DESIGN OF SECURITY ARCHITECTURE FOR SDR SYSTEM

Mun Gi Kim, Byung Ho Rhee
Graduate School of Information and Communications, Hanyang University
17 Haengdang-dong, Seongdong-gu, Seoul, 133-791, KOREA
Phone: +82-2-2296-0391, Fax: +82-2-2299-1886
E-mail: clevermg@nate.com, bhrhee@hanyang.ac.kr

ABSTRACT

SDR(Software Define Radio) will give future users a number of benefits like global roaming, multi mode, multi band, and multi standard. It will also offer complete programmability and reconfigurability to both multi mode and multi functional communication terminal and network nodes. Also, SDR is expected to solve the compatibility problem among various mobile communication standards so that people can use the same device for different wireless network. If theses mobile communication environment is constructed, integrity and confidentiality of data and Terminal authentication become very important. Also, Mutually authentication and security formality problem are important for nodes. Therefore, In this paper propose authentication scenario and Transmission security for software download using PKC(Public Key Certificate) and AC(Attribute Certificate).

1. INTRODUCTION

SDR is new radio technology of mobile communication. SDR Terminals aim to be able to dynamically reconfigure the structure of a wireless device. SDR most of communication function technology that use hardware that reconfiguration is available to software be SDR is elements of a wireless network whose operation modes and parameters can be changed or augmented post manufacturing via software. If such communication environment becomes available, reconfigure software to use in Domain that communication protocol is different without necessity to change terminal receiving download, grovel communication becomes available. Research about SDR System, Much of researches on next generation mobile communication system are being carried out in SDRForum, ITU-R, 3GPP and 3GPP2 etc. Research on main function of 4G System (SDR) and standardization of SDR technology is being progressed in SDRForum.

In this paper, describe about necessary security authentication formality and Software access control method for software download, Construct trust chain mutually using PKC (Public Key Certificate) and protect software resources from attack of interception, Main-in-middle etc from unlawfulness user's access, Restrict Terminal access extent using AC for trustability and safety of system. finally, propose about transmission security using random key for data stability.

2. ABOUT RESEARCH

2.1. SDR System Security Requirements

SDR is expected to solve the compatibility problem among various mobile communication standards so that people can use the same device for different mobile environment. If theses mobile communication environment are constructed, integrity and confidentiality of data and between terminal and service provider or each network server authentication become very important. Contents to propose in this paper are as following [2].

- Available security algorithm negotiation

Terminal moves to Visit Domain and sends service request signal. This time, Service request signal includes terminal profile to the security management server. The profile is information about security algorithm that terminal is using. BS transmits profile that sends to Authentication Server(AS).

AS confirms version information and available algorithm. Thereafter, All communication use security algorithm that is decided through mutual negotiation.

- Authentication & Access Control

Authentication and Access control need to satisfy various use field of mobile terminal. Authentication is required to allow service use that offer within domain, and Access Control need to restrict user's service access extent. Access Control directive in the PKC determines access to the service provider server. Therefore, service allow or deny can[1].

- Data encryption

Encryption is required for data integrity and confidentiality. AS creates and supplies encryption key for terminal. This key is used for encryption communication between end points. Encryption key distribution uses PKI Based.

[1] This work was supported by HY-SDR Research Center at Hanyang University, Seoul, Korea, under the ITRC Program of MIC, Korea.

The paper will present the details on security architecture, about method of data encryption and authentication, certificate managements and creates and security algorithm negotiation to use in SDR system.

2.2. Access Control method for SDR Terminal

Access control prevents unlawfulness access of Software, and offer integrity, confidentiality and availability of software service. Access competence is established in Terminal's state. Access control manage technology is divided by Identity-Based Policy, Rule-Based Policy, Role-Based Policy. Access control of Identity-Based Policy consists on subject or sub group's position, Rule-Based Policy consists based on rule that is established on permission grade for object. Role-Based Policy is access control technology that use together with Identity-Based and Rule-Based Policy. In this paper, Access control of RBAC (Role-Based Access Control) base is achieved [4]. Establish access competence to RBAC based. It can divide by Core RBAC and Hierarch RBAC.

2.3. Certificate management

2.3.1. X.509v3 Certificate Fields

Certificates may be used in a wide range of applications and environments covering a broad spectrum of interoperability goals and a broader spectrum of operational and assurance requirements. In particular, the emphasis will be on supporting the use of X.509 v3 certificates for informal Internet electronic mail, IPsec, and WWW applications. The X.509 v3 certificate basic syntax is as follows. For signature calculation, the certificate is encoded using the ASN.1 distinguished encoding rules (DER) [X.208]. ASN.1 DER encoding is a tag, length, value encoding system for each element [1].

```
Certificate::= SEQUENCE {
      tbsCertificate      TBSCertificate,
      signatureAlgorithm  AlgorithmIdentifier,
      signatureValue      BIT STRING  }

  TBSCertificate ::= SEQUENCE {
      Version        [0] EXPLICIT Version DEFAULT v1,
      SerialNumber       CertificateSerialNumber,
      Signature          AlgorithmIdentifier,
      Issuer             Name,
      Validity           Validity,
      Subject            Name,
      subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID [1]    IMPLICIT    UniqueIdentifier
      OPTIONAL,
-- If present, version shall be v2 or v3 subjectUniqueID
[2] IMPLICIT UniqueIdentifier OPTIONAL,
 -- If present, version shall be v2 or v3
extensions     [3]  EXPLICIT Extensions OPTIONAL
  -- If present, version shall be v3
       }
```

2.3.2. X.509 Attribute Certificate Definition

ACs may be used in a wide range of applications and environments covering a broad spectrum of interoperability goals and a broader spectrum of operational and assurance requirements [5].

```
AttributeCertificate ::= SEQUENCE {
            acinfo              AttributeCertificateInfo,
            signatureAlgorithm  AlgorithmIdentifier,
            signatureValue      BIT STRING
         }
        AttributeCertificateInfo::= SEQUENCE {
            Version             AttCertVersion -- version is v2,
            Holder              Holder,
            Issuer              AttCertIssuer,
            Signature           AlgorithmIdentifier,
            SerialNumber        CertificateSerialNumber,
            AttrCertValidityPeriod  AttCertValidityPeriod,
            Attributes          SEQUENCE OF Attribute,
I                issuerUniqueID              UniqueIdentifier
OPTIONAL,
            Extensions          Extensions OPTIONAL
         }
```

Attribute certificate composition with form of PKC resemblant. But do not include terminal's public key. Then, terminal's attribute information is as following.

- Service Authentication Information: Terminal ID and Passwd
- Charging Identity: Accounting information
- Group: terminal group
- Role: RoleAuthority, Role of Terminal
- Clearance: information about the AC holder

3. DESIGN OF SECURITY ARCHITECTURE FOR SDR TERMINAL

In this paper, Define following component for Software manage:

- SDR Terminal: Terminal that free communication is avaliable in any mobile communication environment.
- SP (service provider): Development Software for SDR Terminal.
- TMS (Terminal Management Server): TMS authenticate terminal and possess software list that Terminal wants. Then, Terminal connects to TMS to download Software.
- DS (Data Server): DS means server belonging to TMS, and stores Software that receive from SP. Can software

service that terminal wants through OTA, Internet, Sim card.

- CA (Certification Authority) : Issue certificate to Attribute CA and Terminal
- Attribute CA : Decide Terminal's access competence scope and issue Attribute certificate about software access competence.

That is, all nodes issues PKC from CA. If Terminal requests software, all nodes establish relationship of mutual trust using PKC, connecting to TMS and confirm software list and receives download from DS.

3.1. PKC and AC issuance process

SDR Terminal and all servers are issued PKC that can confirm own from CA that plain is authorized. Also, Terminal is issued AC that can confirm own competence from Attribute CA. Such basic process needs for software download.

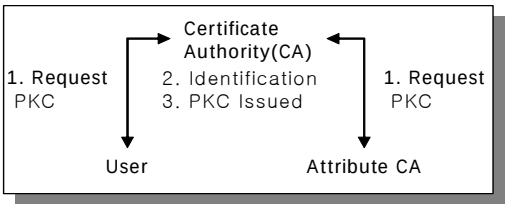


Fig.1. Terminal and Attribute CA PKC Issuance Process

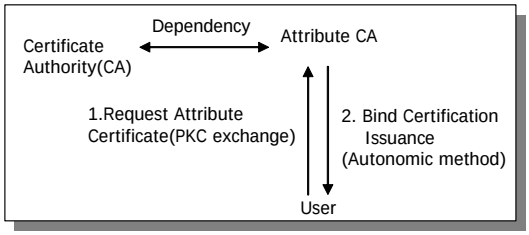


Fig.2. AC Issuance Process

3.2. Access control model using PKC and AC

Attribute certificate division way to deliver attribute certificate in application service can be classified into pull model and push model. When pull model creates attribute certificate, the server searches attribute certificate in repository by way to bulletin attribute certificate that attribute certificate issuance person is issued to directory that search is available in application service. Push model is a way to deliver attribute certificate directly when client connects to server. Such push model and pull model can be utilized electively according to application service environment.

Table. 1. Push model and Pull model's comparison.

	PUSH MODEL
Advantage	-Simplification of certification processing process. -Frugality of certification processing time. -Frugality of additional expense for certificate administration.
Shortcomings	-There are certificate damage and loss danger. - Can be eavesdropping by hacker at certificate transmission process.
Utilization environment	-When outside user's number is too many. -Safe environment in data transmission.

	PULL MODEL
Advantage	-Can reduce certificate damage and loss. -Have softness of attribute information alternation.
Shortcoming	-Additional expense is required for certificate administration. -Is proportional in user number and communication load happens.
Utilization environment	-Is effective in interior user administration. -Security set is effective in weak system environment.

This paper uses push model. Because Push model can bring simplification of certification processing system and certification processing speed elevation that is required. Also, additional expense for user certificate administration of push model is not required. This picture is describes access control process; Use push method for software download. Certification processing server restricts Terminal's DS access within authorized extent using PKC and Attribute Certificate that user submitted. certificate verification process is as following.

When SDR Terminal came to new domain, Request software to TMS. Then Terminal delivers PKC to TMS, and it is establish relationship of mutual trust. If Terminal is authenticated, TMS is request terminal Attribute Certification. Decide DS's access competence using AC. Access competence establishment is achieved through ACL(Access Control List). ACL list decides Terminal's access competence using RBAC. If all certification processes and access competence examination are completed, Terminal downloads software from DS.

