# A TECHNICAL OVERVIEW OF THE INTERNATIONAL RADIO SECURITY SERVICE API

Anthony DiBernardo (Harris Corporation, Rochester, NY; adiberna@harris.com); Leonard Picone (Harris Corporation, Rochester, NY; lpicone@harris.com); Charles Linn (Harris Corporation, Rochester, NY; clinn@harris.com); Scott Leubner (Harris Corporation, Rochester, NY; sleubner@harris.com); Rafael Aguado Muñoz (Indra, Aranjuez, Madrid, Spain; ramunoz@indra.es); Javier Fernandez Alonso (Indra, Aranjuez, Madrid, Spain; jfalonso@indra.es); Alvaro Mayol Garrido (Indra, Aranjuez, Madrid, Spain; amayol@indra.es)

## ABSTRACT

The Joint Program Executive Office (JPEO) Joint Tactical Radio System (JTRS) established a new paradigm for software definable radios with their release of the Software Communications Architecture (SCA) specification [1]. This specification details requirements and outlines a framework for software based radio platforms. The intentions of this specification are to foster portability of waveform applications between divergent radio platforms. However, beyond an initial historical draft, the JTRS program currently has no provisions for a publically available security API that the broader international community can use to develop portable waveforms. Recognizing a lack of an internationally available security API for SCA based radios, the Wireless Innovation Forum (WInnF) has developed a security API called the International Radio Security Services (IRSS) API to fill this gap. This paper introduces and presents a technical overview of the major interfaces supporting streaming waveforms and wideband networking waveforms. An example of typical usage by a streaming waveform is included. Primary focus of the paper will center on the establishment of secure networking channels via network security protocols, including the application of asymmetric key management techniques. Additionally, examples of how radio manufactures can map the API to representative military and government radio architectures/topologies is presented.

## 1. INTRODUCTION

The JPEO JTRS released its initial version of the SCA with the basic goal of standardizing the operating environment (OE) for software definable radio systems. As the standard gets more mature, the initial set of goals was refined to fulfill the market requirements, focusing on the facilitation of the portability of waveforms between different platforms. In order to achieve this set of goals, the earlier versions of the SCA included both an API supplement [3] and a security supplement [4] to define the applicable system APIs between waveform components and the OE and between OE components themselves. However, later revisions of the SCA deprecated both these supplements to instead give preference to API standards being developed by an API standardization committee. Unfortunately, today there does not exist an internationally available API standard that defines the security interfaces for SCA based radio systems. Recognizing this gap, the WInnF has endeavored to create a security services API, applicable to the international community, for standardizing the interfaces of the radio security services (RSS) provided by an SCA based radio platform.

Following the aforementioned general SCA goals, the objective of this API, called the International Radio Security Service API, is to extend the waveform portability between different platforms to the security boundary. By standardizing the security API, the WInnF's IRSS API task group promotes portability of waveforms developed against those standards to platforms that provide those APIs. Figure 1 provides a brief overview of how the inclusion of this API fills the missing piece in the portability puzzle:

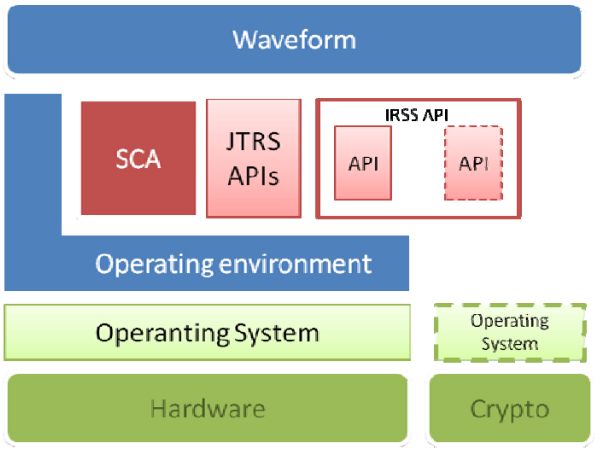


Figure 1 IRSS API Localization

As shown in Figure 1, the IRSS API is a large service that not only has to support waveform components in their interaction with the platform, but also has to provide

support among the platform components. It is essential to specify the interfaces used by the waveforms as those interfaces foster portability. On the other hand, waveforms do not connect to and use the security interfaces provided for other platform components, and thus, specifying platform security interfaces only serves to constrain platform development without adding to waveform portability. Taking into account these considerations, the IRSS API focuses on detailing security interfaces that are likely to be used by waveforms.

To develop this API, the IRSS API task group drew upon its experience with existing waveforms and on existing security APIs. In particular, the working group considered use cases for legacy circuit-based waveforms and also newer networking waveforms. Existing security APIs referenced include version 1.1 of the deprecated *Security Supplement to the SCA* [4], which defined the original RSS API for SCA based systems, and the *Common Interface to Cryptographic Modules (CICM)* [5], which is an IETF draft to standardize interfaces to cryptographic modules.

