52] Application Programming Interface (API) standardized the market for smart cards that run Java applets. The Java Cryptographic Extensions (JCE) [53] and the Java Authentication and Authorization Service (JAAS) [54] are efforts to standardize the authentication and cryptographic products available to programs written in Java. Two groups of companies that manufacture biometric authentication devices are trying to steer the industry into accepting common APIs. The BioAPI [55], proposed by the BioAPI Consortium is an open standard. The BAPI [56], developed by I/O Software, is now the property of Microsoft, which is pushing to make this the global standard.

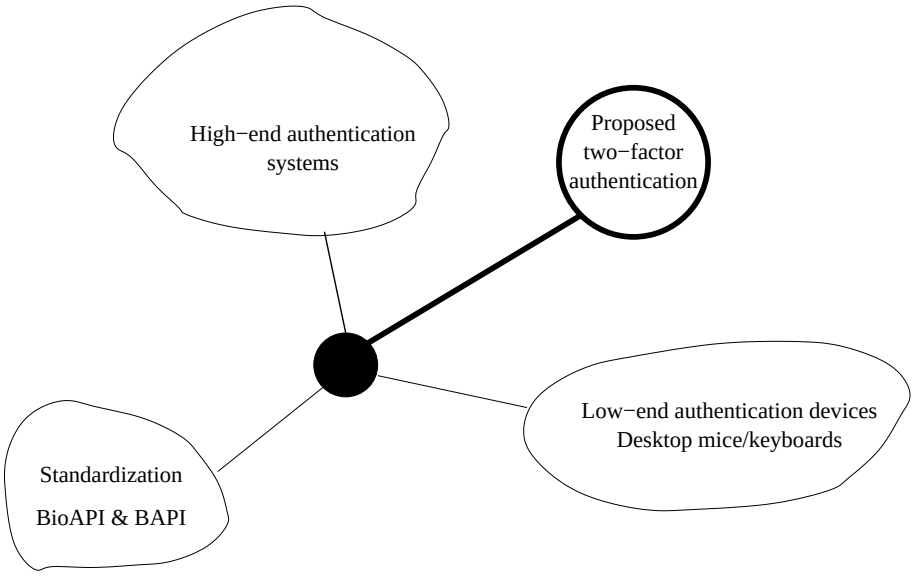

Figure 5.1: The dominant forces that are driving the user authentication industry.

Two-factor user authentication is used in the proposed scheme. In this chapter the proposed

authentication scheme is analyzed. The sections that follow explain this scheme, and then relate it to the dominant forces that are driving the industry, viz. high-end authentication schemes, desktop authentication devices and the efforts to standardize the use of biometrics in the industry (see Figure 5.1).

5.1 The Proposed Scheme

The performance of an authentication system is broadly divided into the security of the user devices used and the security of communications protocols used. In the proposed biometric authentication system (see Figure 5.2) the devices used are the iButton, the FDA01, the fingerprint scanner, the Rabbit and the FPGA. All communication between these devices need to be secured by using protocols that are secure. The privileges of the user are retrieved when the user is authenticated. The privileges decide the applications of the radio that are available to the user. Further, the details of the implementation decide the range of applications and the number of users that the radio will support. In this section the proposed authentication system is analyzed by examining the security of the devices used and the security in the protocols used by the implementation.

Section 3.2 and Section 4.2 describe the proposed authentication system in detail. Section 3.5 delves into the security of some of the components used in the system. This chapter will delve into the security of each component and protocol that is used in the implementation of the authentication system on the radio. Figure 5.2 is a block diagram of the entire system that authenticates the user and then retrieves and applies the user's privileges. The *Configuration Manager* and the *Authentication Unit*, which are in the FPGA, are shaded in blue. The iButton, which is the token used, is shaded in olive green. The data path from the *Configuration Manager* to the *Configuration Store* through the Rabbit is shaded in purple.

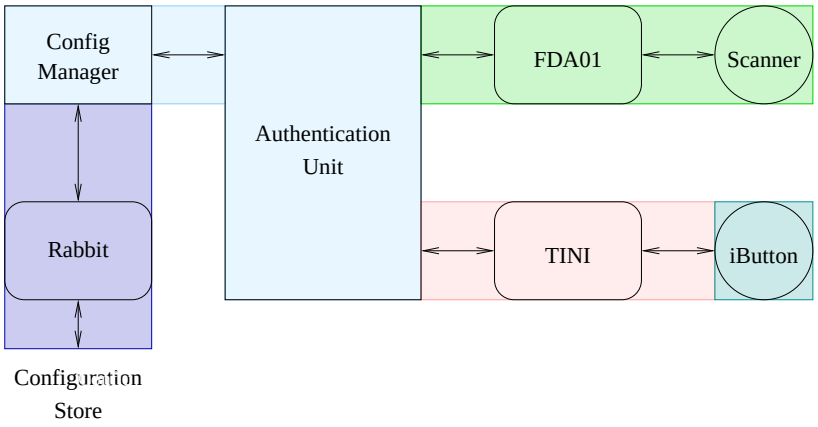

Figure 5.2: A block diagram of the components and data flows required to authenticate a user and retrieve the user's privileges. *The secure components/paths are: FPGA (Blue), iButton (Olive Green). The components/paths that need to be secured are: TINI(Red), Rabbit(Purple), Biometric(Green).*

The FDA01, the fingerprint sensor the data paths that are used during the verification of the biometric are shaded in green. The data path between the iButton and the FPGA through the TINI are shaded in red. Each of the above are described in detail in the following sections.

5.1.1 The Discovery Protocol

The first stage in an authentication run is the discovery of the presence of a user. The system knows that a user is present when an iButton is inserted. Data is transferred between the *Authentication Unit* and the iButton, via the TINI. The data exchanged contains the biometric signature and the key-table. The key-table contains the access privileges of the user. The iButton and the *Authentication Unit* are secure. The link between them, via the

TINI, is not. Data that is being transferred between the iButton and the *Authentication Unit* is secured using encryption. Each of the following subsections describes the security of one of the following: the iButton, the Authentication Unit, the TINI and the protocol used to secure communications.

The iButton

The iButton [13] has been designed with features to protect it from intrusion. The SRAM within the iButton is designed to rapidly erase its contents if an attacker tries to penetrate the stainless steel casing of the iButton. This process, known as zeroization of the SRAM is carried out in less time than that it would take the attacker to pry open the casing and then read the contents of the SRAM. The keys that are stored in the button will never fall into the wrong hands. Memory zeroization will also be triggered if the chip is micro-probed or subject to extremes of temperature.

Excessive voltage levels are sometimes applied to smart card terminals in an attempt to erase the master key of the device and thereby break the perimeter defense. A fuse within the iButton is designed to blow, permanently, if an excessively high voltage is detected on the 1-wire. The iButton is a 1-wire device. The link between the host and the iButton is physically just one wire. If the fuse blows, the only path of communication with the iButton is broken.

The iButton has a true time clock, a tamper-proof real-time clock that is set when the device is manufactured. Once set, the clock can never be reset and is forever increasing. The iButton is able to use this clock to stamp transactions. Stamping is used to prevent replay attacks on a system that uses the iButton as a token. The clock can also be used to generate a timeout to protect it from waiting indefinitely for a transaction to complete.

The software on the iButton is written in Java. A Java applet is loaded into the button before it is assigned to a user. The applet exports function calls, using the Javacard interface [9]. These function calls are available to other one-wire devices connected to the iButton. The applet, itself can never be read back from the iButton. Hence once it is written to the iButton, the applet is not accessible for reverse engineering. The programming interface of the iButton is protected with a master Personal Identification Number (PIN). Once programmed, the iButton cannot be reprogrammed to perform a different task unless the PIN is available. The Java byte-code on the iButton is, therefore, protected from unauthorized modification.

The iButton has a 1024-bit cryptographic coprocessor which speeds the task of computing the RSA modular exponentiation. The key used by the iButton to encrypt any data that goes out from it, is written into the iButton using functions protected by the PIN. In the implementation, there are no function calls that allow the key to be read back from the iButton. Once set in the iButton, the key is protected from read-back and from unauthorized modification.

The Java cryptographic iButton is secured from intrusions that are aimed at recovering sensitive data stored within it. A version of the iButton has been certified to be conforming to FIPS 140-1 [57] by the National Institute of Standards. This certifies the iButton to be a trusted and physically secure module, to protect sensitive, unclassified information used by the government.

The TINI

The Tiny Internet Interface (TINI) [46] is placed in the discovery path between the iButton and the *Authentication Unit* on the FPGA. It has a one-wire interface to communicate with the iButton. The authentication modules on the FPGA appear to the TINI as external bus peripherals (see Figure 4.2). The TINI is able to read and write to a *Control Register*, a

Status Register and a *DMA Unit*. The TINI works like a link between the FPGA and the iButton, relaying information from one to the other.

The TINI checks for the presence of an iButton. When the iButton is detected, it checks that the iButton inserted is a Java iButton and then signals the *Authentication Unit* that an iButton is available for authentication by writing to the *Control Register* (see Figure 4.3). When the iButton is removed, the TINI writes to the same register to signal the end of the session.