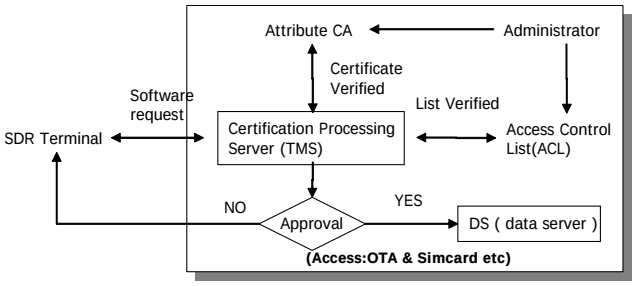


Fig.3. Software request/response process

3.3. Software transmission security method

In this chapter explains about transmission encryption method. Terminal approaches to DS using PKC and AC for software download. Then, Need transmission security to prevent security threats that happen between nodes. Transmission security algorithm uses Symmetric Cipher System and Asymmetric Cryptosystem. Terminal and TMS exchange PKC for relationship of mutual trust. Terminal and TMS confirm each other's certificate. Certification is achieved through CA's signature value. If relationship of mutual trust is formed, TMS creates and shares session key for data encryption.

< Definition > Symbol that is used for encryption

K_{TP} : Terminal public key K_{TS} : Terminal private key

K_{TMS_p} : TMS public key K_{TMS_s} : TMS private key

K_{DS_p} : DS public key K_{DS_s} : DS private key

$K_{session}$: Session Key that server creates

$E_{K_{TP}}(K_{session})$: Session Key that encode by public key of terminal

$E_{K_{DS_p}}(K_{session})$: Session Key that encode by public key of DS

$E_{K_{TMS_s}}(K_{session})$: $K_{session}$ that encode by K_{TMS_s}

$E_{K_{TS}}(E_{K_{TMS_p}}(K_{session}))$: Key that encrypt encoded key to K_{TS}

$K_{session}(R_{m2})$: Terminal request message that is encoded by $K_{session}$

$K_{session}(M_{m2})$: TMS response message that is encoded by $K_{session}$

S_{m2} : Digital signature message that is encoded by K_{TP}

$MD(M)$: Message digest that use Hash algorithm

$MD(M)^*$: Terminal creates Message digest that to compare with $MD(M)$ value.

$E_{K_{DS_s}}[MD(M)]$: Digital Signature that encrypt $MD(M)$ by K_{DS_s}

3.3.1. Session Key shared method between Terminal and TMS

All request messages, response message and software between Nodes are encoded using session key. Session key is random number, Session key can creates any one. But, Session key creation does TMS in this paper for to minimize transmission protocol. and Created session key is transmit to Terminal using Terminal's public key. Terminal is decrypt using Terminal's private key. So Terminal acquire session key.

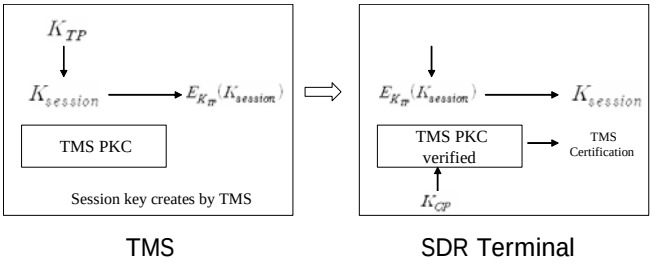


Fig.4. TMS and Terminal session key shared method

3.3.2. Encryption method between Terminal and TMS

Send-receive data is encoded using session key. TMS has role that certify Terminal, and software management for Terminal. TMS decides software through Terminal profile message, Software to supply can know through Terminal Profile message that Terminal sends.

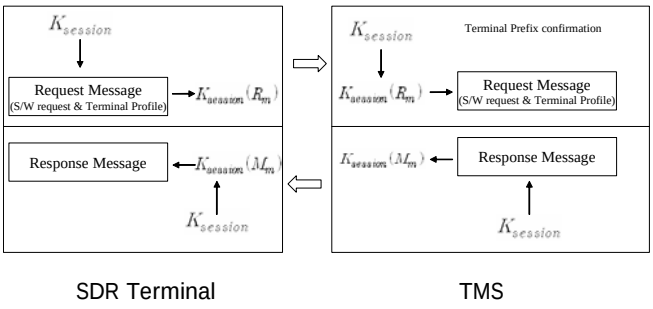


Fig.5. Encryption method between TMS and Terminal

3.3.3. DS Transmission Security

1) Encryption method to use Session key

DS receives result that Terminal certified and receives Session Key for data encryption from TMS. All data are encoded using session key for transmission security. Then, download software to Terminal is decided by TMS.

Next picture is explaining that DS receives session key from TMS and transmit Software to Terminal.

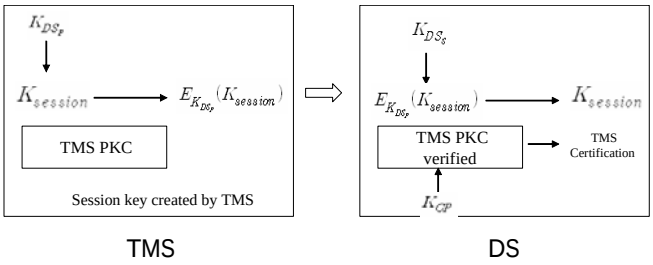


Fig.6. TMS and DS session key shared

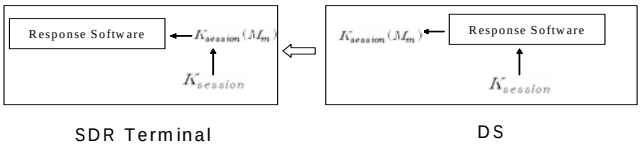


Fig.7. Data transmission between DS and Terminal

2) Encryption method including node's signature function
The following is method to add Digital Signature for data integrity.

DS encrypts software by session key ($K_{session}(R_{m2})$), and makes message digest ($MD(M)$) using Hash algorithm. And creates Digital Signature ($E_{K_{DS_p}}[MD(M)]$) using DS's private key.

Digital Signature encrypts by Terminal's public key. Such method can support data integrity. After all data receive, through next process, Terminal verifies integrity of data that received.

- Terminal does decrypt $K_{session}(R_{m2})$ using session key ($K_{session}$). And makes message digest ($MD(M)^*$).
- Terminal is decrypt digital signature ($E_{K_{DS_p}}[MD(M)]$) using DS's public key. Acquire $MD(M)$ and compares with $MD(M)^*$. If value is equal, accepts data. If value is not equal, abandons data.

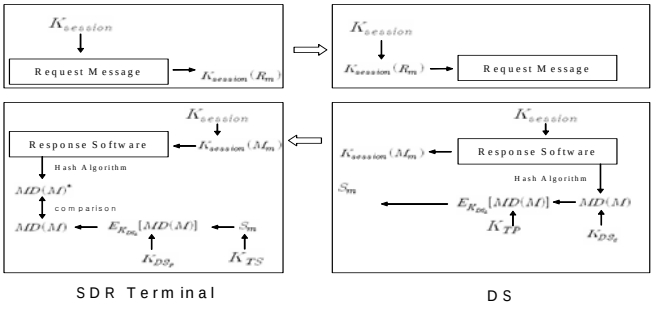


Fig.8. Encryption method including node's signature function

4. CONCLUSION

In this paper described about transmission security using session key, mutually authentication using PKC and AC and data integrity. Mechanism that proposes in this paper can protect Terminal from virus or bugs through safe Software download, can protect away attack about data forgery and alteration from attacker and can protect Terminal and servers from data interception and Main-in-Middle attack. Proposed mechanism is shortcoming that system resources should be got to apply in present mobile communication. But When consider SDR Terminal system that is developed at the future, it may get into safe download mechanism if minimum size of certificate and simplify verification process.

At future, research about SPKI may have to be gone for simplification of certification system, and research for software grade classification may have to be gone for Terminal access control mechanism.

[1] R. Housley, W. Polk, W. Ford and D. Solo "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" *RFC 3280,* April 2002.

[2] SDRForum, *"Security Consideration for Operational Software for Software Defined Radio Devices in a Commercial Wireless Domain" DL-SIN Document SDRF-04-!-0010-V0.0 Ballot version, 27 October 2004.*

[3] R. Shirey, *"Internet Security Glossary" RFC 2828, May 2000.*

[4] David R. Ferraiolo, Janet A. Cugini and D. Richard Kuhn, *"Role-Based Access Control (RBAC); Features and Motivations" Proceedings of the 11th Annual Computer Security Application Conferences, December 1995.*

[5] S. Farrell and R. Housley, *"An Internet Attribute Certificate Profile for Authorization" RFC 3281, April 2002.*

[6] Kate Cook and Carlos Martinez, *"Developing end user and operator requirements for software reconfigurable radio" IST Summit 2001.*

DYNAMIC POLICY ENFORCEMENT FOR SOFTWARE DEFINED RADIO

Patrick Flanigan, (Security Engineer, NCSA, University of Illinois Urbana-Champaign, flans@ncsa.uiuc.edu), Von Welch, (Senior Security Engineer, NCSA, University of Illinois Urbana-Champaign, vwelch@ncsa.uiuc.edu) Meenal Pant, (Software Engineer, NCSA, University of Illinois Urbana-Champaign, mpant@ncsa.uiuc.edu)

ABSTRACT

Our research analyzes security policy enforcement issues inherent to handheld Software Defined Radio (SDR) devices. We have developed an abstraction for Dynamic Policy Enforcement (DPE) for a SDR system which consists of three distinct modules that monitor changes in external conditions, validate system configuration based on those conditions and a given policy, and implement changes to ensure policy compliance. In order to demonstrate the viability of our system, we created a prototype that implements the roles and responsibilities of our abstraction in conjunction with a prototype SDR system previously developed by NCSA that is based on the GNU SDR software.