## 2. API OVERVIEW

These days, security requirements for radio systems encompass a broad range of services. However, not all of these services are directly used by the waveforms running in these systems. Typical waveform security needs include transformation of user traffic (i.e. encryption and decryption services), transmission security (TRANSEC) services, key management services, bypass services, integrity and authentication (I&A) services, and general security configuration and control services. However, grouping security services into one large interface does not promote understandability, nor does it support least privileges principles (LPP) when connecting to those services. Instead, the IRSS API factors services into several logical groupings, denoted by modules in the Interface Definition Language (IDL), which themselves contain one or more IDL interfaces. Today, these groupings include the *Control* module, the *Infosec* module, the *Bypass* module, the *IandA* module, and the *Protocol* module.

### 2.1. The *Control* Module

The *Control* module contains interfaces related to waveform configuration and control of the security services. These include the three interfaces shown in Figure 2: the *ChannelMgmt* interface, the *CertificateMgmt* interface, and the *KeyMgmt* interface.

The *ChannelMgmt* interface allows waveforms to create and configure various communications channels with the security subsystem. When created, the system allocates cryptographic resources for use with that channel. The interface provides a unique method for each channel type. When a channel is created, the methods return a channel identifier to the client for subsequent use of the channel. Waveforms use channels to exchange information with the security subsystem as part of accessing the security services. Many channel interfaces have a corresponding consumer interface. The security subsystem uses the various consumer interfaces to send data to the waveform.

The *CertificateMgmt* interface provides services that allow waveforms to validate, and retrieve certificates used by many asymmetric key protocols. The *KeyMgmt* interface allows waveforms to update and selectively zeroize keys.

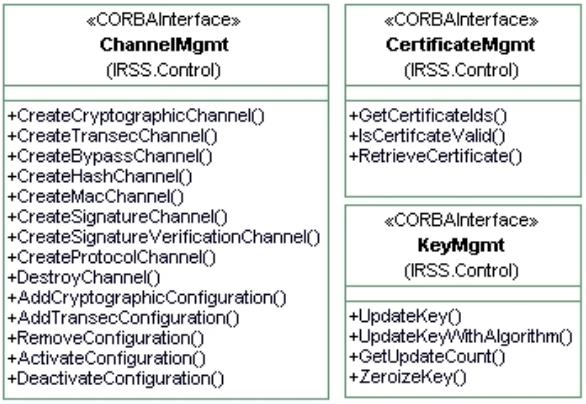


Figure 2 *Control* Module Interfaces

### 2.2. The *Infosec* Module

The *Infosec* module contains interfaces that waveforms use to access the information security services of the radio. These services fall into two categories: transformation services and TRANSEC services.

Waveforms use transformation services to encrypt and decrypt user traffic. Transformation services are supported through three interfaces (shown in Figure 3), one provided by the platform and two provided by the waveform. The IRSS provides the former interface, called the *CryptographicChannel* interface, which a waveform uses for encryption and decryption. This interface supports two modes of operation: one for circuit-based legacy waveforms, which waveforms use to stream data to the security subsystem, and one for packet-based networking waveforms, which waveforms use to send individual packets of data for transformation processing. Both modes provide flow control that allows the security subsystem to manage the flow of information into itself. Waveforms provide the latter two interfaces, called the *CryptographicConsumer* interface, which defines a standard interface for the security subsystem to push data to the waveform, and the *ControlSignals* interface, which defines a control interface that the security subsystem uses to resume flow after pausing a waveform. Like the

*CrypotgraphicChannel* interface, the *CryptographicConsumer* interface supports both streaming modes and packet modes. However, the *CryptograhicConsumer* interface differs in that flow control is not employed when passing data to the waveform – it is assumed that the waveform can accept the data, or that out-of-band techniques are employed to prevent data overflow scenarios.

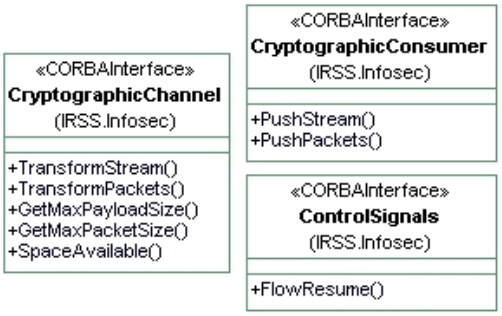


Figure 3 - Transformation Interfaces

Waveforms use TRANSEC services to provide TRANSEC cover to their transmissions. The IRSS supports this through the *TransecChannel* interface shown in Figure 4. This interface provides two modes of TRANSEC support. The first mode allows a waveform to send transmission information in to be encrypted or decrypted as part of TRANSEC cover processing. Alternatively, waveforms can use the security subsystem to generate a TRANSEC keystream using the second mode. In this mode, the waveform applies the keystream to its transmission information directly.

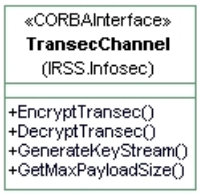


Figure 4 - TRANSEC Interface

### 2.3. The *Bypass* Module

The *Bypass* module contains interfaces that waveforms utilize to bypass control messages through the cryptographic subsystem. This type of bypass mechanism is needed in high assurance radio systems with physically separate security domains. The IRSS provides bypass support through a pair of interfaces, one provided by the security subsystem and one provided by the waveform. The former interface, called the *Channel* interface[1], allows a waveform to push control messages to the security subsystem for bypass through the crypto. The latter interface, called the *Consumer* interface[1], provides a standard interface for the security subsystem to send bypassed control messages back to the waveform. The *Bypass* module interfaces are shown in Figure 5.