The TINI is, essentially, used to relay encrypted packets of data between the iButton and the FPGA. It does not have the encryption/decryption capabilities that the iButton has. Further, the keys used for encryption and decryption are not available to the TINI. So decrypting packets of data as they flow through the TINI is not a trivial task.

Since only the TINI can read the iButton, the presence of the iButton can only be checked by the TINI. A denial of service attack on the system is possible, if the software on the TINI were changed or if the TINI were replaced by an attacker. The TINI is required to inform the *Authentication Unit* as soon as the iButton is removed from the system. If an attacker gets control of the TINI while the radio is in use, he could prevent the TINI from informing the *Authentication Unit* of the removal of the iButton. This would result in the radio being in use even after the user has left.

The TINI is used to relay information between the iButton and the FPGA. If the TINI were replaced with a malicious unit, the system could be compromised. The radio cannot be started up without a valid iButton. Once the radio is up, the TINI could prevent the radio from shutting down, even after the user has left. This can be solved by implementing a one-wire interface within the FPGA. The details of the one-wire protocol for the Java iButton are not available, so the using the TINI is the only way to read data from the iButton. An Application Programming Interface (API) for the iButton, using a Dallas Semiconductor

chip DS2480B (an RS232-to-one-wire convertor) [58], will be available soon. When it is released, a state machine to control the iButton without the TINI can be written in the FPGA.

The Authentication Unit

The *Authentication Unit* on the FPGA is designed to be secure. It is comprised of modules for each of the tasks that is has to perform during authentication (see Figure 4.4).

The *Authentication Unit* performs a 512-bit RSA modular exponentiation operation to encrypt/decrypt data in packets of 512 bits. A 512-bit key is a compromise between the size/speed of the module and the maximum data that can be transferred in to the module in each packet. The larger the key, the more secure the communication (see Table 3.1). The RSA encryption/decryption operation is slower than the operations performed by all symmetric ciphers. Since authentication is a one-time process involving fourteen RSA operations, this is not critical to the performance of the system. In fact, it makes the system more secure since any attack on the system, which tries to feed random/chosen packets into this unit to see how it responds, will take longer for each packet used. This reduces the number of packets that can be tried by the attacker in a give time, which, in turn, increases the level of security of the system (see Equation 3.1).

Figure 4.6 shows the structure of the memory used as a scratch pad for all encryption/decryption operations. Region 2 in the memory map, is read from and written to whenever data is transferred into and out of the *Authentication Unit*. Data in this area of memory is always encrypted, so an inadvertent leak of clear-text information from the *Authentication Unit* is not possible.

The *Authentication Unit* generates a one-time session-id, which is used to flag all the commu-

nication for an authentication run. This number is pseudo-random number generated using a 64-bit free running maximal length LFSR. Since the LFSR is free running and clocked at 24MHz, the period of the LFSR is over a hundred centuries. As long as the radio is kept running, without a reset, from the time it was turned on, this pseudo-random number will not repeat in the lifetime of the system. The assumption that the radio will not be turned off or reset is not valid. The power supply and the reset signal to the radio will be controlled by whomever is in possession of the radio. Since the output sequence generated by the LFSR is repeated every time it is reset, the pseudo-random session-id can be determined at any point in time after a reset. If the taps used in the implementation of the LFSR are kept confidential, the pseudo-random session-id itself will not be available. Only the encrypted value containing the session-id, which is sent as a challenge, can be recorded. A direct implication of this is that as long as the radio can be reset and the signals on the input bus can be controlled to an accuracy of 4.16667ns, a replay attack is possible to gain access to the system. The values generated by the LFSR are only as random as the seed used to start the LFSR. A random seed can be generated using a white noise source. This is essentially an analog/hybrid circuit which is not available on the FPGA. A level of randomness can be derived from the Delay Locked Loops (DLL) used to synchronize the clocks used in the FPGA. The time taken for a DLL to lock on to a clock generated on the board is variable ($< 20ns$). This will be different every time the FPGA is turned on/reset. The level of randomness derived using this method is not good enough for a secure system. A noise source will be required.

The *Authentication Unit* implements strong encryption and secure partitioning of scratch pad memory within the FPGA, but is limited in its security by the lack of a truly random session-id. The availability of a random source on the FPGA will solve this problem.

The Protocol

The communication protocol used between the iButton and the *Authentication Unit* is an extension of the PKCS#1 [25] protocol. The PKCS#1 is an established protocol for data transfer using RSA. Packets of data are sent between the iButton and the *Authentication Unit* to transfer enough information to verify each other's authenticity. The rest of the data transfer retrieves a biometric signature and the user's privileges from the iButton. The security issues involved with the use of the biometric signature is described in Section 5.1.2.

Each packet of data transferred from the *Authentication Unit* to the iButton uses a fixed two-byte magic number. This is used by the iButton to determine if the data that it received was a valid packet, or the result of a random attack on the iButton. If the first two bytes in the decrypted packet are verified then the packet is assumed to be a valid challenge generated by the *Authentication Unit*. The next eight bytes of the packet will contain a session-id, which will be used for all the data transferred by the iButton to the *Authentication Unit*. This session-id is generated by a pseudo-random number generator on the *Authentication Unit*.

If a challenge is replayed to the iButton, the iButton will reply with the same set of packets every time the same challenge is repeated. Therefore, the replay attack will not reveal any information to the attacker that he does not already have.

Once the iButton has detected a valid challenge, it will be ready to send thirteen packets of data containing the biometric signature and the privileges of the owner of the iButton. Thirteen packets of encrypted data, each containing the session-id and a sequence number, are retrieved by the TINI and sent to the *Authentication Unit*. The session-id of each packet is compared at the *Authentication Unit*. If the session-id generated were truly random a replay attack on the authentication system could be prevented. The sequence number in each packet ensures that the order of the packets is not modified en route to the *Authentication*

Unit. This prevents an attacker from reordering the packets to break the protocol.

The *Authentication Unit* implements a timeout to ensure that it does not get stuck in a state waiting for a packet to arrive from the iButton. The authentication system would hang until it were reset, if the timeout is not used.

The data transfer protocol used between the iButton and the *Authentication Unit* is resilient to packet dropping, packet re-ordering, packet corruption and packet modification, whether malicious or not. A session-id is used to protect the system from the replay of packets generated during the a previous session. This is not foolproof, since the session-id is currently not generated by a true-random number generator.

Results

The results of the analysis of the discovery protocol in the previous sections are summarized in Table 5.1 and Table 5.2.

5.1.2 The Biometric Protocol

The biometric signature of a user is retrieved from the iButton using the discovery protocol. The signature is a compressed, encrypted version of the data that is used by the Secugen FDA01 [16] to verify a user's fingerprint. The signature is sent to the FDA01 encapsulated in an instruction to verify the current user. The FDA01 runs the verification check and returns the result of the biometric verification. The *Authentication Unit* is secure, but the channel of communication between the FDA01 and the *Authentication Unit* is not.

	Security Feature Implemented
iButton	tamper-proof real-time clock PIN protected code and keys 1024-bit crypto co-processor FIPS 140-1 certification
Authentication Unit	512-bit RSA secure memory map pseudo-random 64-bit session-id
Protocol	extension of PKCS#1 protection from random challenge use of session-id packet ordering time-out

Table 5.1: A summary of the security features of the modules/protocols used in discovery.

The FDA01

The FDA01 has an ARM processor that processes the image of a fingerprint to extract the minutiae and then compares it to the fingerprint supplied as a reference. The software that carries out this comparison is proprietary. The algorithm used to match fingerprint templates is characterized on the basis of the False Acceptance Rate (FAR) and the False Rejection Rate (FRR) (see Section 2.1.3). The FDA01 is capable of running at different levels of security, which in essence fixes the position of the system on the ROC curve (see Figure 2.2). Different security levels result in different values of FAR and FRR for the system.

	Security Hole Detected
iButton	
Authentication Unit	unable to generate real random number
Protocol	susceptible to replay attack

Table 5.2: A summary of the security holes in the modules/protocols used in discovery.

The template generated by the FDA01 contains information on the minutiae extracted from the fingerprint of the user. The minutiae correspond to characteristic features of the user's fingerprint. This information needs to be kept secure since the knowledge of these features could be used to generate a template for use in another biometric system. The user's fingerprint would then be compromised for ever. The FDA01 encrypts the minutiae that is extracts from the user's fingerprint into a template of a fixed size (four hundred bytes). This protects the user's biometric signature.

The Optical Sensor

The optical sensor scans the fingerprint window line by line and streams the results of the scan to the controlling device. The data that is sent to the controller is not encrypted in any way. This data will give a very good reproduction of the user's fingerprint. If this channel is not secured, physically, the image of the user's fingerprint will be accessible to any attacker.

This sensor has its limitations. It is not capable of properly resolving a dirty finger. A dirty finger can only be resolved properly with an ultra-sonic sensor. The optical sensor is incapable of determining whether or not the fingerprint that it is scanning is live. If an image of the user's fingerprint is captured, it could be placed on the window of the optical scanner in place of the real finger. The scanner would be fooled into accepting it as a valid

fingerprint. This can only be prevented by using a capacitive CMOS sensor, which has limitations of its own.

The window of the optical sensor is specially coated to prevent a residual fingerprint from being left on the glass. This protects the system from an attempted break in using the residual fingerprint as the biometric.