1. INTRODUCTION AND MOTIVATION

Typically, standard radio devices have been built for extremely specific functions, limited by the use of narrow bandwidths and rigid hardware specifications. Software Defined Radio (SDR) allows for functionality previously statically-cast in hardware to be implemented in software. The power of SDR lies in the ability to dynamically reconfigure its functionality by changing flexible software. With this flexibility, we achieve tremendous advantages over hardware-only platforms because software can be developed to perform the complex tasks and dynamically updated, altered or even removed based on changing conditions, users or policy. However, with these gains in flexibility, security policy enforcement becomes a major concern. Our work is focused on the design and implementation of a Dynamic Policy Enforcement (DPE) for SDR security.

There are a number of distinct security issues with SDR. The focus of each issue is dependent upon a balance between the required flexibility and the level of security that is desired. There has been a fair amount of prior work focused on allowing for secure dynamic download and installation of software into a SDR and the protection of the base SDR software from malicious code. Our focus is at a higher level of abstraction – the implementation of secure and dynamic policy enforcement for SDR to ensure that the functional pieces of software deployed adhere to policy dependant on the user of the SDR and the conditions in which the SDR is being used.

SDR policy enforcement must take into account dynamically changing users, conditions, environments and needs. In order to decide if a given configuration, by which we mean combination of software in use and parameters such as broadcast frequency, protocol and power, there are a number of factors that effect how the SDR should behave:

- **Who is holding the SDR?** What is the role of the holder of an SDR device? Is it, for example, an average citizen, a responder, a member of law enforcement, or the commander of the response?
- **What are the environmental conditions?** Is it a normal day or is there a condition alert or is there an emergency response going on in the immediate area?
- **What policies are in effect?** Policies would seem to be more static than the previous factors, but may vary in time or as the device moves from one region to another, changing administrative jurisdiction.

It is key to notice that, in particular with the first two criteria, these may change dynamically and outside the control of the SDR device. This requires policy enforcement to not only consider requested changes (e.g. in broadcast parameters or software installation), but factors that change outside the device's control (e.g. who is holding the device or the state of emergency).

Consider the scenario of first responders from an emergency agency (e.g. fire, medical, law enforcement) using SDR-based handheld radios during a response. When the emergency is declared, the first effect might be that average citizens holding SDR-based cell phones or two-way radios would be severely limited in how they could use those devices in order to preserve bandwidth for responders. The responder's devices however, should remain fully functional, or even increase in functionality, allowing them to access normally private channels to facilitate cross-agency communication or to allow for (or even require) encrypted private communication.

And, if under some unusual circumstance, a responder were to use a device belonging to an ordinary citizen, that device should readjust, granting as much access to that responder as permitted by its policy and its implementation.

In this scenario, it is clear that we have many possibilities for authorization and configuration. There could be any number of users with different levels of access, and these users could change at any time as the radio is passed from person-to-person. A change in the alert condition may require a change in basic functionality, for example, reducing functionality to preserve spectrum space for critical services. Different modules for encryption may need to be changed on the fly for higher security. And software modules themselves may have specific policy restrictions as well. The intersection of all of these factors can become quite complex. Because of this, a dynamic and reliable policy enforcement solution is required. By creating an abstract security layer that interacts with the SDR application layer, we can isolate and manage these security and configuration needs. This needs to be accomplished at a layer that is independent of the SDR so that we do not compromise the implementation of the radio device.

Our goal was to design and implement an enforcement system that is capable not only of vetting changes prior to their occurrence, e.g. a user requesting a change in frequency or the application of an encryption scheme, but after their occurrence as well, e.g. the radio is dropped and picked up by a different user. This requires not only traditional policy-based authorization gateways that vet requests, but active system monitoring which validates all elements of the system (user, software state, external environment, etc.) against policy and is capable of making changes to ensure policy is enforced in the face of changes outside the control of the system. First, in Section 2, we present our architecture and design of a Dynamic Policy Enforcement for SDR. In Section 3 we describe our implemented prototype to validate our design. In Section 4, we cover the NCSA implementation of SDR. We conclude with a discussion of related work.

2. POLICY ENFORCEMENT ARCHITECTURE

Our Dynamic Policy Enforcement (DPE) system is focused upon preventing intentional or unintentional behavior on the part of the SDR user, which violates policy in regards to the methods of use of the SDR system. We contend that an independent management system composed of three abstract roles can successfully accomplish this. We call these roles the *monitor*, the *implementor* and the *validator*. These roles are implemented as separate modules that intercommunicate.

The monitor is the entry point to the entire system. It detects and handles all changes to or requests for changing the current configuration. This can be done passively by capturing events, either through software or hardware, such as a sensor, or by actively monitoring the activity within the application layer. The monitor passes all events to the implementor.

The role of the implementor is to enact changes to the SDR system, either by servicing requests that are deemed to be valid under the current policy, or in reaction to changes in external environment that have caused the current system configuration to become invalid based on current policy. The implementor communicates with the validator to determine what configurations are permissible under current policy.

The validator contains policy which describes all permissible configurations of the SDR system based on environmental conditions and user attributes. It receives queries from the implementor and responds with the resultant configuration that should be implemented. This configuration may be one requested or may be modified if the a requested configuration is not permissible.

Dynamic configurations can be represented by permutations of the application's components. Some of these components are external because they exist outside the software. A specific person, role or group denotes the current user. The weather is denoted by some well-defined, finite set of possible weather states. The condition is something that is enforced externally to our system entirely, such as an alert status. As a first level of abstraction, we can consider these factors much like the modules of the SDR stack. Let us call the entire set of components and their dependencies our application layer. And it is crucial that this layer is closed. That is, all possibilities are accounted for at any given time. And each of the components is verifiable, so that we are always sure that the component is what it says it is. These components and their configurations can be mapped-out by sets of permutations. We can represent such images of the application layer in a standard, ubiquitous format such as XML. The monitor, implementor and validator can use these images to communicate about configuration decisions. And due to the abstraction of these security policy issues, we are able to concentrate and isolate authorization and module replacement apart from the application itself.

3. POLICY ENFORCEMENT SYSTEM

We now turn to the detailed design of our system. Our work is composed of two distinct systems – the Dynamic Policy Enforcement (DPE) modules and the SDR modules. How these two systems interact is the focus of our research. This section provides an overview of the DPE implementation, while the next is about the NCSA implementation of SDR. Our intention was to build a prototype of the two systems coexisting on a handheld

SDR. For our purposes, it has been sufficient to build them both on a Linux box (Fedora 4) using GnuRadio 2.5. First we will look at the individual components, their roles and how they intercommunicate.

Instead of using sensor mechanisms to provide external requests, we created a web-based interface that allows a user to select a controlled set of parameters. This GUI was built with Python and consists of three dropdown menus that allow the user to select an 'external' request. We decided to use the role of the user, the current alert condition and the weather as typical parameters for this study. Many more variables could have been used, some perhaps more relevant for specific reasons, but our intention was to keep it simple. We wanted to demonstrate functionality – not complexity. We use specific permutations of these parameters as criteria for the security policy decisions. When changes are selected, they are reflected by rendered diagrams showing the resultant configurations of the SDR. We will also follow a typical flow of a request as it is processed by the DPE system.

3.1 Individual Roles and Responsibilities

In Section 2, we discussed a viable abstraction for a dynamic policy enforcement system which could manage and make security policy decisions for SDR. We built modules which implement this abstraction. The module names were shortened for brevity. The names are montor (monitor), imptor (implementor) and valtor (validator). They are C++ modules that use named pipes to communicate with each other using a messaging API developed specifically for this project.

Montor is the entry point and the event sink for the DPE system for external requests and events fired off by the SDR. Its job is to handle requests for configuration changes and to monitor the SDR so that configuration changes can be made in case there is some failure in the current SDR setup. The motivation behind the event monitoring is so that the SDR can be watched to ensure stability and security. We foresee that threats upon a system can be detected as internal configuration changes or requests and they can be inspected as such. This 'monitoring' capability is not within the current implementation of montor, but it is certainly an area that invites further research.

In the Linux workstation version of this system, the GUI provides external requests to montor. The GUI also allows the user to start and stop the DPE system and the SDR. Each is started as a whole using multiple forked processes. This was implemented for demonstrative purposes. In an actual handheld SDR, these requests could be enabled with sensors designed to detect changes that should be handled by DPE. An example is the use of biometrics to determine a user change that may require a change to the SDR configuration.

Imptor is the workhorse for the DPE system. It handles requests that have been picked up by montor, then gets them validated by communicating with valtor, then actually makes the actual SDR configuration changes as needed. It is aware of configuration needs through the use of specific XML-based configuration files. These files contain resource information used to set up the configuration such as module names as well as repositories for module downloads. They are also capable of ensuring that the correct/secure modules are being used for a specific configuration.

The final piece of the Dynamic Policy Enforcement system is valtor. Valtor assumes the role of the validator. It receives requests from imptor, opens and inspects the XML-based policy file, then processes the request with a configuration that is appropriate for the given parameters and sends it back to imptor. The security policy file can be updated 'on the fly' as well. The current policy that is being used is based upon eXtensible Access Control Markup Language (XACML). We will discuss this further in our Related Works section. We wanted to rely upon a standardized way of representing our security policy that could be applied to SDR.