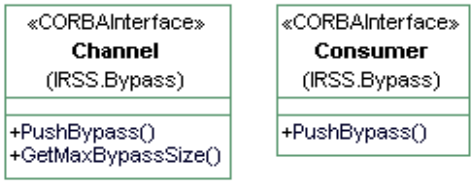


Figure 5 - *Bypass* Module Interfaces

### 2.4. The *IandA* Module

The *IandA* module defines interfaces that waveforms use to access the I&A features of the security subsystem. These features include generating hashes, generating and validating message authentication codes (MAC's), generating and validating signatures, and generating random numbers.

Waveforms access the hash, signature, and MAC I&A features through channel interfaces called *HashChannel*, *SignatureChannel*, *SignatureVerificationChannel,* and *MacChannel* as shown in Figure 6. Each of these interfaces inherits from a common *Channel* interface[2] that defines the mechanism for pushing relevant data to the security service. The derived interfaces themselves define the unique methods for retrieving the results of the requested operation.

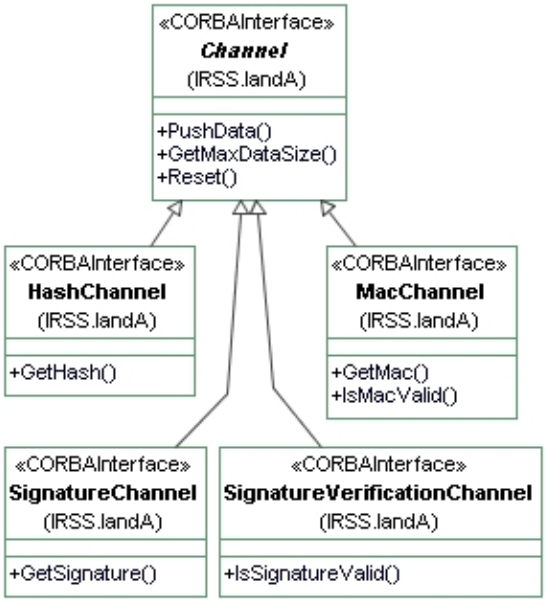


Figure 6 - *IandA* Channel Interfaces

Waveforms can use the RSS to generate random numbers through the *Random* interface as shown in Figure

[1] The fully qualified name for these interfaces is *IRSS::Bypass::Channel* and *IRSS::Bypass::Consumer*.

[2] The fully qualified name for this interface is *IRSS::IandA::Channel*.

7. This interface supports both true random number generation and pseudo random number generation using a seed.


Figure 7 The *Random* Interface

### 2.5. The *Protocol* Module

The *Protocol* module defines interfaces that waveforms use to exchange protocol messages with the security subsystem (for example, as part of an asymmetric key protocol). These interfaces define a generic messaging protocol that supports the various message exchanges needed by different protocols. Appendices to the IRSS API will standardize the specific message details needed by each protocol. As in other modules, the IRSS provides protocol support through a pair of interfaces, the *Channel* interface and the *Consumer* interface[3] as shown in Figure 8.


Figure 8 - *Protocol* Module Interfaces

## 3. OPAQUE ALGORITHM USAGE

One challenge facing an international security API is in providing an interface that is suitably generic across an expected multitude of cryptographic algorithms, protocols and implementations, while still fulfilling the need to standardize interface and semantics. Security by nature is a moving target, as evolving threats are addressed, and future waveforms will need new or modified services and protocols. Additionally, the international nature of the API requires that, in addition to public algorithms, the API must support coalition or sovereign national algorithms. Some details of these algorithms cannot be openly published, but must still be supported in the context of the IRSS.

To address this challenge, the IRSS employs a pattern utilizing opaque algorithms. When clients create channels (be they Cryptographic, Transec, Bypass, Hash, Mac, Signature, Signature Verification or Protocol), they select associated algorithms by Id. The IRSS standard does not bind a specific algorithm to an Id – this is left to the platform implementation. Clients access these algorithms (e.g. *TransformStream*) by requesting a combination of fully-specified parameters and generic parameters (usually in the form of OctetSequences). These are handled generically by the overall IRSS and then interpreted in an algorithm-specific way by a given algorithm. This defers full specification to a separate algorithm usage specification, which not only could be changed independently from the IRSS specification, but is also subject to limited access as required.

## 4. API MAPPING TO REAL WORLD TOPOLOGIES

Security domains are logically separated elements of the system whose only connection is through a cryptographic subsystem (CSS). A CSS contains one or more functional crypto modules and a single cryptographic control module. These modules are functionally separate and not necessarily physically separate. The crypto modules perform encryption/decryption, TRANSEC processing, cryptographic bypass, asymmetric key negotiation, etc. The cryptographic control module handles overall key management, policy management, certificate management, channel management, etc.

The IRSS API was designed to be applicable for radio systems that could be implemented in multiple physical topologies. The simplest topology, typical for a commercial radio, is a single waveform module connected to a single CSS as shown in Figure 9.