The Protocol

The FDA01 receives commands from the host controller via an RS232 serial connection. It processes the command and sends back a reply to the controller. The reply depends on the operation that the FDA01 has performed.

The communication between the FDA01 and the controller, which in this case is the *Authentication Unit*, is not encrypted. The command from the *Authentication Unit* to verify a user, contains the template of the fingerprint retrieved from the iButton. This template is encrypted so that it does not reveal the user's fingerprint, but is not time-stamped. The command issued by the *Authentication Unit* can be recorded and replayed to the FDA01 in future authentication runs. Since the command issued is a stream of bytes in clear text, the bytes that correspond to the biometric template can also be swapped, while in transit, with the biometric template of the attacker.

The response from the FDA01 to the *Authentication Unit* is the result of the biometric verification. As with the command, the response is not encrypted. Since the communication protocol of the FDA01 is open knowledge, a response can be synthesized by an attacker to indicate that the verification succeeded.

The communication between the FDA01 and the *Authentication Unit* must be secured by either using an FDA01 unit which has custom firmware or by physically securing the con-

nection between the FDA01 and the *Authentication Unit*. The link between the biometric sensor and the FDA01 must be physically secured. Securing a second link will not add much to the cost or the complexity of the system.

Results

The results of the analysis of the biometric protocol in the previous sections are summarized in Table 5.3 and Table 5.4.

	Security Feature Implemented
FDA01	False Acceptance Rate = 0.0% (default) False Rejection Rate = 0.45% (default) encrypted biometric template
Optical Sensor	anti-residual coating high resolution
Protocol	

Table 5.3: A summary of the security features of the modules/protocols used in biometric verification.

5.1.3 The Configuration Protocol

Once a user has been authenticated, the access privileges that were retrieved from the iButton, are used to decide which configurations to load into the FPGAs (see Section 3.4 and Section 4.4). If a configuration is not available on the SLAAC board, it is requested from the client attached to the radio. The *Configuration Manager* parses the key-tables for the tags of the configurations to which the user has access. The requests are sent to the client

	Security Hole Detected
FDA01	
Optical Sensor	fingerprint scan data transferred in clear text unable to detect life of finger
Protocol	command to FDA01 sent in clear text biometric template can be extracted from command result can be synthesized

Table 5.4: A summary of the security holes in the modules/protocols used in biometric verification.

through the Rabbit.

The Rabbit

The Rabbit [50] is an embedded controller which is used to relay data communication between the FPGA and an ethernet link which terminates at the client. The Rabbit is not cognizant of the contents of the data that it is relaying. Since the data retrieved from the client is always encrypted, if the data is modified at the Rabbit, the modifications will be detected when the configuration is being used to program an FPGA. If tampering is detected the configuration will not be used. This may be used by an attacker to prevent a valid configuration from being loaded into the system, but it does not give the attacker any unauthorized privileges.

The Configuration Manager

The *Configuration Manager* is the unit within the FPGA that processes the key-table and decides which configurations to request from the network. Since it is embedded in the FPGA,

it is secure from probing. Each retrieved configuration is not checked by the configuration manager until it is required for programming an FPGA. If the configuration is invalid, it will be detected when the configuration is decrypted.

The Protocol

The communication between the *Configuration Manager* and the client is relayed by the Rabbit. The request from the *Configuration Manager* and the configuration that is returned by the client may be snooped on the network. The configuration is encrypted with a key that is available only in the key-table of the user. An attack on the configuration would require the decryption of a blowfish cipher. In this implementation, a 32-bit cipher is used. This can be increased to up to 448 bits if the need arises. The size of the key has no impact on the time taken for decryption.

If either the request to the client or the returned configuration is modified in any way, the decrypted bitstream will be invalid. This will be caught by the hardware that implements the programming functions. The worst that an attacker could do is to prevent the radio from starting up with a valid bitstream. Since each configuration is linked with a key in the user's key table, replaying the wrong configuration to the radio when one is requested will not cause any damage to the system. It will not give the attacker any unauthorized privileges, either.

Result

The results of the analysis of the configuration protocol in the previous sections are summarized in Table 5.5 and Table 5.6.

	Security Feature Implemented
Configuration Manager	
Protocol	448-bit blowfish encryption possible

Table 5.5: A summary of the security features of the modules/protocols used in configuration loading.

	Security Hole Detected
Configuration Manager	invalid configuration not caught immediately
Protocol	uses 32-bit blowfish encryption

Table 5.6: A summary of the security holes in the modules/protocols used in configuration loading.

5.1.4 Summary

In this chapter the entire process of authentication, right from the time the user inserts his token to the time the configuration is loaded onto the system, is analyzed for potential holes in security. The results of the analysis are tabulated at the end of each section. The biggest security risk in the system is caused by the inability to generate a true random number in the FPGA. This leaves the system susceptible to replay attacks until such a time when a true random number can be generated on the FPGA.

5.2 High-end Systems – SecurID: A Case Study

The security of the proposed radio hinges on effective user authentication. In this section the method used to authenticate a user in this system is compared to user authentication

technology available in the marketplace. It was not possible to study a high-end, full fledged, working, biometric authentication system since these systems are usually custom engineered for an application and the details of the implementation are not revealed. Two-factor authentication using a token and a password, however, has matured enough that some companies reveal the details of their systems. RSA Security markets a platform called the *SecurID* (see Section 2.4). This platform is widely accepted in the industry. Over 8000 enterprises worldwide have 12 million users registered to use SecurID. The performance of the authentication system is broadly divided into the security of the user devices used, the security of communications protocols used and the range of users and applications supported. In the following sections, the proposed authentication system is compared with the RSA SecurID platform under these three broad classifications.

5.2.1 User Devices and Authenticating Factors

In any user authentication system, the user has to be provided a means to identify himself to the system. The means of identification can be password based, token based or biometric. Each means has its own strengths and weaknesses. Each has its own identifying factor. A system can incorporate any combination of factors to make the authentication process more secure. The number of factors used in the authentication process is a general indication of the strength of the process.

Both SecurID and the proposed radio use a two-factor authentication process. SecurID uses a token and a user memorized pass code. The radio, on the other hand, combines a token with a biometric to authenticate a user. A token must be immune to tampering, since loss of the data on the token is akin to losing one factor in the identification process. If the token were lost, it must not be possible to glean any information about the user from the token.

	Authentication System	
Item	*Radio*	*RSA SecurID*
token tampering	secure	secure
unauthorized token use	secure	insecure
token encryption	very strong	strong
second factor used	biometric	password
encrypted second factor	yes	partial
second factor storage	in token	user memory

Table 5.7: A comparison of the devices and the factors used during authentication.

The SecurID token, stores a hash function and a 64-bit symmetric key. It has an internal clock which is synchronized with all other SecurID tokens and authentication systems. The internal time is encrypted with the 64-bit key and the hash function is applied to it to create a unique six to eight digit id. This id is used along with the pass code to generate a one-time pass id used to authenticate the user. The token used in the radio system is an iButton. The iButton stores the biometric signature. It is secure against tampering and will self-destruct if it detects an intrusion. Data never leaves the iButton unencrypted. A 512-bit key is used to encrypt the data. This key is much stronger than that used by the SecureID token (see Section 3.5). The one-time id that is generated by the SecurID token has a maximum of eight decimal digits, and is always the same size for a given implementation. Since this id changes every minute, an intruder has a one minute window within which to try 10^8 different combinations of the token value. In practice, to protect the system against denial of service due to clock drifts between the token and the authentication server, the window is usually stretched up to five minutes. The data generated by iButton is always binary and 512-bits wide. The data is encrypted with a nonce generated by the radio and each nonce can only be used once. If the authentication run fails, a new nonce is required from the radio. Each

authentication run, in the radio, is independent of other runs. Therefore an intruder is not guaranteed access even with an exhaustive search of the space of combinations of 512 bits (2^{512} combinations).

The second-factor used in the radio is a biometric. The SecureID system uses a pass code. The pass code is a numeric password and it suffers from all the drawbacks of the latter. A user has a tendency to write it down somewhere as an aid to memory. Since only decimal digits are used, the pass code is susceptible to an exhaustive search. The pass code needs to be changed every few months. The biometric used by the radio, does not need to be remembered. It is a physical characteristic of the person that needs to be authenticated and it cannot be duplicated easily. It is more secure than a pass code for these reasons. Further, the biometric is communicated to the system directly by the iButton. Hence the iButton is intrinsically tied to a user. If it is stolen, it will be of no use, since the authorized user has to be present for the biometric check. The actual biometric signature is never transmitted out of the button in clear text. To steal the biometric, the encryption key needs to be cracked. On the other hand, if a SecurID token is stolen, it will continue to generate the secure one-time id that is one of the factors of the authentication. The intruder will then possess one of the factors and will need to guess only the pass code.

The combination of the biometric stored on the iButton is, therefore, more secure as a set of authenticating factors than the combination of the RSA SecurID and the associated user pass code. Table 5.7 summarizes the results of the comparison of factors used in the two schemes.