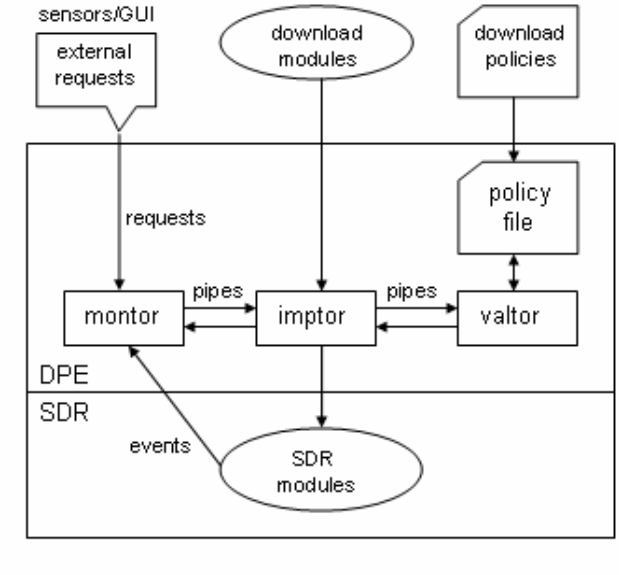


Figure 1: An Overview the Entire System

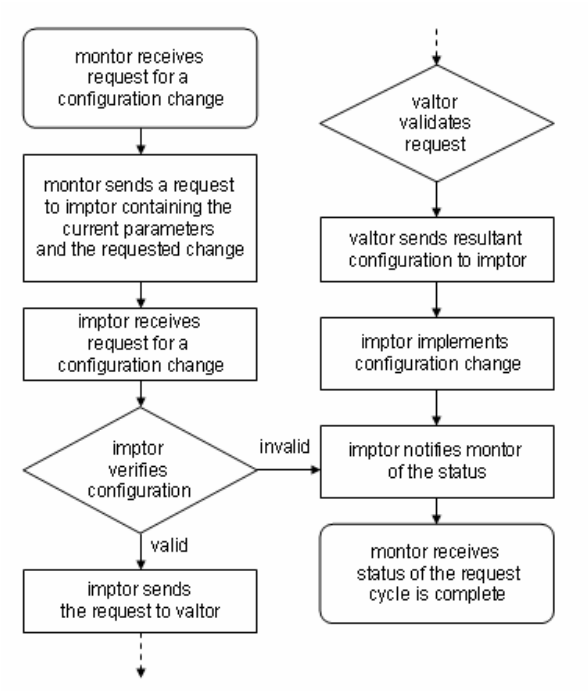
3.2 The Request Data Flow

The primary vision that guided the development of this system is that each module has its own unique role and responsibility. They do not need to know anything about what the other modules are doing. They watch their

message pipes and react, process and reply. In fact, one can shut a module down and the entire DPE system will stop. But if the module is then restarted, the system starts moving again. The entire process is driven by the messaging that is passed through the named pipes. Each module processes its message by reading a pipe, and then completes its role by writing out to another pipe. Figure 2 is a diagram of the flow of a request through the DPE system. The message API between the modules is quite simple. The data is written and read via pipes using a message buffer layer which is mapped into structures which pass the appropriate parameters.

Let's take a quick walk through the diagram. Montor is waiting for something to happen. It receives a configuration change request. In our case, this is very simple because we only have two configurations that are possible. We are only swapping out encryption/decryption modules which reflect the security level given the current parameters – the user, the alert condition and the weather. Montor takes this state information and relays it to imptor. Imptor checks the appropriate configuration file and verifies that it is a valid configuration. If not, imptor replies to montor that it was an invalid configuration request. Nothing is changed. However, if successful, the request gets passed to valtor.

Figure 2: The Request Flow



Valtor receives the request and opens the XML-based security policy file. It checks the permutation of the requested parameters and determines if it is valid for the requested configuration. Valid or invalid, valtor sends a resultant configuration back to imptor. If valid, the requested change is returned. If the policy criteria is invalid, the 'best alternative' configuration is returned. This ensures that even if the security level is not what is required for the request, an appropriate configuration is returned. Thus, the SDR always remains operational. The policy file must be built so that any request will return a valid configuration. It certainly can be flagged as a failed request, but continuous operation without user intervention is a positive alternative to SDR shutdown.

Consider this scenario: the handheld SDR is being used by someone who has a high security clearance. His credentials match the current encryption scheme that is implemented by the SDR configuration. He loses the radio and it is found by someone who has inadequate credentials for the configuration. Montor picked up the change of user, but it has no knowledge of the current configuration. By the time the request gets to valtor, the configuration is passed as the current one, which fails for the new user. A less secure configuration is passed back to imptor and the change is made. The user does not even need to know about the change. It can all be mapped out in the security policy file. This ensures that all final decisions about accessibility are made according to the policy file.

4. NCSA SDR IMPLEMENTATION

In order to demonstrate the viability of our Dynamic Policy Enforcement upon SDR, we have implemented it in conjunction with a SDR system. For our implementation, we are using GnuRadio 2.5 as the underlying software required for implementing a SDR. In order to demonstrate a typical SDR application, we previously developed a reconfigurable software radio 'data stack' that consists of four executables that are inter-connected via UNIX pipes. [3] The data stack is essentially a collection of modules/layers such as source, sink, software defined radio (SDR), *encryptor* and *decryptor* that inter-communicate through a well-defined API. (see Figure 3).

The purpose of this module setup is to provide a data stack that is dynamically reconfigurable. That is, any layer can be replaced at runtime. We can replace modules for security needs such as encryption or replacing modules dependent upon functional needs that vary according to the weather, conditions or the user. And when these modules are swapped, allowable configurations and proper authorization will need to be considered. This requires a standard way of representing all possible configurations of the modules and any other determining factors. We refer to all modules and factors as 'components' of the application layer. These components comprise the SDR.

The impetus behind building such a system is to keep the design fairly simple and well-bounded. Encryption and decryption are somewhat ubiquitous and easy to swap and analyze for a given system. Modifying the SDR behavior in other ways such as bandwidth usage or output power is much more complicated and raises even more security considerations such has regional regulations and standards. Thus, the SDR model for our purposes is intentionally simple and straightforward. We needed a simple model so that we could focus on dynamic policy enforcement without sidestepping into other security issues. If the SDR is built with modules, then our system can be used with it. The layout of the SDR can be specified in the configuration file.

As the name suggests, Source is a data provider, such as a file source or an audio source. Data from a transmitter's Source goes to the intended receiver's Sink, such as a file sink, audio sink etc. The Encryptor module, located at the transmitter, is responsible for data encryption before the data travels through the air interface. This module is implemented as a software object providing a specific encryption scheme, such as Triple DES, AES etc. The Decryptor module, located at the receiver, will decrypt the data before sending it down to the sink. Similar to the Encryptor, this module is also implemented as a software object providing a specific decryption scheme. The SDR module provides a transmit/receive path, filtering and modulation schemes for the data to travel through the air interface.

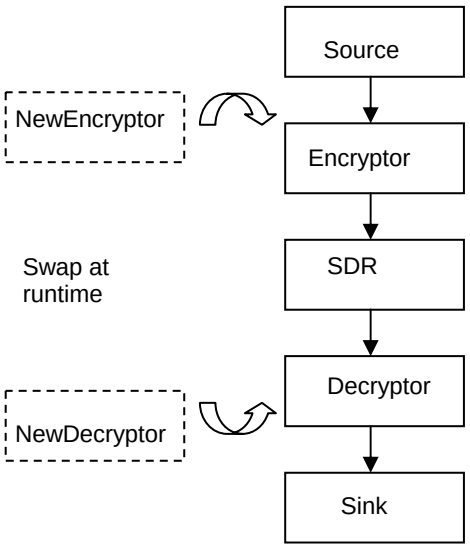


Figure 3: The reconfigurable data stack

4.1. Previous NCSA work with SDR

The reconfigurable data stack was first implemented at NCSA using GnuRadio 0.9 [2], C++ and UNIX pipes. This stack is comprised of Application, Session, Security, Radio Manager and Radio Hardware layers. It is reconfigurable at runtime as well. The modules communicate using named pipes. For a detailed description of the architecture and implementation of this stack please refer to [3].

4.2. Current Implementation

For the current implementation, GnuRadio 2.5, Python 2.3.4 [4] and UNIX sockets [5] are used. Our previous work could not be reused, as the GnuRadio 2.5 code base went through a major implementation change from GnuRadio 0.9. Each of the modules are Python objects. The SDR module is an extension of the GnuRadio 2.5 code. The modulation scheme used by transmit and receive paths is Frequency Shift Keying (FSK). The SDR module receives encrypted data from the Encryptor. This data is then filtered, interpolated and modulated. The signal is then transmitted over the air interface. At the receiver the signal is filtered and demodulated to extract the original data. The modules talk to each other via UNIX sockets. During runtime any of these modules can be swapped out with another similar module seamlessly, based on the command received from the Policy Enforcement front end.

5. RELATED WORK

In this section we briefly review and contrast some related work in the field of SDR security.

The Next Generation (XG) program [6] is developing specifications and concepts related towards using SDR technology for a dynamic redistributable spectrum. Their proposed architecture [7] bears strong similarity to GNU Software Radio-based data stack design, which is not surprising since it also seems to be inspired by the ISO network stack model. The XG group also has a proposed policy language [8] (for which they have a prototype [9]) with implied policy enforcement architecture. This architecture is fairly similar to ours, with a "Policy Conformance Reasoner" corresponding to our validator, an "Accredited Kernel" playing the role of implementor, and the notion of a "Sensor" which partly fills our monitor role. The major differences between the projects are that the XG has spent considerable effort in developing what appears to be a comprehensive policy language and our project has incorporated external environmental conditions besides radio spectrum use, e.g. alert level and device user.

Lam et. al. [10] propose a "Radio Security Module" for validation and lifecycle management of software on a SDR. This work is complementary to the work described in our paper as it serves to validate downloaded software,

while our work strives to manage that software's use under different conditions once it is installed on the SDR.

Hill et. al. [1] have performed a threat analysis on the GNU Software Radio [2] which forms the basis of our prototype implementation. Their work focused on a number of software vulnerabilities within the GNU implementation. These problems include memory access threats and the risks associated with the manipulation of the execution graph. This analysis pointed out some execution weaknesses of the GNU implementation which effect our implementation as well, namely the single address space in which the different software modules run, which in our implementation includes the policy enforcement modules as well. Advancement of our work beyond the prototype phase would need to address this concern. Their work also stressed the need for a strong policy driven configuration that would provide a framework to minimize the risks associated with the programmability of RF parameters. It is crucial that operating constraints be in place so that security policies can be effectively enforced.

6. CONCLUSION

We have presented a possible architecture for dynamic policy enforcement for a SDR system which takes into account dynamic attributes external to the SDR device such as the device user and environmental conditions such as level of alert. Our architecture consists of three main components, which serve to monitor the current system configuration and accept requests for changes to that configuration, validate configuration changes, either requested or externally driven, and then implement changes based on requests or deviation of the configuration from what is valid under policy. To demonstrate the validity of our system, we have prototyped our architecture in conjunction with a GNU Software Radio-based application data stacked previously implemented at NCSA.

We examined at length the problems and constraints that were encountered in this development. As an extension to our findings, we also considered the viewpoint that attacks upon an application\system and internal failure could be seen as changes in behavior that can be detected by the monitor. These could certainly be interpreted as requests for new configurations that could be handled just as safely and easily as we have shown above. Transitioning our focus into software and away from hardware dependency has brought along many inherent security issues. This is seen very clearly with SDR. Our research has been focused upon abstracting these issues out of the application layer and addressing them independently. We have found that secure software systems can be represented in a model that is highly adaptive and configurable. Our prototype provides us with a strong, dynamic security policy enforcement solution for SDR.