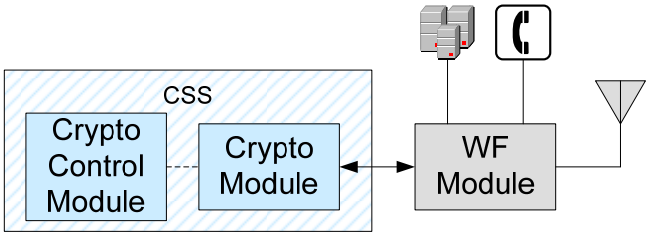


Figure 9 - Single Security Domain

A typical single channel military radio would contain a secure waveform security domain connected to an unsecure waveform security domain through a CSS as shown in Figure 10.

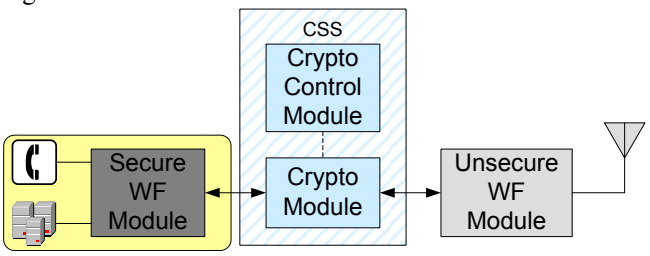


Figure 10 - Military Radio Topology

Multichannel radios could have a single CSS with multiple crypto modules that have connections to multiple

[3] The fully qualified name for these interfaces is *IRSS::Protocol::Channel* and *IRSS::Protocol::Consumer*.

waveform security domains as shown in Figure 11. The IRSS API supports these example topologies as well as others.

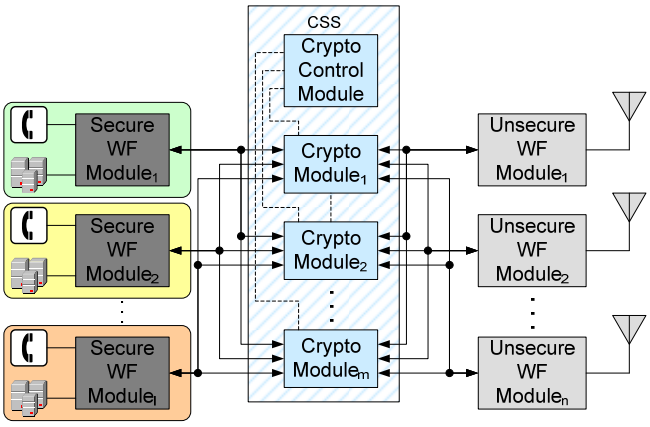


Figure 11 - Multichannel Rado Topology

### 4.1. Channels

Channels, managed via the *ChannelMgmt* interface, are communications paths with the security subsystem that share several characteristics. Since a CSS may contain multiple crypto modules (as shown in Figure 11), clients create channels on a specific crypto module that will provide the service. These crypto modules may contain multiple access points in their interfaces that define the inputs and outputs to the module. For example, in a system with multiple security domains, the crypto module will likely have one or more secure access points and one or more unsecure access points. Therefore, channels are communication paths that can be characterized by the crypto module providing the service and the access points, called endpoints, which a client uses to interface with the module.

The IRSS API makes no assumptions about where the inputs and outputs of a channel exist. This allows the definition for an endpoint to be platform defined. For example, one could choose to use endpoints for each HW interface. Alternatively, one could choose to use endpoints for each API instance. In Figure 9, a *CryptographicChannel* would naturally have both the plain text (PT) and cipher text (CT) endpoints connected to the single WF module, while in Figure 10 and Figure 11 the PT and CT endpoints would be connected to the secure and unsecure WF modules respectively.

Although similar to other channels in their characteristics, cryptographic channels and TRANSEC channels have some behavioral differences that distinguish them from other channels. In particular, cryptographic and TRANSEC channels allow clients to use the channel with different configurations. Upon creation, a waveform specifies all the potential cryptographic applications (discussed in the following section) that could be used on the channel. The security subsystem uses this Cryptographic Applications (CA) list to allocate cryptographic resources for the channel. Waveforms then add one or more detailed configurations to the channel after channel creation. Prior to using the channel, a waveform activates a specific configuration on that channel. If a waveform desires a different configuration, the waveform only needs to activate the new configuration. This allows a waveform to swap between configuration on the fly without having to create and destroy channels. There are some caveats to this process though. Since a single set of cryptographic resources are allocated to the channel upon creation, changing a configuration for the channel will cause the CSS to loss any cryptographic state for the previous configuration. If a waveform wants to swap between two different configurations without losing state (for example, to resume a previously started transformation), it must create two separate channels, each allocated with its own set of cryptographic resources.

### 4.2 Cryptographic Applications

While this IRSS strives to provide a standardized framework for security-based operations, being an open international standard, it is impractical to standardize specific cryptographic manipulation and protocols. To address this, the concept of cryptographic applications was created. From an IRSS specification standpoint, CAs are responsible for the specific algorithms and/or protocols for Cryptographic, Transec and Protocol channels. A CA is generically specified by ID only, and accessed by a waveform through a set of standard operations which themselves employ a combination of standard and generic (opaque) parameters. In this way, the overall flow, management and use of algorithms and data handling is done in a common way, while allowing specialized use and configuration.