5.2.2 Communication Protocol

It is always difficult to secure the communication protocol in an authentication system. The communication protocol of a system comprises the following: retrieval of factors from user, forwarding of the factors to the authenticator and encryption of the channel used to the transfer the factors. Table 5.8 shows the security impact of different parts of the authentication system protocol. Table 5.9 summarizes the results of the comparison of the protocols of the two schemes.

Item	*Attack*	*Security impact*
user input	iButton spoofed	critical; denial of service
		critical; continuation of service after removal of ibutton
iButton-radio	replay attack	critical; but finding the same nonce is not easy
	spoof packets	none; the packets are encrypted with a nonce
	drop packets	none; timeout implemented
	reorder packets	none; packets are numbered
protocol encryption	exhaustive attack	none; 512-bit key is cryptographically secure
Secugen-radio	replay attack	critical; this link must be physically secured
	theft of biometric	critical; this link must be physically secured
	spoof result	critical; this link must be physically secured
config encryption	exhaustive attack	very low; 448-bit key is considered secure
config transfer	replay attack	none; tag & key have to be in user privileges
	transfer corrupted	none; the configuration is verified before programming
	partial transfer	low; the system will hang until all data is received

Table 5.8: The effect of attacks on the authentication system proposed.

The electronic transfer of the factors from the user to the system could require user interven-

tion. The pass id generated using the pass code and the one-time id of the SecurID token, needs to be typed into a local software client. While it is being entered into the system manually, the pass id is available to an unscrupulous co-worker, for example, in a human readable format. The biometric is transferred from the iButton directly into the hardware without user intervention. This sort of communication is more difficult to capture and replay. The software client used in the SecurID system is called the terminal. It is susceptible to a replay of the pass-id as long as the replay occurs within the authentication window defined by the server. In fact the pass-id need not be replayed at the same terminal. Once the time bound pass id is compromised, it can be taken to another terminal and played back to gain unauthorized access at that terminal. Since the binary data transferred to the radio by the iButton is encrypted with a one-time nonce, it is not even susceptible to an exhaustive search. The encryption scheme used in the radio authentication system is a 512-bit RSA cipher. In comparison, the a 64-bit RC5 cipher is used to encrypt data in the SecurID system. The latter is inherently less secure than the former, due smaller key-size used.

The SecureID system requires that all the data retrieved from the user, during the authentication, be transferred to a remote authenticating server. This server receives the request for authentication, processes the request and sends back the result to the terminal. Since the server is usually at a remote location, all communication has to be over a local area network or a wide area network. Communication between the client and the server can be compromised in this network. Denial of service attacks, and even replay attacks on the server are not uncommon. All the data from the iButton is sent directly to the radio. After the radio has received the data all the data is retained within the radio. There is no point in the communication path between the reception and the authentication that is susceptible to an attack from outside. Fewer links need to be secured in the radio than in the SecurID system. The probability of a compromise increases with the number of links that can be compromised. Therefore the authentication system in the radio provides a higher level of

	Authentication System	
Item	*Radio*	*RSA SecurID*
user input	none	pass code + one-time id
clear text transfer	none	pass code
encryption scheme	RSA 512-bit	RC5 64-bit
communication intensity	low (direct to system)	complex (client/server/agent)
terminal	radio hardware	local software client
authenticator	radio hardware	remote software server
information interception	physical presence	remote access
attack at terminal	replay	replay attack
replay attack	susceptible	susceptible within replay window
authenticator attack	not accessible	network attacks

Table 5.9: A comparison of the protocol security of the authentication systems.

security to the communication protocol.

5.2.3 User Support and Access Rights

The final goal of a user authentication scheme is to validate a user and determine the access rights granted to that user. Since this data has to be stored and indexed, system parameters limit the maximum number of supported users and the set of access privileges that may be granted. Further, the access rights of a user or a group of users might need to be updated. Depending on whether the privileges are granted by the system locally or remotely, the ability to modify access rights differs from system to system. Table 5.10 compares the support for users and access rights available in the radio with those available in the SecurID system.

	Authentication System	
Item	*Radio*	*RSA SecurID*
maximum supported users	unlimited	20 million
maximum supported services	2 billion	20000
access control devices	configurations	commercial hardware
centralized control	no	yes
user access rights up datable	no	yes

Table 5.10: A comparison of the support for users and the access rights in the authentication systems.

The radio can support a maximum of two billion configurations. A configuration gives a user access to a certain stream of data or a processing capability. Therefore the maximum number of services that the radio supports is limited only by the number of configurations. Each user is given a token, which contains the tags for the configurations that the user is allowed to use. Therefore, the access rights are stored on the user's token. Since there is no limit to the number of tokens that can be distributed, there is no cap on the number of users that the radio supports. The SecurID system is limited in the fact that the access rights for each user are stored on a centralized authentication server. When queried, the server returns the access rights granted to a user. The number of users and services is limited to the capabilities of the server. Each server in the SecurID system supports one million users and can control access to a thousand different services. A maximum of twenty servers can be connected together in a network to raise the capacity of the system to twenty million users and twenty thousand services. The radio is capable of connecting a greater number of users to a greater number of services than SecurID can.

When it comes to updating user access privileges, the radio is not as flexible as the SecurID

system. The set of configurations that a user can access is limited to those stored on the iButton assigned. To update a user's access rights the iButton must be reprogrammed with the tags of the new set of configurations. This is not very convenient if the user is out in the field. All user access privileges in the SecurID system are controlled by the set of centralized authentication servers. Updating a user's privileges is no more challenging that changing entries in the server's authentication database.

5.2.4 Summary

In this section the authentication system of the radio was compared to that of SecurID, a commercially available system developed by a RSA Security. RSA Security is a company of high standing in the market. The authentication system used in the radio provides a higher level of security than that afforded by SecurID. Unlike SecurID, the authentication system on the radio is not centralized. Therefore it is more difficult to update the access privileges of a user of the radio than it is to update the access privileges of a user of the SecurID system.

5.3 Standard API

Standard authentication Application Programming Interface (API) are available, but they are only concerned with the collection of data, the encryption of the data and the protocol used to transfer the data. The effectiveness of the API is limited by the hardware that is used. The authentication APIs that come to the forefront are the Javacard, JCE, JAAS, BioAPI and the BAPI. Their effectiveness in attaining the level of security desired is discussed in this section.

The system proposed in this thesis combines various devices and methods of authentication

to achieve a high level of security. This involves the interaction of various hardware devices and their controlling software/state-machines. The security of the system depends on the security of the devices used as well as the robustness of the software. The software is designed to use the required functionality on the hardware devices in the most efficient and robust manner possible. It is not written with the aim of maintaining compatibility with as many existing devices as possible.

The gradual drift towards standard biometric and smart card APIs in the industry results applications being developed that have multiple layers of abstraction between the software and the hardware used. At the lower levels, the hardware is required to provide standard functionality that is available to the higher layers through hooks in the device drivers. The higher layers abstract the functions of the hardware to present a generic API for the application to use. Multiple layers result in less efficient software, since the API has to take care of every capability available to all conforming hardware.

In general the API, itself, do not provide the capability to acquire user information and to process it to arrive at a decision. The API just defines the set of functions that a device driver needs to support as well as the set of functions that are available to the applications programmer. Since, by itself, the API does not provide any functionality, the security of an implementation is limited by the capabilities of the hardware used. In our application the API would add levels of inefficiency to the implementation. In addition, the biometric API (BAPI and BioAPI), require the underlying interface to provide the software with raw biometric data. This is used to create Graphical User Interfaces, that improves the marketability of the product, but which hinders the development of a secure system using these API. In all fairness, the BioAPI does control the ammount of data that is returned to the application to protect the system from a hillstepping attack. It incorportates key management techniques that can be used in distributed client server models of the system. In the proposed scheme, the application (data channels) and the authentication subsystems are isolated to

the fullest extent possible. The only information that crosses the boundary is the table of user privileges (key-table). Therefore, the implementation of authentication is secured from the implementation of the application, to reduce the possibilities of a compromising the security of the system.

These API do not target embedded applications like the secure radio described in this thesis. The application written for the API must be run on a Personal Computer and the matching algorithms are executed on this machine. The biometric device being controlled is just used as a fingerprint scanner and template pre-processor. In an embedded system, this would be result in a colossal waste of resources. The API, with all its requirements, would have to be implemented in hardware which is not an easy task. The biometric device would not used to its full capacity, since most of the processing would done offline.

The Javacard, JCE and the JAAS are similar to the BAPI and the BioAPI. They provide a layer of abstraction between the application and the unit that carries out the function. The Javacard API incorporates a standard method to read and write to a Java enabled smart card. The JCE and JAAS provide a standard interface to Java applications that require to use cryptography and authentication. The API does not provide the cryptographic or authentication methods: they only provide a standard interface to modules that can be plugged in to perform the task.

The proposed authentication system has no use for the standard API, since these API are inefficient and insecure for an embedded application. The API will not take the place of custom state-machines/software that exploit the required functions of the embedded device used.