7. ACKNOWLEDGEMENTS

This work is funded by the Office of Naval Research through the National Center for Advanced Secure Systems (NCASSR), as was the previous work developing the GNU SDR Radio-based data stack described in [3]. The GNU SDR Radio software base provided the foundation for our project.

8. REFERENCES

[1] R. Hill, S. Myagmar, R. Campbell, **Threat Analysis of GNU Software Radio,** World Wireless Congress (WWC) , May 2005.

[2] http://www.gnu.org/software/gnuradio/

[3] A. Betts, M. Hall, V. Kindratenko, M. Pant, D. Pointer, V. Welch, and P. Zawada, **The GNU Software Radio Transceiver Platform**, Procs of 2004 Software Defined Radio Technical Conference (SDR Forum), Phoenix (AZ), Nov 2004, Vol. C, pp. 41-46.

[4] http://www.python.org/

[5] W.R.Stevens, "Unix Network Programming, Volume 1: Networking APIs - Sockets and XTI", 1997, Prentice Hall PTR

[6] "XG Overview", visited September 29th, 2005, http://www.darpa.mil/ato/programs/xg/overview.html

[7] XG Working Group, "The XG Architectural Framework, "Request for Comments Version 1.0" http://www.darpa.mil/ato/programs/xg/rfc_af.pdf

[8] XG Working Group, "XG Policy Language Framework Request for Comments Version 1.0", http://www.ir.bbn.com/projects/xmac/rfc/rfc-policylang-1.0.pdf

[9] "The BBN XG Projects", visited September 29, 2005. http://www.ir.bbn.com/projects/xmac/pollang.html

[10] Chih Fung Lam, Kei Sakaguchi, Jun-ichi Takada, and Kiyomichi Araki, "Radio Security Module that Enables Global Roaming of SDR Terminal while Complying with Local Radio Regulation," 2003 Fall IEEE Vehicular Technology Conference (VTC 2003 Fall), Oct. 2003 (Orlando, FL, USA).

DESIGN SECURITY WITH WAVEFORMS

Jie Feng
Altera Corporation
101 Innovation Dr
San Jose, CA 95134
(408) 544-6753
jfeng@altera.com

Joel A. Seely
Altera Corporation
101 Innovation Dr
San Jose, CA 95134
(408) 544-8122
jseely@altera.com

ABSTRACT

Military communications applications such as the Joint tactical Radio System (JTRS) are increasingly turning to FPGAs for large portions of their system design. The reasons for this are many, but include the benefits of increased density, functionality, and performance of FPGAs, as well as higher flexibility, lower development costs and risks over ASICs. However, as FPGAs become a more integral part of the leading edge architectural design, replacing ASICs and ASSPs, security of the FPGA design and configuration bitstream is of utmost importance. This paper describes two techniques – configuration bitstream encryption and handshaking tokens – for securing designers' intellectual property (IP) within SRAM-based FPGAs.

1. INTRODUCTION

Military applications are becoming increasingly complex. Major programs such as the Future Combat Systems (FCS) and Joint Tactical Radio System (JTRS) are pushing technological capabilities on all fronts to their limits. The electronics in these systems are relying on programmable logic and FPGAs to provide extreme flexibility at a reasonable cost while not giving up the requisite computational power. For example, secure communication systems are used to connect a variety of airborne, space ground and sea-based military communication networks. They are used in the transmission, processing, recording, monitoring and dissemination functions of a variety such networks, including secure data links. All this functionality requires processing power and reconfigurability.

As FPGAs advance in density, functionality and performance, they are increasingly used in critical military system functions that were traditionally filled by ASICs or ASSPs. However, SRAM-based FPGAs are volatile and require a configuration bitstream to be sent from a flash memory or configuration device to the FPGA at power up. Since this bitstream could be intercepted during transmission, design security in high-performance FPGAs is a concern.

2. TECHNIQUES FOR ENSURING BITSTREAM SECURITY

Two techniques – configuration bitstream encryption and handshaking tokens – can be used for securing intellectual property (IP) within SRAM-based FPGAs. The bitstream encryption is enabled using 128-bit advanced encryption standard (AES) and a non-volatile key. The 128-bit AES key makes it much more secure than data encryption standard (DES - 56-bit key size) and triple DES (112-bit effective key size). The non-volatile key is stored on the FPGA and retains its information when power is off, eliminating the need for unreliable battery backup in harsh military environments. Handshaking tokens is a method whereby the FPGA communicates with a CPLD which includes a non-volatile stored encrypted token. The FPGA design must read this token and have the matching key, otherwise the design will shut down.

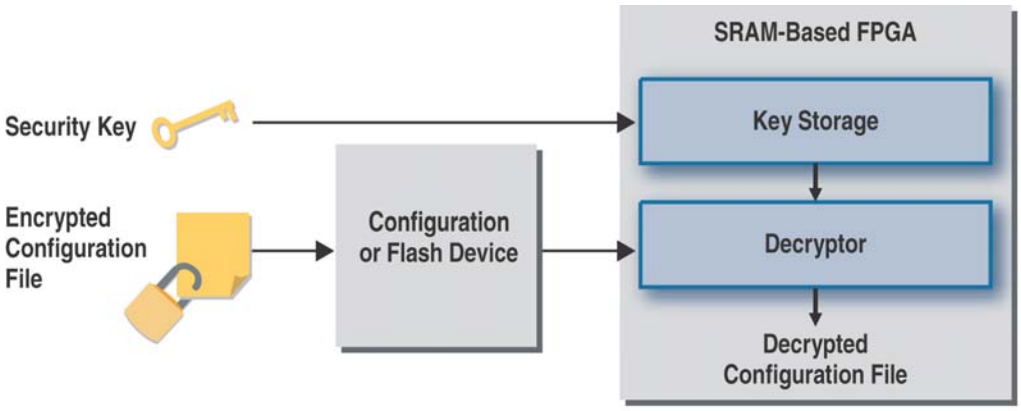


Figure 1: Altera DSP Builder Design Flow

AES comes in three different key sizes: 128-bit, 192-bit, and 256-bit. The longer the key size, the more secure, but also the more processing-intensive and costly. For many applications, 128-bit AES key size is probably the most suitable for both security and efficiency. To understand the level of security, studies have shown that if one could build a machine that could discover a DES key in seconds, then it would take that same machine approximately 149 trillion years to discover a 128-bit AES key.

Security key storage, which can be in either a volatile or non-volatile location, is an important part of overall security. When the key is stored in volatile memory, an external backup battery is required when there is no power to the device. While this solution is quite secure (because the key will likely be lost if someone tries to attack the solution by decapping the device), reliability, especially in military environments, is a major concern. Battery life depends on temperature and moisture levels of the surrounding area. If the battery dies, the key will be lost, and the device becomes unusable and must be sent back to the factory for repair. Also, adding a battery increases overall system cost and requires additional manufacturing steps. The battery needs to be soldered onto the board after the reflow process. The volatile key needs to be programmed into the FPGA after both the FPGA and the battery are on board.

When the key is stored in a non-volatile location, no external battery is required. This method is more reliable, practical and flexible. The key can be stored into the FPGA during regular manufacturing flow, with the FPGA either on-board or off-board. Various security techniques need to be employed to make the key difficult to find.

Because only the encrypted configuration file is physically located in the system with the key stored securely inside the FPGA, even if the configuration bitstream is captured, it cannot be decrypted. Read-back of a decrypted configuration file is not allowed by the FPGA vendors. Further, the encrypted configuration file cannot be interpreted and used to configure another FPGA without the appropriate key, making it very difficult to copy such a design.

Reverse engineering any FPGA design through configuration bitstream is very difficult and time-consuming, even without encryption. For high-density devices, the configuration file could contain millions of bits. Some FPGA vendors' configuration file formats are proprietary and confidential, providing another layer of security. With the addition of configuration bitstream encryption, it may be easier and quicker to build a competitive design from scratch than to reverse engineer such a design.

Tampering cannot be prevented if a volatile key is used because the key is erasable; once the key is erased, the device can be configured with any configuration file. For the non-volatile key solution, the device can be set to only accept configuration files encrypted with the stored key. A configuration failure signals possible tampering with the configuration file, whether in the external memory, during transmission between the external memory and the FPGA, or during remotely communicated system upgrades. This is another advantage of a non-volatile key.

3. HANDSHAKING TOKENS

Configuration bitstream encryption is only available in high-density, high-performance SRAM-based FPGAs. The following solution allows any FPGA designs to remain secure even if the configuration bitstream is captured. This is accomplished by disabling the functionality of a user design within the FPGA until handshaking tokens are passed to the FPGA from a secure external device. The secure external device generates continuous handshaking tokens to the FPGA to ensure that it continues operation. This concept is similar to the software license scheme shown in Figure 2.