## 5. STREAMING WAVEFORM APPLICATIONS

The CryptographicChannel abstraction supports two distinct protocols – one for streaming and one for packets. Streaming operation is supported using the TransformStream() operation. Streams have traditionally been employed by non-networking, circuit-switched legacy waveforms, but are also used to encrypt / decrypt files and other non-packet traffic. In a generalized stream, a message is processed across multiple calls to the IRSS, yet handled as an overall entity with intermediate state preserved in the cryptographic application. Typically such streams flow in real-time, with overall message length not being known by the IRSS in advance.

When processing a message within the context of a stream, a given call to TransformStream() can represent a

start-of-message (SOM), middle, or end-of-message (EOM) packet[4], as indicated by passed SOM and EOM boolean parameters. In the typical case, the CSS cryptographic application will, after encrypting the payload, prepend a cryptographic preamble to the first (SOM) packet in a stream. Additionally, it may optionally add a postamble to the EOM packet, and/or intersperse additional framing information in mid-message, as required per the specifics of the employed cryptographic algorithm configured for the channel. When a message is active, the cryptographic algorithm usually needs to maintain a "context" between calls so that subsequent packets can continue advancing the crypto state.

Data flow, either for encryption or decryption, flows from one endpoint to another per the configuration made when the CryptographicChannel was created. Simple, non-simultaneous use of streams is straightforward, as exemplified by the following transmit scenario:

a) The plaintext-side waveform (or platform) creates a channel using CreateCryptographicChannel(). This includes the specification of the plaintext and ciphertext endpoints.
b) One or more configurations are added to the channel using AddCryptographicConfiguration(). This selects a specific CA, key and configuration to be used.
c) A configuration is activated using ActivateConfiguration(). This selects a specific configuration for use.
d) A sequence of calls to TransformPacket() is made, with the first call in a message marked with SOM, and the last marked with EOM.
e) Subsequent messages can be passed without need to destroy / create a channel by repeating step d for each new message.
f) Finally DestroyChannel() is called when the channel is no longer required.

Packets passed from the waveform consist of a pair of payload data and associated in-band bypass information. In keeping with the opaque handling of cryptographic algorithms, the content and handling of these fields is not specified in the IRSS API, but rather deferred to a separate algorithm usage specification. In this way, any combination of initialization information, live data, bypass data or algorithm control data can be passed to the CA, and can be interpreted in a context sensitive fashion. Furthermore, as the data is processed by the CA and sent back to the waveform (using the CryptographicConsumer interface), the usage of these fields is similarly unspecified. To be used in a portable fashion, the CA-specific field usage will need to be specified by the entity specifying the CA, but this is done outside of the IRSS API.

[4] In this context, packet refers to a collection of information presented in an API call, not a networking packet.

In some cases, such as when TDMA waveforms such as Mil-Std-188-183 [6] are used, it is necessary to process multiple streams of traffic in parallel, often with overlapping message lifetimes. As a given channel only has a single cryptographic context (contexts are not stored as part of a CryptographicConfiguration), to do this a waveform must create multiple channels, with each channel dedicated to a given stream. Furthermore, due to the dynamic nature of these waveforms, the stream characteristics, or even the number of required simultaneous steams cannot always be known in advance. To support this, the IRSS API supports dynamic creation and destruction of such channels, as usage scenarios are typically not known a priori, and cryptographic resources are limited in some implementations. An example is shown in Figure 12 illustrating such dynamic usage.