5.4 Desktop Devices

The move to develop a standard API, mentioned in the previous section, has promoted the availability of user authentication devices that are within the price range of a home/office user. Keyboard and mice are available with embedded smart card readers and biometric sensors. Desktop iris scanning devices are available but are still expensive. The common factor in all these devices is that the processing for user authentication is performed on the host PC. Usually, the data extracted from the device is transferred to the host either in clear text or encrypted with a weak encryption key. This is because almost all processing is done in software to lower the cost of the hardware used. The algorithms used for authentication are limited by the capabilities of both the hardware and the software. This puts a cap on the quality of the authentication performed. Sometimes the number of users that the device can distinguish is restricted to just a hand full.

These devices are targeted at the uninformed user. The devices and the accompanying software do not have the security capabilities of a high-end system. The user is lulled into a false sense of security, which can be counter-productive from the security standpoint. Devices of this type have no place in a secure environment.

5.5 Summary

This chapter attempted to place the proposed authentication scheme into perspective. The security of the proposed scheme was analyzed. The implementation of the scheme compared favorably to a high-end authentication system manufactured by RSA Security. The use of standardized API and low cost biometric devices was discouraged since they did not provide the required levels of security.

Chapter 6

Analysis and Results – Data Processing

The focus of this research is to implement the authentication scheme to secure a radio. The radio is the gateway for data flowing to the attached client from the airwaves. Data throughput, latency and error rate are important parameters for characterizing the performance of the radio. In this chapter, the performance of the radio is analyzed using these three parameters. Further, system enhancements that facilitate reliable communication are tested to check their impact on performance.

A network of radios can be configured in many ways. For the purpose of this analysis, a point-to-point connection is assumed, in which the flow of data is unidirectional, with only acknowledgments flowing in the opposite direction. At the receiving end of the network is a functional radio that reads data from the airwaves and feeds it into an Ethernet (Local Area Network) LAN. A wireless radio modem is used to transmit data at the other end of the wireless network. The system under analyses in this section is shown in Figure 6.1. In this configuration, the characterization of data flow is greatly simplified. The results stated

in this chapter, can be extrapolated to more complex radio links.

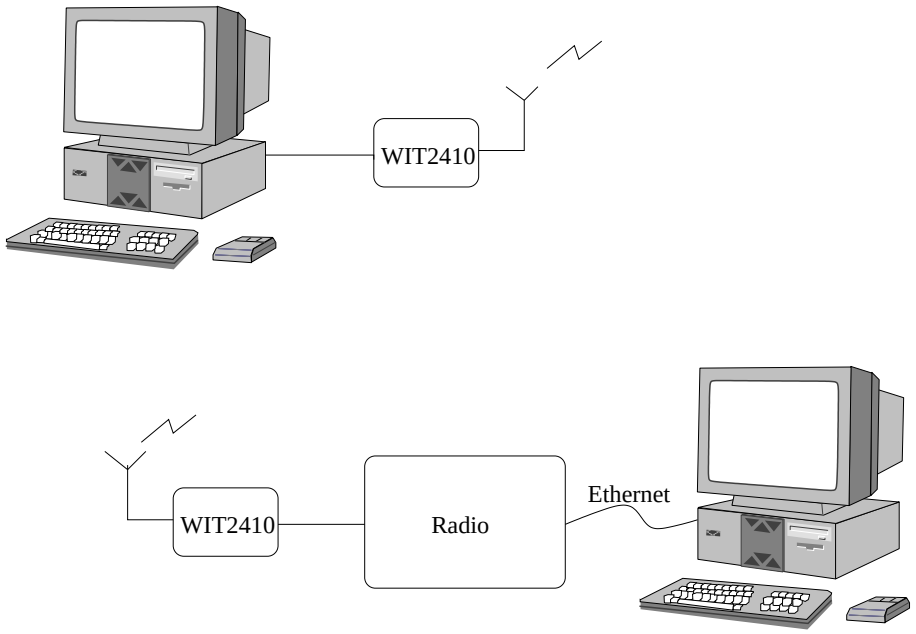

Figure 6.1: A block diagram of the radio link that is analyzed in this chapter.

6.1 Data Throughput

Data throughput is one of the primary concerns in every network. In this section, the actual throughput that is achieved by the radio is measure and the theoretical maximum throughput is determined. The bottlenecks in the system that limit throughput are identified so that they may be upgraded when the requirement arises.

Figure 6.2 shows the relationship between the measured throughput of the radio (in bits of payload information transferred) and the size of the packet (which includes a header, payload and trailing CRC value), for three different links between the source and the destination. The three different connections between the transmitter and the receiver are: a serial crossover

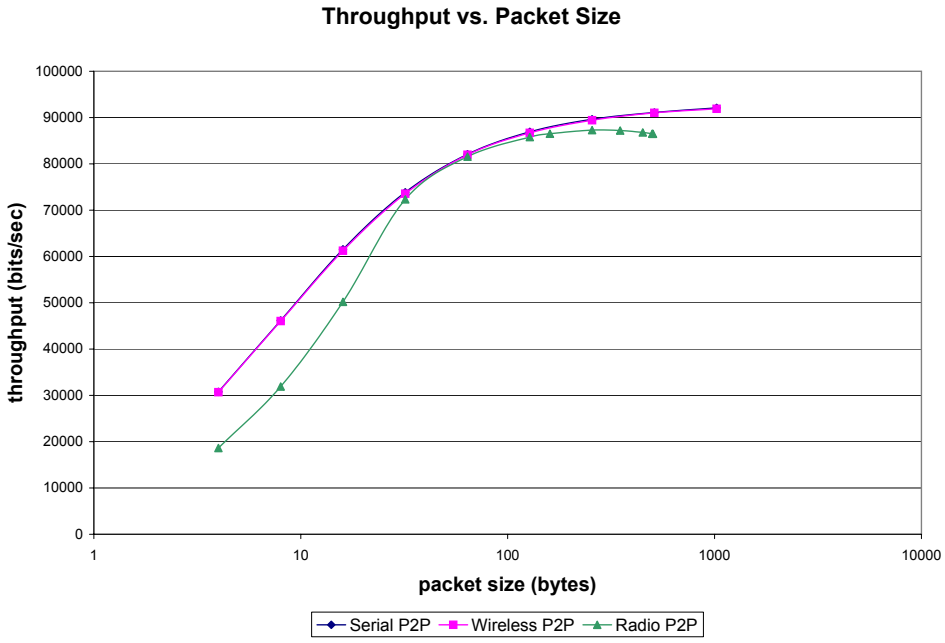

Figure 6.2: Throughput of the radio vs. transmission packet size.

link (*Radio P2P*), a wireless link using the WIT2410 radio modem (*Wireless P2P*) and a wireless link using the radio (*Radio P2P*). The serial crossover link and the wireless link using only the WIT2410 radio modems are reliable and provide an upper bound to the data throughput between the source and the destination. The radio link is implemented as a superset of the other two links, since it has a serial interface to the WIT2410, which, in turn, forms the wireless link between the source and the radio.

The throughput of a communication link is governed by the *system overhead* and *packet error rate*. The *packet error* rate is nil for the first two types of connections, viz. *Radio P2P*

and *Wireless P2P*. The radio link is prone to packet loss (see Section 6.2). The packet error rate increases with the size of packets used. The effect of packet loss is more prominent when larger packets are used. The throughput of the radio link is shown to increase up to a point after which it begins to drop instead of flattened off. This is a direct result of the increase in packet loss rate as the packet sizes increase.

System overhead in this radio has components related to the *packet flow rate* and to the *data flow rate* through the radio. The results in this section assume that there is a constant flow of data through the link. Hence, latency of the link is not considered here. Every component in the radio, which processes data one packet at a time, can potentially add to the system overhead with delays that are proportional to the number of packets processed. The TCP/IP stack on the Rabbit is one such component. The Rabbit runs through a software loop, in which it reads one packet of data, if available, from each channel, and then writes the data to the destination output channel. After each iteration of this loop, the function that processes the TCP/IP stack is invoked, which takes approximately one millisecond if there is no processing, and up to a hundred milliseconds if there is data to be processed. The system overhead resulting from processing the stack is proportional to the packet transfer rate. If the rate of data transfer is kept constant, the system overhead will increase with a decrease in size of the packet and vice-versa. The instructions that the Rabbit executes to process the TCP/IP stack is in a library, which contains functions used in the Rabbit Development Kit.. Hence, the direct analysis of the source code to determine the maximum network throughput is not feasible. The maximum Ethernet network data that the Rabbit is able to generate, when it is not performing any other task, has been determined experimentally. This is presented in Table 6.1. The table also lists the results of the same experiment run with the Rabbit processing data fed to it from the FPGA. The point to be noted is that the network throughput generated by the Rabbit depends on the size of the network packets. If the packets are small, the number of packets that need to be

processed to transfer a given amount of data is large. The overhead associated with packet processing increases with a decrease in the packet size. The data in the table shows the cap on the maximum throughput of the radio for different packet sizes.

	Network Throughput (bits/second)		
Network Packet Size	*Unloaded*	*Loaded (500byte)*	*Loaded (100byte)*
4	39989	14045	10708
8	79459	28612	21086
16	156869	51478	41045
32	300974	106151	79623
64	552878	195251	144604
128	950085	346810	249598
256	1482787	527630	390765
512	2042051	730214	538605

Table 6.1: Output network performance of the Rabbit. The three values in each row show the output data rate that the Rabbit can sustain when it is not loaded, when it is loaded using 500 byte input packets and when it is loaded using 100 byte input packets, respectively.