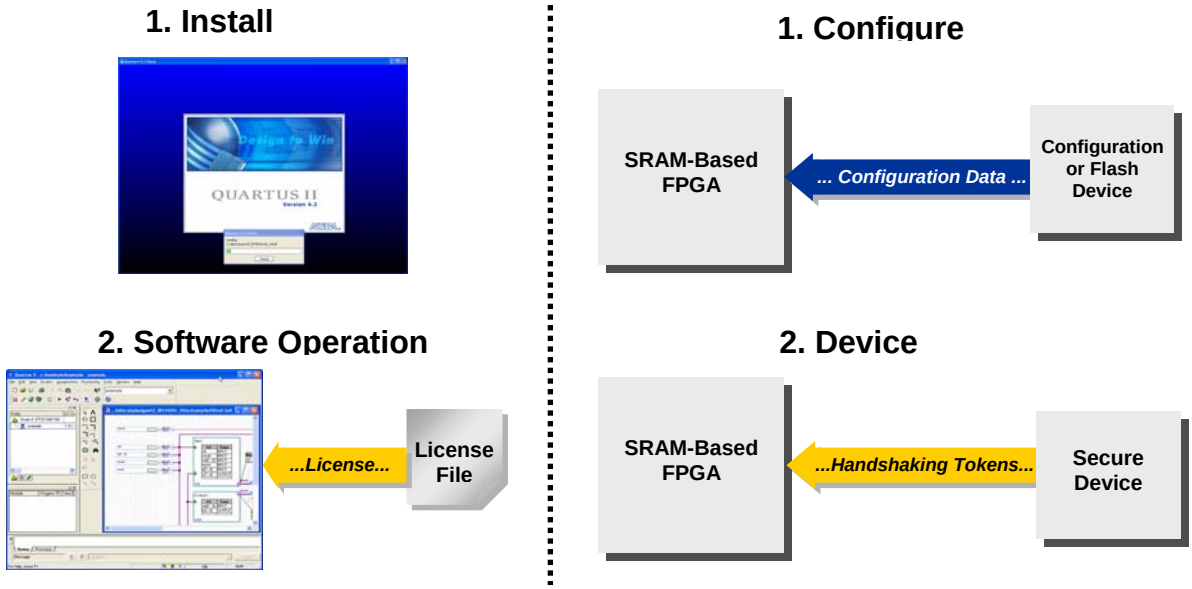


Figure 2: Comparison of Software License Scheme & FPGA Security Scheme

Configuring the FPGA is similar to installing software onto a computer; the configuration bitstream is not protected. The external secure device is similar to the license file. The software will only operate when a valid license file is present. The user design within the FPGA will only operate when the handshaking tokens sent from the external secure device are valid. A simplified hardware implementation for this solution is shown in Figure 3. In this example, a CPLD is used as the secure external device because it is non-volatile and retains its configuration data during power down.

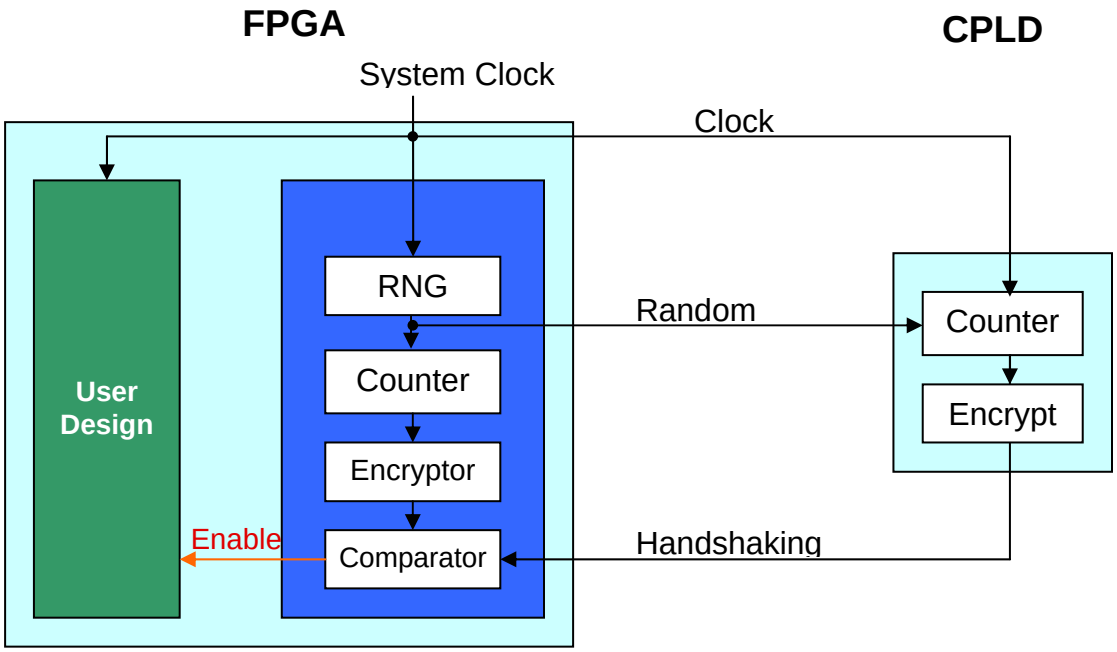


Figure 3: Simplified Hardware Implementation of the FPGA Design Security Solution

After the FPGA is configured, the functionality of the user design within the FPGA is disabled because the enable signal is not asserted, while the security block within the FPGA starts to function. The random number generator (RNG) generates and sends the initial counter value to the CPLD. The CPLD encrypts the counter value and sends the resulting handshaking token to the FPGA. If the handshaking token matches the data generated internally inside the FPGA, the enable signal is asserted, and the user design starts functioning. This process continues during the entire operation of the FPGA. A mismatch will cause the enable signal to go low and disable the functionality of the user design. Figure 4 shows an example of how the enable signal is used with a simple AND gate.

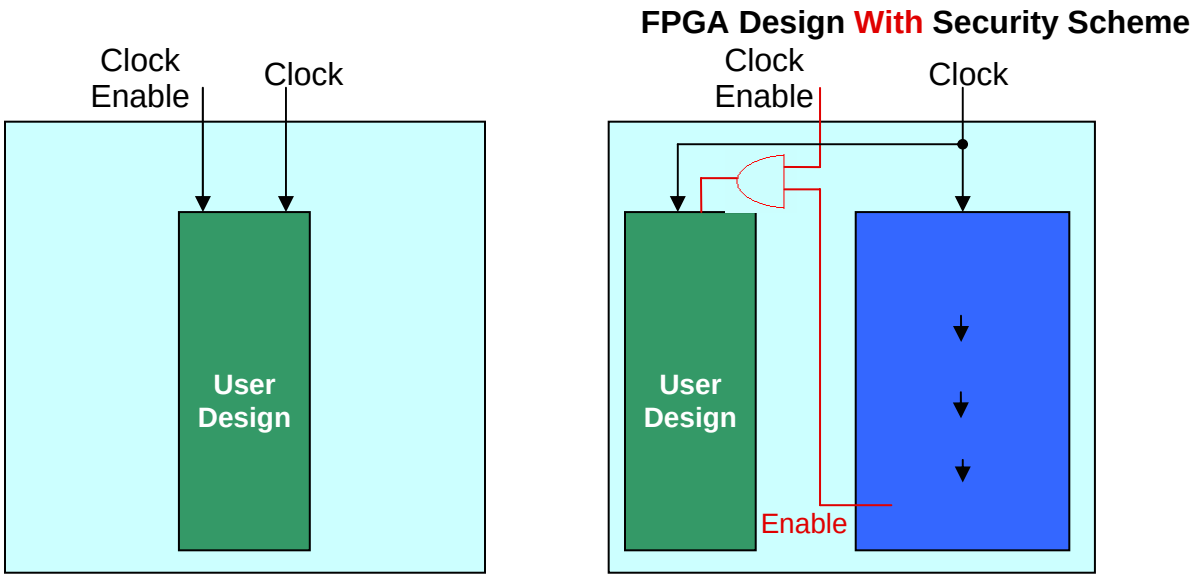


Figure 4: Design with Security Scheme

The FPGA user design only works when the handshaking tokens from the external secure device and the data generated inside the FPGA are identical. Even if the FPGA configuration bitstream is stolen, it is useless, similar to software without a license. Therefore, the FPGA user design is secure from copying. This solution does not provide additional protection against reverse engineering (though difficult) and tampering.

The security of the solution relies on the external secure device to be secure and the handshaking tokens to be unpredictable. A secure external device needs to be non-volatile and retain its configuration during power down (e.g. CPLDs or security processors). The RNG in the solution is critical. It ensures that every time the device starts up, it uses a different initial value. This prevents anyone from storing the handshaking tokens in a storage device. To prevent someone from detecting the pattern in the handshaking tokens, a proven encryption algorithm such as AES should be used.

To ensure that the security scheme works properly, the system clock feeding the FPGA user design should be the same as the system clock feeding the security block. This prevents someone from disabling the security block when the enable signal is asserted. To further increase security, the comparator block can be duplicated several times to produce more enable signals to feed different portions of the user designs.

4. CONCLUSIONS

In an era of ever-increasing security concerns, SRAM-based FPGAs combined with bitstream encryption offer designers of military systems critical advantages. In addition to high density, high performance, low development risk and fast time-to-market benefits over other implementations, they also deliver a secure approach for protecting proprietary designs and IP. For FPGAs without this built-in security feature, an additional non-volatile device can be used to protect the FPGA design by supplying handshaking tokens.

5. REFERENCES

[1] Design Security using MAX II CPLDs, Altera
http://www.altera.com/literature/wp/wp_m2dsgn.pdf

[2] Design Security in Stratix II Devices, Altera
http://www.altera.com/products/devices/stratix2/features/security/st2-security.html

DESIGN OF SECURITY ARCHITECTURE FOR SDR SYSTEM

Kim Mun Gi

Contents

- Introduction
- About Research
- Design of security architecture for SDR Terminal
- Conclusion

1. Introduction

- New radio technology of mobile communication.
- No necessity to change H/W.
- Through S/W, H/W reconfiguration is available.
- That is, can download S/W and reconstruct H/W.
- Then, S/W's administration is required.
  - Security Consideration
    - Software Integrity and Confidentiality
    - Terminal and Data Authentication
    - Transmission security

2. About Research

2.1 SDR System Security Requirements

- Expected to solve the compatible problems of various mobile communication standards.

- If this is constructed, We need
  - Integrity and Confidentiality for download software
  - Available security algorithm negotiation
  - Authentication & Access Control
  - Data encryption

2. About Research

2.2 Access Control method for SDR Terminal

- Access Control prevents unlawfulness access of software.
- Access competence is established in Terminal’s state.
- Access control of Identity-Based Policy consists on subject or sub group’s position.