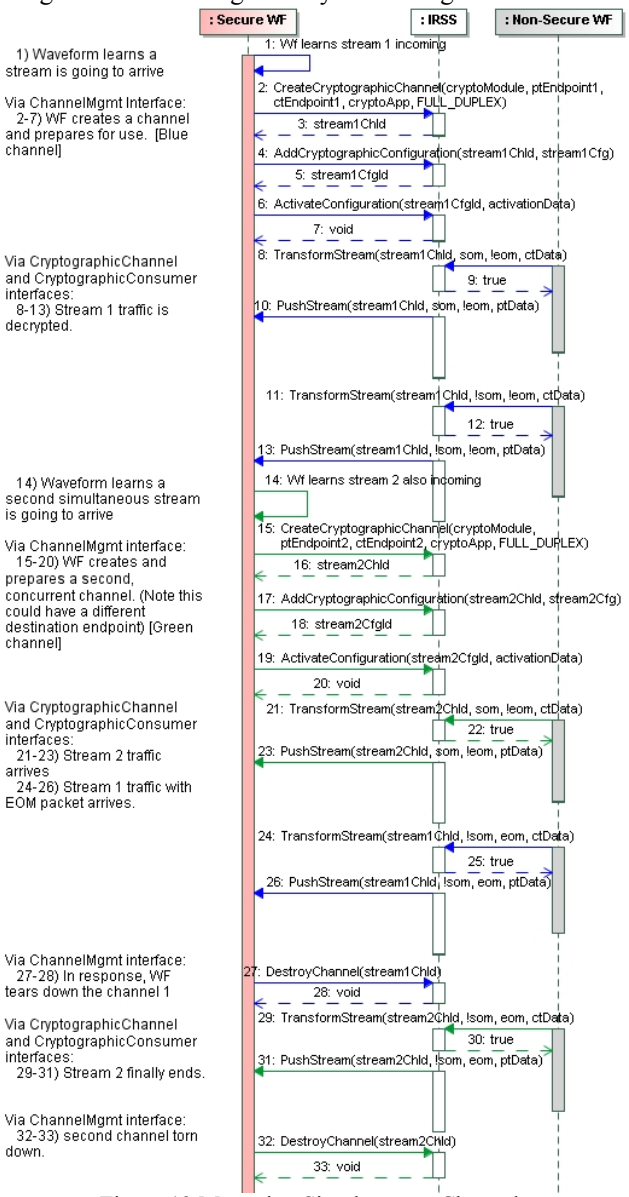


Figure 12 Managing Simultaneous Channels

## 6. NETWORKING WAVEFORM APPLICATIONS

Unlike legacy waveforms, modern waveforms typically support wireless networking protocols. These waveforms are characterized by passing packets of data (i.e. TCP/IP traffic) over-the-air as part of a larger networked system. Transformations of user data in these waveforms will likely involve asymmetric key protocols that establish a key pair for each destination of the network. Since waveforms could define their own key generation protocols, the IRSS API needs to define a generic set of interfaces to support these protocols. In addition, networking waveforms are characterized by high throughput rates. Defining efficient APIs is essential to meeting throughput requirements, especially in size, weight, and power (SWAP) constrained systems.

To develop APIs that support these networking waveforms, the IRSS API working group analyzed two common networking use cases, IPsec ([7] and [8]) and Transport Layer Security (TLS) ([9]). The IPsec protocol is integrated with, or sits below, the IP layer of the networking stacks, whereas the TLS protocol sits above the TCP/UDP layer of the networking stacks. When analyzed, these protocols employ some common security capabilities, but then each protocol also has its own needs for interacting with the security subsystem. Common needs include I&A capabilities such as generating hashes or MACs and using certificates to sign data and verify the signature of signed data. Protocol unique needs include command and response messaging for key derivation functions and other support messaging. The IRSS API working group recognized these common traits and differences and factored this into the definition of the security APIs.

### 6.1. Integrity and Authentication Support

The common traits exhibited by the networking protocols generally revolved around the I&A services required by the protocols. When analyzing the I&A needs, the IRSS API working group identified an aspect of these capabilities that they all shared. In general the capabilities required that the waveform pass data to the security subsystem (for example, to generate a hash, or to compute a signature) and then retrieve the result of the operation from the security subsystem. This led to the development of I&A channels. Like other security channels, clients create and configure I&A channels using the *ChannelMgmt* interface. The common support needed by all the I&A channels is factored out into the I&A *Channel* base interface. This interface defines a generic mechanism for passing data to the security subsystem using octet sequences. The interfaces derived from the I&A *Channel* interface extend the base interface by adding the service unique mechanism for retrieving the result of the I&A operation. For example the *HashChannel* interface extends the base I&A *Channel* interface by adding support for retrieving the results of the hash operation.

Basic usage of the I&A channels involves several steps as detailed in the following text and shown in Figure 13 (for hash channels):

a) A client first creates an I&A channel (e.g. a hash channel) using the *ChannelMgmt* interface. The create operation returns a channel ID for the client to use for future requests on that channel.
b) Using the specific I&A channel interface (e.g. *HashChannel*), the client determines the maximum data size for the channel. This data size allows for platform specific customizations of their underlying transport layer to the CSS.
c) The client then pushes the data to be processed, using the I&A channel interface, to the IRSS in octet sequences. These octet sequences should not exceed the maximum data size as determined in the previous step.
d) Once all the data has been pushed, the client can obtain the results of the operation using the specific methods found in the derived I&A classes (e.g. GetHash).

Note that channel creation allows the client to configure the I&A channel and allows the CSS to allocate cryptographic resources for the channel.

In addition to the I&A channel interfaces, some of the I&A services require certificate management support. In particular, networking security protocol require access to certificates, stored within the cryptographic subsystem, and validation of certificates received from a peer system. The *CertificateMgmt* interface defines an interface that allows the security subsystem to provide these services. In particular, it provides methods for retrieving and validating certificates.