Further, every packet that is processed by the radio has eight bytes of appended header/CRC information. When small packets are used, these eight bytes form a significant percentage of the transmitted data. The effective throughput of the system, which is the rate at which the payload data reaches the receiver, is lower when smaller packets are used.

A delay that is independent of the number of packets transferred, but is dependant on the actual amount of data that has been transferred is said to constitute the second component of total system overhead. The *Rabbit Interface* is the only module in the radio which takes more than one clock cycle to process a byte of data. The Rabbit, which reads data out

through the Rabbit Interface, is slower than the FPGA. The Rabbit Interface on the FPGA is capable of feeding one byte of data to the Rabbit every four clock cycles (a maximum throughput of 6MB/s with a 24MHz clock). The Rabbit takes 30.1μs to read each byte of data, which translates to a maximum throughput of 33.222KB/s. This is determined from an analysis of the compiled assembler code for the Rabbit. The link between the Rabbit and the FPGA is limited to a maximum throughput of, approximately, 33.2KB/s.

The throughput of the serial crossover connection for a given packet size, is the practical limit on the throughput of the radio for that packet size (see Figure 6.2). The WIT2410 radio modem, which is used in this radio, is connected via a 115200 baud serial interface. The maximum data transfer rate of the radio modem is 230400 baud in either direction. The radio modem is, therefore, able to match the data throughput rates of the serial connection. Since data is fed to the radio via the radio modem, the throughput of the radio modem governs the rate at which data is received at the radio. The UART at receiving end, on the FPGA, facilitates the transfer of data from the radio modem. With the current system clock running at 24MHz, the UART is able to process data at rates up to 1.5MB/s, but since both the serial link and the radio modem are slower than the UART, the maximum throughput on the input to the radio is 11.5KB/s. This is the maximum throughput of a 115200 baud serial link if every byte of data is padded with a stop and a start bit during transmission.

Table 6.2 lists the bottlenecks that could potentially limit the throughput of the radio. The bottleneck in the data path at present, is the serial link between the radio and the transmitter. The maximum data throughput of the radio in this configuration is 11.5KB/s. This is the upper limit to the throughput shown in Figure 6.2. If the serial link were replaced with a faster connection at the transmitter, the WIT2410 would become the bottleneck in the system. If the WIT2410 were replaced with a faster radio modem, the Rabbit, which is used to transfer received data to the client, will limit the throughput to 33.2KB/s.

Bottleneck	*Maximum Throughput*
Rabbit Interface	6MB/s
UART	1.5MB/s
Rabbit (Output, not loaded, 128 byte packet)	118.8KB/s
Rabbit (Input)	33.2KB/s
Rabbit (Output, loaded, 128 byte packet)	31.199KB/s
WIT2410	23KB/s
Serial Link	11.5KB/s

Table 6.2: The maximum throughput at identified bottlenecks in the data path.

6.2 Data Error Rate

Any wireless link is prone to data errors in the link. The WIT2410 wireless modem is set in a mode in which it will wait for an acknowledgment for every packet of data that it sends, before sending the next packet. In our system, the WIT2410 is essentially non-lossy. The radio on the other hand, does lose an occasional packet.

Figure 6.3 is a plot that shows the percentage of data loss as it varies with increasing packet size. The relationship between the percentage of loss and the packet size is linear. This makes it extremely plausible that amount of data that is lost is a constant percentage of the total data transmitted.

Information on packet loss is propagated through the system to the output of the Rabbit. The region of the data path in which the errors occur is infered from this information. This information reveals the number of packets that were lost in the Rabbit and the number of packets that have been marked as having a bad CRC, by the *Classifier*. The *Classifier* is a module within the radio that processes packets as soon they are received. Experiments

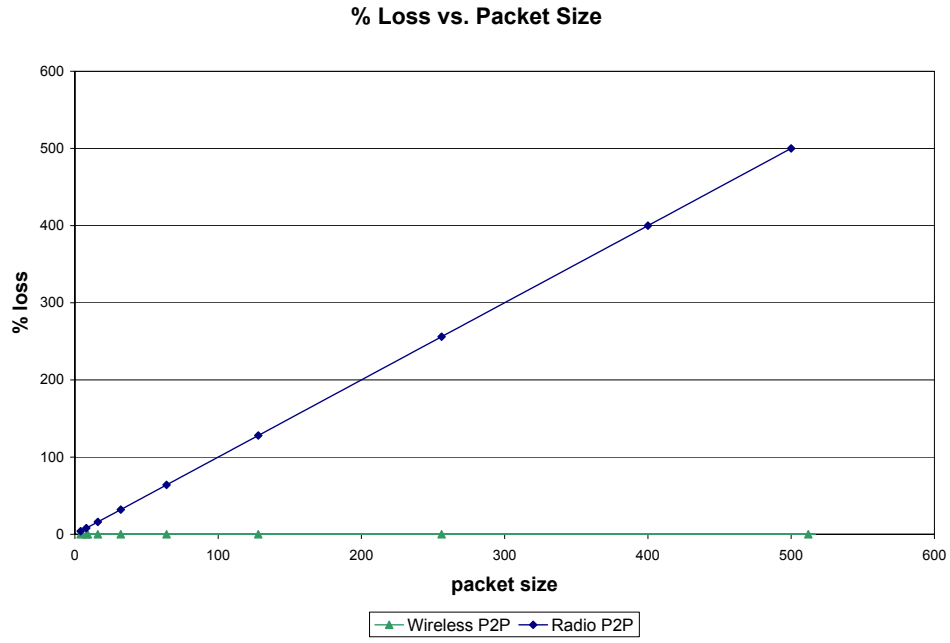

Figure 6.3: Data error rate of the radio as the packet size is increased. This is contrasted to the reliable transmission of the WIT2410 radio modem.

show that no packets are lost in the Rabbit. Ninety-seven to ninety-nine percent of all the packets that are lost are due to a bad CRC. The loss of the remaining one to three percent of the packets is attributed to the inability of the *Classifier* to capture the packet, because the packet header was corrupt. In an experiment, each packet that reached the receiver was tested for data corruption. Every packet that was received at the receiver, was free of any sort of error. Since the *Classifier* is the only module in the radio that checks for errors, it is averred that the errors occurred after the WIT2410 and before the *Classifier*.

The linearity of the plot in Figure 6.3 indicates that the corruption or loss of data in the

radio, is not random. Instead it must be cyclic and could probably be the effect of a counter.

6.3 Link Latency

The data that is sent from the transmitter is carried on the airwaves, through the radio, through the Rabbit and then across the Ethernet link to the receiver. Each element in the link introduces latency in the data path as it flows through the link. Figure 6.4 shows the latency for the link from the transmitter to the receiver. Latency is linear in packet size. It varies from approximately 15ms for a small packet to 100ms for a large packet.

The latency is calculated from the Round Trip Time (RTT) of a packet of data. Some elements in the data path work as store-and-forward device. A packet that is received at the input port of the store-and-forward device is collected, as far as the internal buffers will allow, before it is sent out at the output port. The WIT2410 radio modems, the *Rabbit Interface*, the Rabbit and the receiver are all store-and-forward devices. Each time the packet is stored and forwarded, the latency in the data path increases by the time taken by that device to receive as many bytes of the packet as it can buffer and then transmit the same over to the next store-and-forward device.

Starting at the transmitter, a path is traced to the receiver, and the latency that enters the data path at each point in the link, is enumerated. The first delay to be introduced in the path is at the serial link. The transmitter feeds data to the WIT2410 through this link, adding a delay of 101μs for each byte of data. The WIT2410 has a 32kByte buffer and it can be programmed to either store and forward the packet or send the information as it arrives. If it stores data before forwarding it to the airwaves, a latency of 10ms (which is the default hop size of the WIT2410) or a multiple of that value will be introduced in the data path. 101μs is added to the latency at UART for every byte that it reads from the WIT2410, at

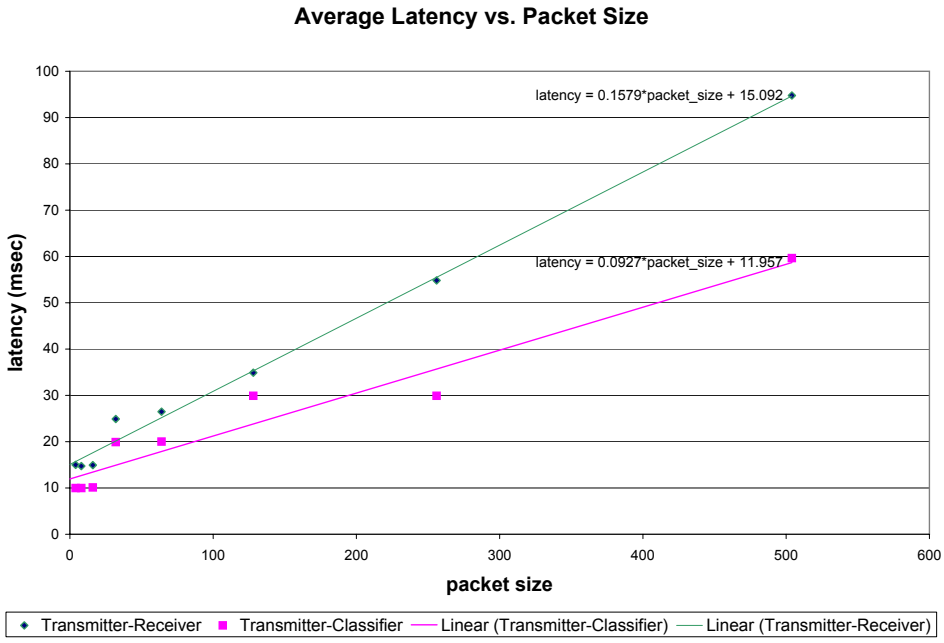

Figure 6.4: The variation in link latency with packet size.

the receiving end of the wireless link. The *Classifier* introduces a twelve cycle delay while it write the data to the *Rabbit Interface*. The delay introduced here is approximately 50ns and is independent of the size of the packet. It is negligible compared to the delays introduced by the Rabbit and the WIT2410. The Rabbit reads data from the FPGA at the rate of 1 byte every 30.1μs. A delay of between 1ms and 10ms is introduced by the TCP/IP stack in the Rabbit. Finally, the latency of the Ethernet connection to the receiver is negligible.