2. About Research

2.3 Certificate management (1)

- X.509v3 Certificate Fields
  - Certificate::= SEQUENCE {
    tbsCertificate TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue BIT STRING }
    TBSCertificate ::= SEQUENCE {
    Version [0] EXPLICIT Version DEFAULT v1,
    SerialNumber CertificateSerialNumber,
    Signature AlgorithmIdentifier,
    Issuer Name,
    Validity Validity,
    Subject Name,
    SubjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL }

2. About Research

2.3 Certificate management(2)

- X.509 Attribute Certificate Definition
  - AttributeCertificate ::= SEQUENCE {
    acinfo AttributeCertificateInfo,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue BIT STRING }
    AttributeCertificateInfo::= SEQUENCE {
    Version AttCertVersion -- version is v2,
    Holder Holder,
    Issuer AttCertIssuer,
    Signature AlgorithmIdentifier,
    SerialNumber CertificateSerialNumber,
    AttrCertValidityPeriod AttCertValidityPeriod,
    Attributes SEQUENCE OF Attribute,

    IssuerUniqueID UniqueIdentifier OPTIONAL, Extensions Extensions OPTIONAL }

3. Design of security architecture for SDR system

- SDR System Components
  - SDR Terminal
  - SP (Service Provider)
  - TMS (Terminal Management Server)
  - DS (Data Server)
  - CA (Certification Authority)
  - Attribute CA
  - PKC (Public Key Certificate)
  - AC (Attribute Certificate)

3. Design of security architecture for SDR system

- 3.1 PKC issuance process(1)

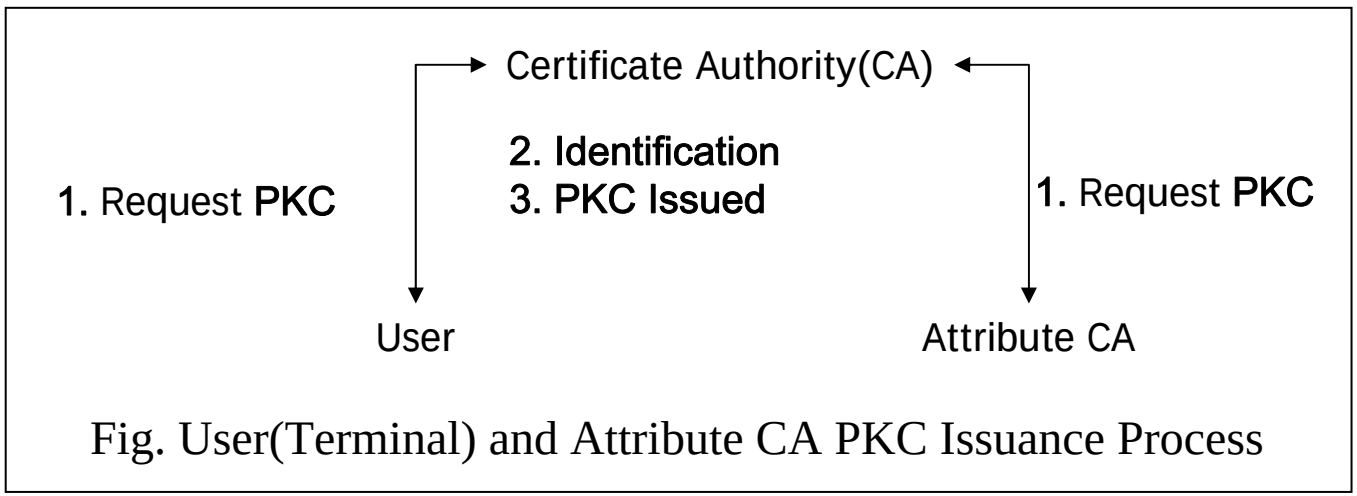


Fig. User(Terminal) and Attribute CA PKC Issuance Process

- User and attribute CA issue PKC from certificate Authority for certification mutually.
- Certificate Authority issues PKC to attribute certificate CA with SDR Terminal.

3. Design of security architecture for SDR system

- 3.1 AC issuance process(2)

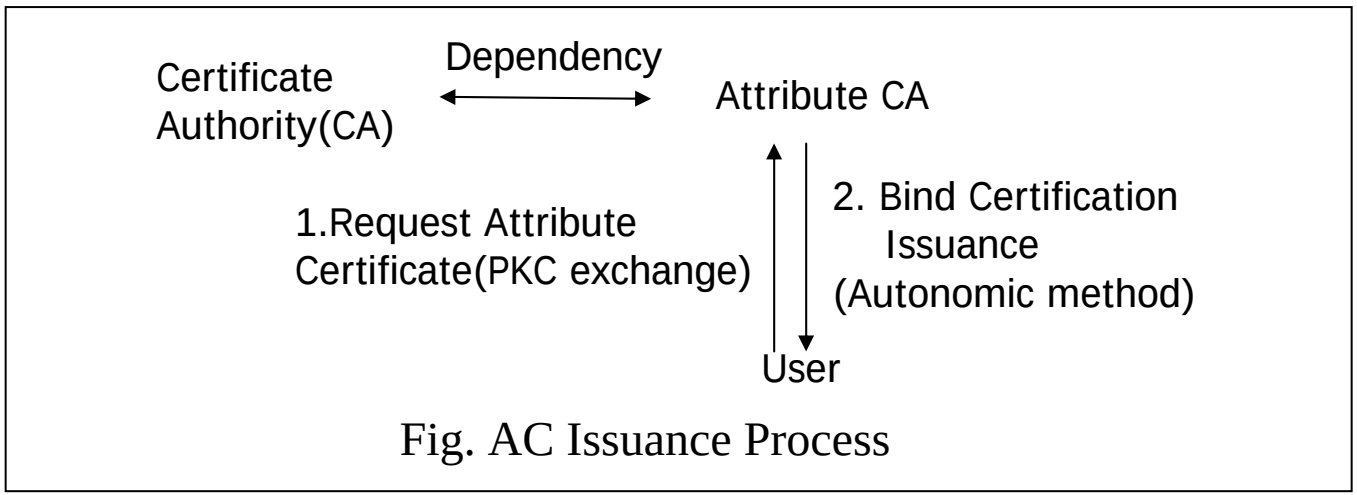


Fig. AC Issuance Process

- User transmits own PKC to attribute CA for attribute certificate issuance.
- Attribute CA transmits own PKC to user. (mutually relationship of mutual trust establishment)
- Attribute CA asks to certification engine for user identification.
- If user certification process is completed, issue bind certificate to user.

3. Design of security architecture for SDR system

- Comparison Push and Pull model(1)
  - Push Model
    - Advantage
      - Simplification of certification processing process.
      - Frugality of certification processing time.
      - Frugality of additional expense for certificate administration.
    - Shortcomings
      - There are certificate damage and loss danger.
      - Can be eavesdropping by hacker at certificate transmission process.

3. Design of security architecture for SDR system

- Comparison Push and Pull model(1)
  - Pull Model
    - Advantage
      - Can reduce certificate damage and loss.
      - Have softness of attribute information alternation.
    - Shortcomings
      - Additional expense is required for certificate administration.
      - Is proportional in user number and communication load happens.