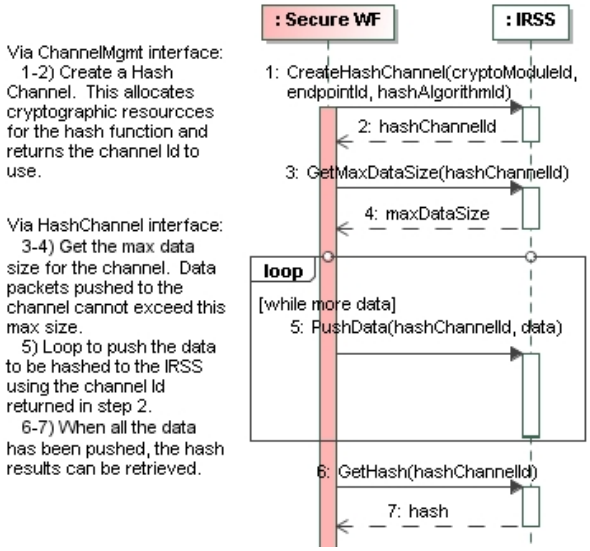

Figure 13 Example I&A Channel Usage

### 6.2. Protocol Message Exchange Support

Unlike the common traits, each networking security protocol will likely contain command and response messaging with the security subsystem that is unique to the protocol. The IRSS API has to allow for this uniqueness and also has to provide generic methods that allow growth for future networking security protocols. To do this, the IRSS API defines the *Channel* and *Consumer* interfaces within the *Protocol* module. These interfaces define generic mechanisms that allow a waveform to exchange messages with the security subsystem. As with other channels, clients create protocol channels using the *ChannelMgmt* interface, which returns the channel identifier that the client uses to access the protocol. Clients send commands to the security subsystem using the *Channel* interface, and receive responses from the security subsystem by providing the *Consumer* interface. These interfaces define methods that pass an opaque octet sequence that encapsulates the command or response and their associated parameters. Appendices to the IRSS API specification will define the unique command and response messages needed by each protocol.

To demonstrate the usage of the protocol interfaces, the following steps, and the accompanying Figure 14, depict how the *Channel* and *Consumer* interfaces might be used to request an IKE key exchange as part of the IPSec protocol:

a) A client first creates a protocol channel using the *ChannelMgmt* interface. Channel creation allows the CSS to initialize the protocol (e.g. IPSec) for that channel. The create operation returns a channel ID for the client and the IRSS to use for future exchanges on that channel.
b) The client requests the initiation of an IKE session by sending a StartIkeSession message to the IRSS. This message establishes the Diffie-Hellman group number to use for the session.
c) The IRSS computes the Diffie-Hellman value to use in the key exchange with the IKE peer and returns it to the client, along with the initiator's nonce and a session Id, in a status message. The session Id allows multiple IKE sessions to be in progress simultaneously.
d) The client then exchanges key parameters with the remote IKE peer.
e) The client sends the key parameters it received from the remote IKE peer to the IRSS using a SetKeyParameters message. The session ID identifies that this message applies to the session started in step b above.
f) The client then requests that the IRSS derive the keys using the previously generated and configured key parameters.
g) Lastly, the IRSS sends the IDs for the generated keys to the client using a status message.

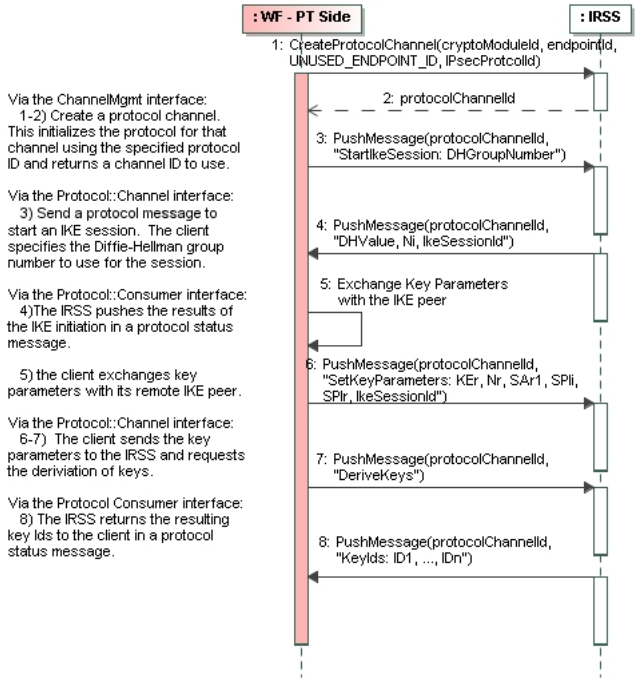


Figure 14 Example Protocol Interface Usage for IPSec

Note that the messages exchanged with the IRSS, as well as the semantics of using those exchanges, are protocol specific and will be different for each protocol that the security system supports.

### 6.3. Information Security Support

With I&A interfaces and generic messaging protocol interfaces, the last pieces needed by networking waveforms are the information security interfaces defined in the *Infosec* module. Critical to these is an efficient API that supports the transformation needs of the waveform. This API must be efficient to support the high throughput requirements of the waveforms. Waveforms utilize the transformation interfaces to encrypt and decrypt packets en route to a destination. As required by the SCA, waveforms utilize this API through a Common Object Request Broker Architecture (CORBA) interface. However, CORBA adds overhead with each invocation of the methods defined by an interface. To minimize this overhead, the *CryptographicChannel* interface allows waveforms to bundle multiple packets with each transformation request into a *PacketSequence*. Through smart management of packets, waveforms can minimize the CORBA overhead incurred when utilizing the transformation interfaces. Since the *CryptographicChannel* interface is another channel to the security subsystem, waveforms create and configure it through the *ChannelMgmt* interface.