If the delays in the previous paragraph are added and multiplied by the size of the packet n, where appropriate, the latency of the link can be modeled as a sum of the delays in the path (6.1).

$$Latency[end\_to\_end] = (101\mu s + 101\mu s + 30.1\mu s) \times n + 10ms[WIT2410] + 5ms[TCP/IP] \tag{6.1}$$

The *HW_ACK* is designed to generate an acknowledgment for packets that request one. If this feature is turned on, the transmitter receives an acknowledgment for the packet much sooner than it would have, had the acknowledgment been generated by the receiver. The latency in this quick return path is modeled in Equation 6.2. The acknowledgment returned by the *HW_ACK* is always two bytes in size, whereas the smallest data packet must contain at least twelve bytes. Hence, the delay in the return path is negligible compared to the delay in the forward path. The model shows only the time taken in the forward path.

$$Latency[upto\_classifier] = (101\mu s) \times n + 10ms[WIT2410] \tag{6.2}$$

Equation 6.1 and equation 6.2 are quite similar to the equation for the best-fit line drawn in Figure 6.4. This validates the model developed in this section for the latency in the link.

6.4 Reliable Communication

Reliable communication is required for many applications. In general the receiver generates an acknowledgment for every packet that it receives. The transmitter might wait for each acknowledgment before it sends the next packet. This radio provides a method for an acknowledgment to be sent to the transmitter soon after the packet is received from the radio modem. The *HW_ACK* module generates an acknowledgment for packets that request one. A software acknowledgment is generated by the receiver while a hardware acknowledgment is generated by the *HW_ACK* module. Figure 6.4 shows the difference in the latency in the

generation of a hardware and a software acknowledgment.

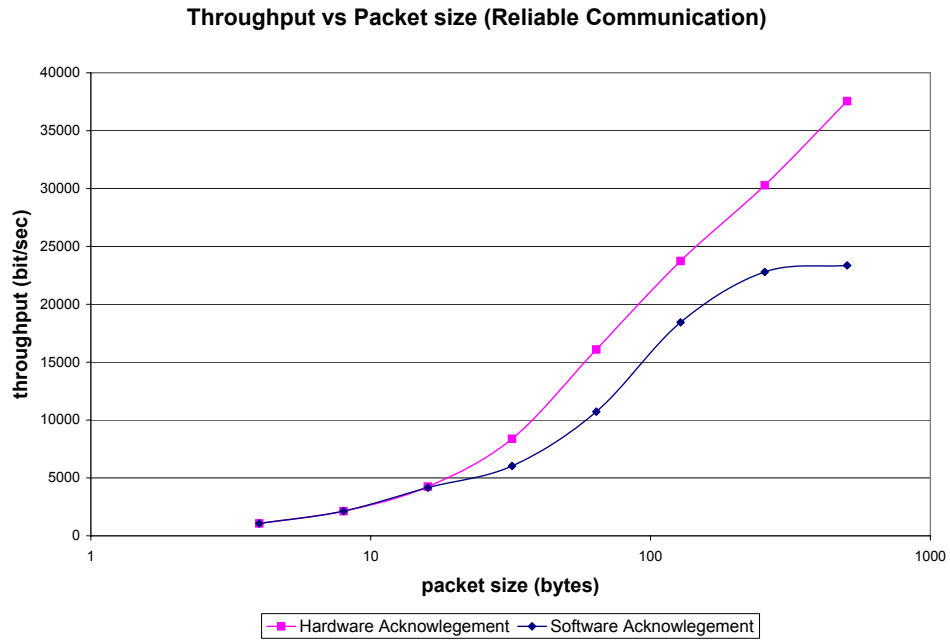

Figure 6.5: A comparison of the throughput achieved when using hardware acknowledgment and when using software acknowledgment schemes.

Figure 6.5 shows how the throughput varies between a system that uses hardware and system that uses software acknowledgments. When the acknowledgment is returned to the transmitter using hardware acknowledgment, the amount of time that the transmitter has to wait between sending packets reduces. Using acknowledgments introduces an inter packet delay that is close to the RTT of the link. This affects the throughput since the transmitter is waiting for the acknowledgment. A hardware acknowledgment reduces the inter-packet delay and thereby increases the throughput of the system.

6.5 Applications

The radio has been described in the previous sections as a high latency, moderate throughput device with a predictable packet loss rate. Table 6.3 lists a set of applications in which this radio can be used. The radio is considered for each application on the basis of the throughput required, the latency and packet loss that can be tolerated by the application.

Application	*Throughput*	*Latency*	*Data Loss*
video transmission	acceptable	poor	acceptable
audio transmission	good	poor	acceptable
text messaging	very good	acceptable	inacceptable
file transfer	good	acceptable	inacceptable
database query (low load)	good	acceptable	inacceptable

Table 6.3: The Application Matrix for the radio.

The applications listed in the table are a small subset of all the possible applications that can be run on the link created by this radio. The throughput achieved by this radio is adequate for all the applications listed. High latency is acceptable in applications that do not require real time response. Data loss is acceptable in protocols that stream video and audio data. The other classes of applications require the protocol to implement methods at a higher level that ensure reliable transfer of data. Since all these applications tolerate delays, latency will not affect them.

6.6 Summary

In this chapter the data flow through the radio was characterized on the basis of three parameters: the throughput, the link latency and the data error rate. Bottlenecks in the data path were identified and their effect on the throughput was analyzed. Data errors manage to always creep into wireless communication channels. A hardware acknowledgment scheme, which has been implemented in this radio, was analyzed. The resulting throughput was compared to that generated by a system that uses software acknowledgment. The use of the radio in certain classes of applications was evaluated.

Chapter 7

Conclusions

This chapter summarizes the work presented and gives suggestions for possible future work.

7.1 Summary

A scheme for biometric user authentication was designed for a reconfigurable radio and implemented on the SLAAC-1V platform. The radio is designed to serve data to the user depending on the user's privileges. These privileges are determined during the authentication process.

The capabilities of the proposed authentication scheme were put into perspective by comparing it to a high-end authentication scheme (RSA SecurID) and evaluating the use of standardized APIs (BAPI, BioAPI) and low-end authentication devices. The proposed biometric authentication scheme compared favorably to the RSA SecurID user authentication system. The authentication scheme that was developed is more secure than SecurID. The SecurID, on the other hand, provides more flexibility in the administration of the access

rights of large groups of users.

The biometric authentication scheme was implemented for a radio based upon reconfigurable hardware. The implementation was analyzed to identify the strengths and weaknesses of the scheme. The major weakness in the scheme was found to be the susceptibility of the authentication system to a replay attack. This is due to the inavailability of a true random number generator on the FPGA. A maximal length LFSR is currently used, but the seed value is constant. The other primary weakness, identified in the scheme, is the clear text transfer of commands and results between the FPGA and the biometric authentication module. This channel must be physically secured to protect the system from a replay attack on this channel. Apart from these weaknesses, which can be resolved as outlined in Chapter 5, the authentication system on the radio works with a high degree of confidence and imparts a level of security to the system.

The performance of the data path through the radio was analyzed on the basis of throughput, latency and reliability of the path. Using experimental data, the bottlenecks in the data path were identified so that future revisions of the radio can remove them.

7.2 Future Work

This research brings forth a few possibilities for future work:

- Develop a scheme to provide more flexibility in the administration of user privileges in a secure radio network.
- Develop a method to generate true random numbers in an FPGA.
- Improve the interfaces between the radio and both the client and the airwaves, to

remove the bottlenecks in the data path through the radio.

- Extend the analysis of the performance of the radio to a system which incorporates a two-way flow of data through the radio.

Bibliography

[1] R. Sandhu and P. Samarati, "Authentication, access control, and intrusion detection," in *The Computer Science and Engineering Handbook* (A. B. Tucker, ed.), pp. 1929–1948, CRC Press, 1997.

[2] R. E. Smith, *Authentication: From Passwords to Public Keys.* Addison-Wesley, Oct 2001.

[3] N.-Y. Lee, "Integrating access control with user authentication using smart cards," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 943–948, 2000.