As previously discussed, *CryptographicChannel*s are managed slightly differently from other channels. When created, they do not have a complete configuration to use

them yet. Instead, waveforms add and activate configurations after channel creation. The following steps, and the diagram shown in Figure 15, demonstrate how *CryptographicChannel*s are utilized:

a) A client creates a cryptographic channel by specifying the potential cryptographic algorithms that might be used on that channel. This allocates the cryptographic resources for that channel and returns a channel Id to associate with that channel.
b) The client adds one or more specific configurations to the channel. Each configuration is associated with a unique configuration Id.
c) The client activates a specific configuration using the unique configuration Id obtained in the previous step.
d) The client retrieves the max packet and payload sizes. This allows for platform specific differences in the transport layer to the CSS.
e) The client requests the IRSS to transform a payload. A payload consists of multiple packets. Each packet in the payload cannot exceed the max packet size. In addition, the entire payload cannot exceed the max payload size. The IRSS returns a flag to the client to indicate whether or not more space is available for another payload.
f) After transforming the data using the current active configuration, the IRSS pushes the transformed payload back to the waveform.

Note that the type of transformation (encryption vs. decryption) depends on the API instance used to request the transformation.

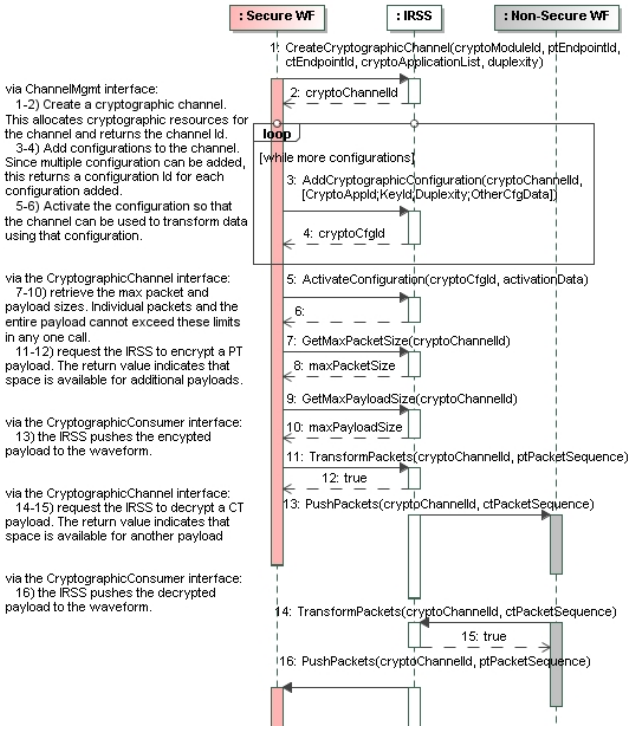


Figure 15 Example CryptographicChannel Usage

The last service that a networking waveform might require is support for TRANSEC operations. The *TransecChannel* interface defines the methods that provide this support. In particular, it defines methods for applying and removing TRANSEC cover, and methods for generating keystream that allows a waveform to manage its own TRANSEC cover. As with all channels, TRANSEC channels are created and configured using the *ChannelMgmt* interface. Like *CryptographicChannel*s, *TransecChannel*s require the client to add and activate configurations prior to use.

## 7. CONCLUSIONS

The SCA establishes a new paradigm for software definable radio systems by defining a framework for the system operating environment. However, to promote portability, waveform developers need standardized interfaces to the platform components that they utilize. After identifying a lack of a standardized interface for the radio security service, WInnF has endeavored to create a standard API for the security service called the International Radio Security Service API. They based development of this API on existing needs for streaming and networking waveforms. Due to the size of the security services required, and to promote LPP practices, WInnF has defined this API in several function grouping found in several IDL modules. Additionally, WInnF used opaque algorithm techniques to allow for adaptation of the API to future protocols and algorithms. Lastly, WInnF understands the need to support different platform topologies and accounted for this in the definition of the interfaces. With these considerations, WInnF has produced a viable security service API applicable to the broader international software definable radio community.

## REFERENCES


[1] Modular Software-programmable Radio Consortium, *Software Communications Architecture Specification*, MSRC-5000SCA, v1.0, May 17, 2000

[2] Modular Software-programmable Radio Consortium, *Software Communications Architecture Specification*, MSRC-5000SCA, v2.2, November 17, 2001

[3] Modular Software-programmable Radio Consortium, *Application Program Interface Supplement to the Software Communications Architecture Specification*, MSRC-5000API, v1.1, November 17, 2001

[4] Modular Software-programmable Radio Consortium, *Security Supplement to the Software Communications Architecture Specification*, MSRC-5000SEC, v1.1, November 17, 2001

[5] D. Lanz, L. Novikov, *Common Interface to Cryptographic Modules (CICM)*, Internet Engineering Task Force, Internet-Draft, January 7, 2011

[6] Department of Defense Interface Standard, *Interoperability Standard for 25-KHz TMDA/DAMA Terminal Waveform,* MIL-STD-188-183A, March 20, 1998

[7] S. Kent, K. Seo, *Security Architecture for the Internet Protocol*, RFC 4301, December, 2005
[8] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC 5996, September, 2010
[9] T. Dierks, K. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, August, 2008