[4] Y. Moon, H. Ho, and K. Ng, "A secure card system with biometrics capability," in *1999 IEEE Canadian Conference on Electrical and Computer Engineering*, no. 1, pp. 261–266, 1999.

[5] S. Liu and M. Silverman, "A practical guide to biometric security technology," *IT Professional*, vol. 3, no. 1, pp. 27–32, 2001.

[6] E. Trichina, M. Bucci, D. De Seta, and R. Luzzi, "Supplemental cryptographic hardware for smart cards," *Micro, IEEE*, vol. 21, no. 6, pp. 26–35, 2001.

[7] Molva and Tsudik, "Authentication method with impersonal token cards," in *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.

[8] D. K. Gifford, "Cryptographic sealing for information secrecy and authentication," *Communications of the ACM*, vol. 25, no. 4, pp. 274–286, 1982.

[9] "Javacard 2.2 runtime environment (JCRE) specification." WWW: http://java.sun.com/products/javacard/javacard22.html, Feb 2002. Beta Release.

[10] R. Sanchez-Reillo, "Achieving security in integrated circuit card applications: Reality or desire?," in *2001 IEEE 35th International Carnahan Conference on Security Technology*, pp. 197–201, 2001.

[11] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," in *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pp. 1–11, Nov 1996.

[12] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *IWSP: International Workshop on Security Protocols, LNCS*, 1997.

[13] "Java i-button datasheet." WWW: http://www.ibutton.com/ibuttons/java.html. Dallas Semiconductor.

[14] S. Nanavati, M. Thieme, and R. Nanavati, *Biometrics: Security Verification in a Networked World*. Wiley Computer Publishing, 2002.

[15] J. Wayman, "Error-rate equations for the general biometric system," *IEEE Robotics and Automation Magazine*, vol. 6, no. 1, pp. 35–48, 1999.

[16] "FDA01 datasheets." http://www.secugen.com/doc/sensors.pdf, http://www.secugen.com/faqs/index.htm#11. Secugen.

[17] J. L. Wayman, "Biometric technology: Testing, evaluation, results." WWW: http://www.engr.sjsu.edu/ biometrics/publications_technology.html. U.S. National Biometric Test Center.

[18] J. Woodward, "Biometrics: Privacy's foe or privacy's friend?," in *Proceedings of the IEEE*, no. 85 in 9, pp. 1480– 1492, 1997.

[19] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[20] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Transactions on Computer Systems (TOCS)*, vol. 8, no. 1, pp. 18–36, 1990.

[21] B. Schneier, *Applied Cryptograhy: Protocols, Algorithms and Source Code in C*, pp. 1–46, 151–168, 336–339, 464–474. John Wiley and Sons, Inc., 2 ed., 1996.

[22] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[23] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *Journal of Cryptology: The Journal of the International Association for Cryptologic Research*, vol. 14, no. 4, pp. 255–293, 2001.

[24] "Has the RSA algorithm been compromised as a result of bernstein's paper?," tech. rep., RSA laboratories, Apr. 2002.

[25] B. Kaliski, "PKCS#1: RSA cryptography specifications, version 2.0.." RFC 2437. Network Working Group.

[26] A. J. Menezes, v. Oorschot, C. Paul, and S. A. Vanstone, *Handbook of Applied Cryptography*, ch. 5, pp. 169–190. CRC Press, 1887.

[27] P. Alfke, "XAPP052: Efficient shift registers, LFSR counters, and long pseudo-random sequence generators," tech. rep., Xilinx, San Jose, CA, Jul 1996.

[28] G. Estrin, "Organization of computer systems - the fixed plus variable structure computer," in *Proceedings of the Western Joint Computer Conference*, pp. 33–40, 1960.

[29] P. M. Athanas and H. F. Silverman, "Processor reconfiguration through instruction-set metamorphosis," *IEEE Computer*, vol. 26, pp. 11–12, Mar 1993.

[30] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *Proceedings of IEEE symposium on FPGAs for custom computing machines*, Apr 1997.

[31] R. Bittner, *Wormhole Run-Time Reconfiguration: Conceptualization and VLSI Design of a High Performance Computing System.* PhD thesis, Virginia Polytechnic Institute and state University, Jan 1997.

[32] M. Soni, "VLSI Implementation of a Wormhole Run-time Reconfigurable Processor," Master's thesis, Virginia Polytechnic Institute and state University, Jun 2001.

[33] K. Puttegowda, "Context switching strategies in a run-time reconfigurable system," Master's thesis, Virginia Polytechnic Institute and state University, Apr 2002.

[34] M. V. Bavel, "Reconfigurable information appliances." WWW: http://www.esemagazine.co.uk/common/viewer/archive/2001/Oct/15/feature2.phtm, Oct 2001. Embedded System Engineering Magazine.

[35] P. Hamalainen, M. Hannikainen, T. Hamalainen, and J. Saarinen, "Configurable hardware implementation of triple-des encryption algorithm for wireless local area networks," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, no. 2, pp. 1221– 1224, 2001.

[36] M. Hani, T. S. Lin, and N. Shaikh-Husin, "FPGA implementation of RSA public-key cryptographic coprocessor," in *TENCON 2000*, no. 3, pp. 6– 11, 2000.

[37] S. Salomao, J. de Alcantara, V. Alves, and A. Vieira, "Scob, a soft-core for the blowfish cryptographic algorithm," in *XII Symposium on Integrated Circuits and Systems Design*, pp. 220– 223, 1999.

[38] S. Blythe, B. Fraboni, S. Lall, H. Ahmed, and U. de Riu, "Layout reconstruction of complex silicon chips," *Solid-State Circuits, IEEE Journal of*, vol. 28, pp. 138 – 145, Feb 1993.

[39] B. Dipert, "Cunning circuits confound crooks," *EDN*, vol. 45, pp. 103–12, Oct 2000. Publisher: Cahners Publishing, USA.

[40] *Virtex-II Platform FPGA Handbook*. San Jose, CA, Dec 2001. UG002 (v1.3).

[41] "RSA Security — RSA SecurID." http://www.rsasecurity.com/products/securid, July 2002.

[42] "The Power Behind RSA SecurID Two-factor User Authentication: RSA ACE/Server." Solution White Paper: AS50 SB 0601.

[43] "Interface between data terminal equipment (DTE) and data circuit terminating equipment (DCE) for terminals operating in the packet mode on public data networks," *Yellow Book of the CCITT*, vol. Fascicle VIII.2, 1981. Recommendation X.25.

[44] R. Oppliger, *Authentication Systems for Secure Networks*, pp. 1–28. Artec House, 1996.

[45] B. Schott, Crago, R. S, Parker, L. Carter, C. Chen, J. Czarnaski, M. French, H. I, T. Tho, and T. Valenti, "Reconfigurable architectures for system-level applications of adaptive computing." ISI.

[46] Dallas Semiconductor, *TINI Verification Module*, Mar. 2001. Preliminary.

[47] SecuGen Corporation, *SecuGen FDA01 Developer's Guide (Stand-alone fingerprint rcognition device with built-in CPU)*, 1998.

[48] R. Branden, D. Borman, and C. Partridge, *Computing the Internet Checksum. Internet Request For Comments RFC 1071.* ISI, Sept. 1988. (Updated by RFCs 1141 and 1624).

[49] Z-World, Inc., *RabbitCore RCM2100 Users Manual*, 2001.

[50] Rabbit Semiconductor, *Rabbit 2000 Microprocessor*, 2001.

[51] Cirronet, *WIT2410 2.4GHz Spread Spectrum Wireless Industrial Transceiver. Integration Guide*, Oct. 2001.

[52] Sun Microsystems Inc., *Java Card 2.1.1 Application Programming Interface*, May 2000.

[53] Sun Microsystems Inc., *Java Cryptography Extension (JCE) Reference Guide.*

[54] Sun Microsystems Inc., *Java Authentication and Authorization Service (JAAS) Reference Guide*, Aug. 2001.

[55] The BioAPI Consortium, *BioAPI Specification Version 1.1*, Mar. 2001.

[56] I/O Software, *Biometric Application Programming Interface (BAPI) Overview, Version 1.2*, Sept. 1998.

[57] National Institute of Standards and Technology, *Security Requirements for Cryptographic Modules*, Jan. 1994. Federal Information Processing Standards Publication FIPS 140-1.

[58] Dallas Semiconductor, *DS2480B Serial 1-Wire Line Driver with Load Sensor.*

Vita

Arya Abraham was born in Calcutta, a city in the eastern provinces of India. He graduated from Corpus Christi High School in 1995, with the Valedictorian award for Academic Excellence. Arya enrolled at the Birla Institute of Technology and Science (BITS), Pilani, for an integrated program where he worked towards a Bachelor of Engineering in Electrical & Electronics Engineering and a Master of Science in Mathematics, simultanously. After he earned his degrees in 2000, Arya worked for six months at the Research Center of Cadence Design Systems (India) Pvt. Ltd. He enrolled into the Electrical Engineering program at Virginia Tech in the Fall of 2000. He obtained a Master of Science in Electrical Engineering in 2002. His hobbies include music, reading and photography. His technical interests include reconfigurable computing, embedded systems and VLSI.