3. Design of security architecture for SDR system

- 3.2 Access control model using PKC and AC

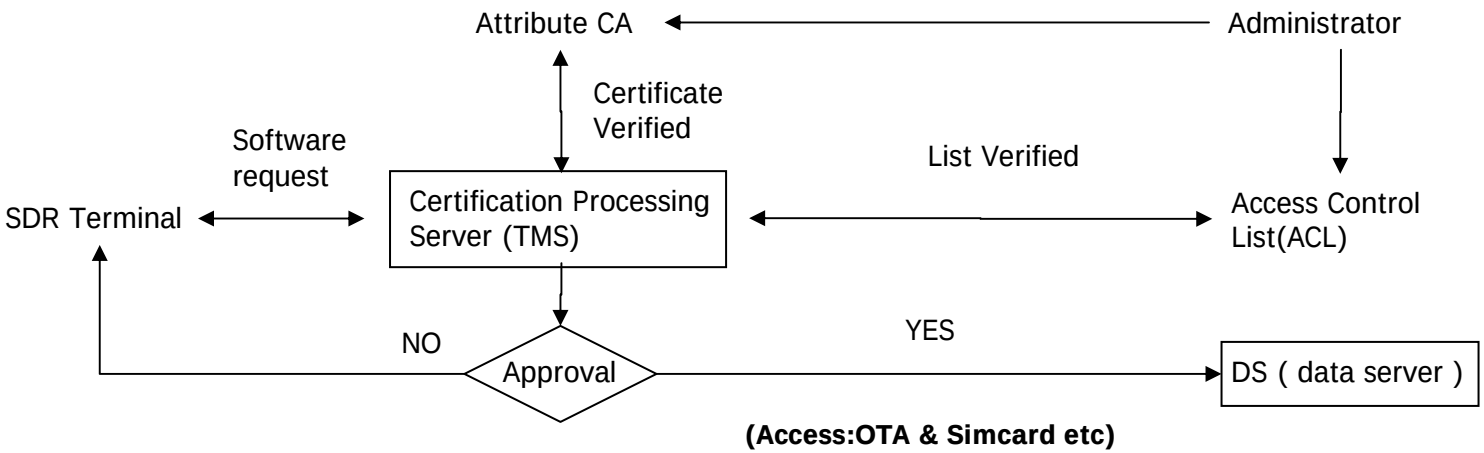


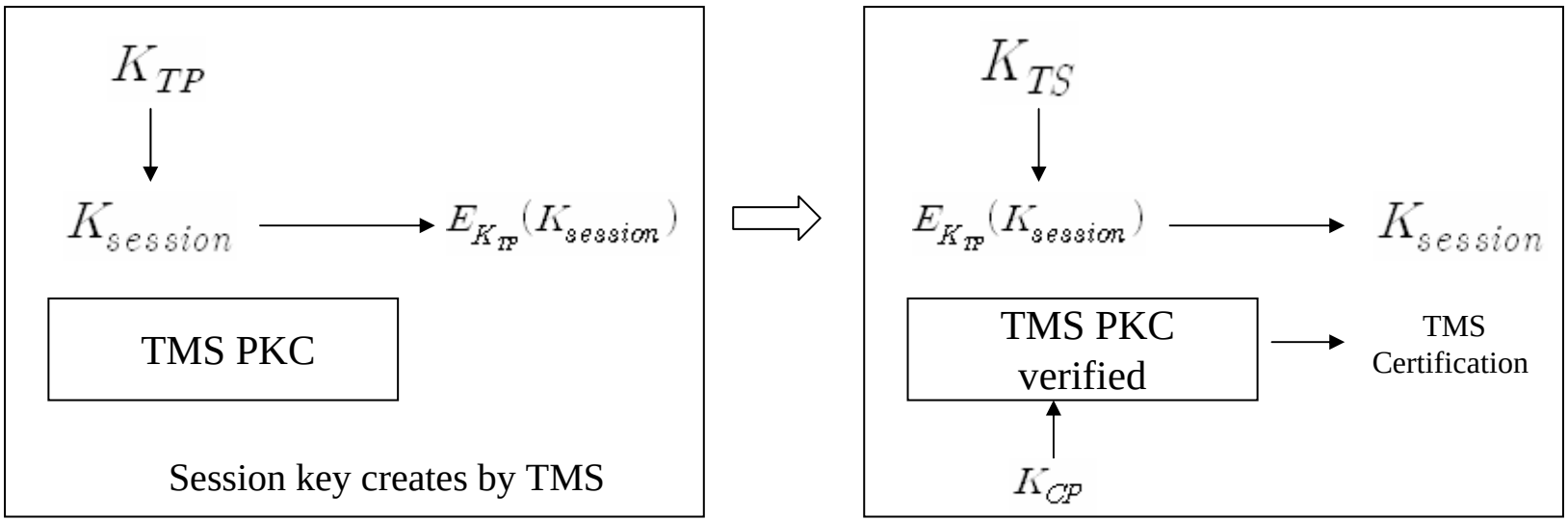
Fig. AC Issuance Process

3. Design of security architecture for SDR system

- 3.3 Software transmission security method
- Definition
  - K_TP : Terminal public key K_TS : Terminal private key
  - K_TMS_p : TMS public key K_TMS_s : TMS private key
  - K_DS_p : DS public key K_DS_s : DS private key
  - K_session : Session Key that server creates
  - E_K_TP(K_session): Session Key that encode by public key of terminal
  - E_K_DS_p(K_session): Session Key that encode by public key of DS
  - E_K_TMS_p(K_session): K_session that encode by K_TMS_p
  - E_K_TS(E_K_TMS_p(K_session)): Key that encrypt encoded key to K_TS
  - K_session(R_m) : Terminal request message that is encoded by K_session
  - K_session(M_m): TMS response message that is encoded by
  - S_m: Digital signature message that is encoded by K_TP
  - MD(M) : Message Digest that use Hash algorithm
  - MD(M)`: Terminal creates Message digest that to compare with MD(M) value.
  - E_K_DS_s[MD(M)] : Digital Signature that encrypt MD(M) by K_DS_s

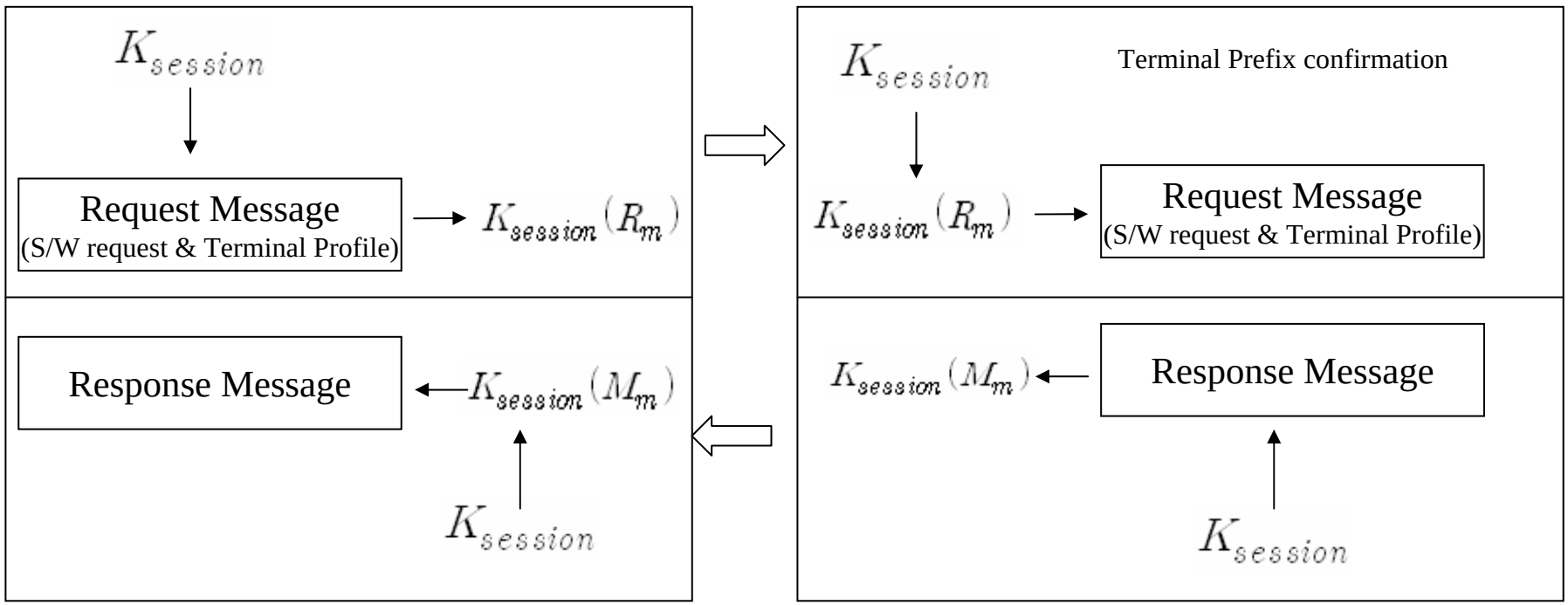
3. Design of security architecture for SDR system

- 3.3.1 session key shared method between Terminal and TMS



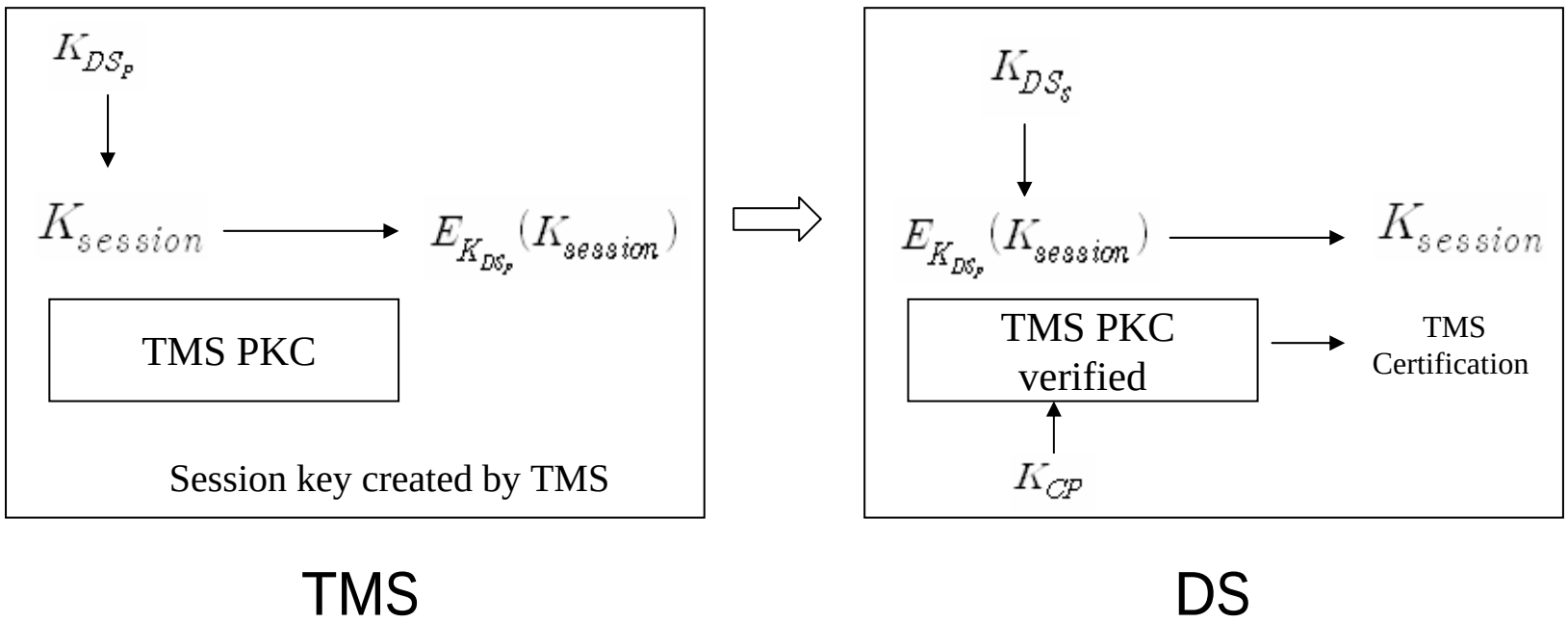
3. Design of security architecture for SDR system

- 3.3.2 Encryption method between Terminal and TMS



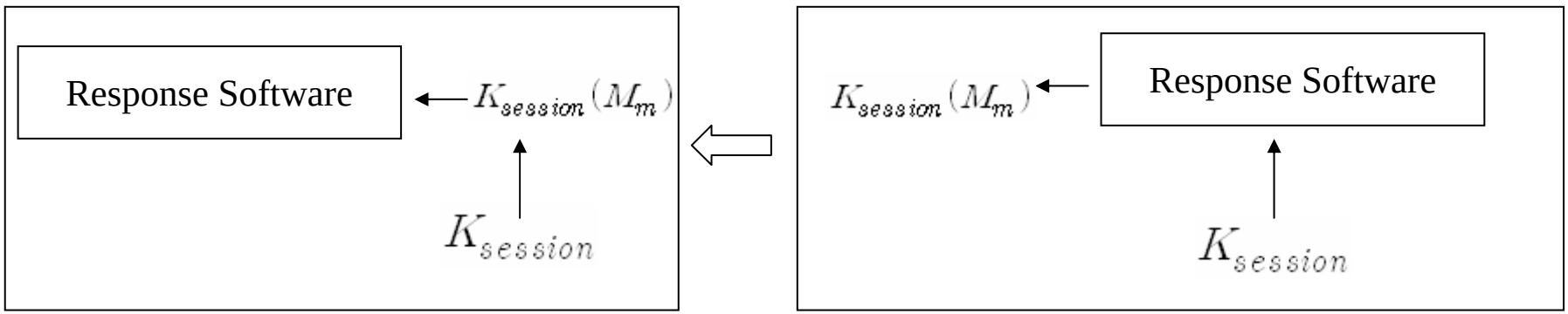
3. Design of security architecture for SDR system

- 3.3.3 DS Transmission Security
  - Encryption method to use Session key



3. Design of security architecture for SDR system

- 3.3.3 DS Transmission Security
  - Encryption method to use Session key

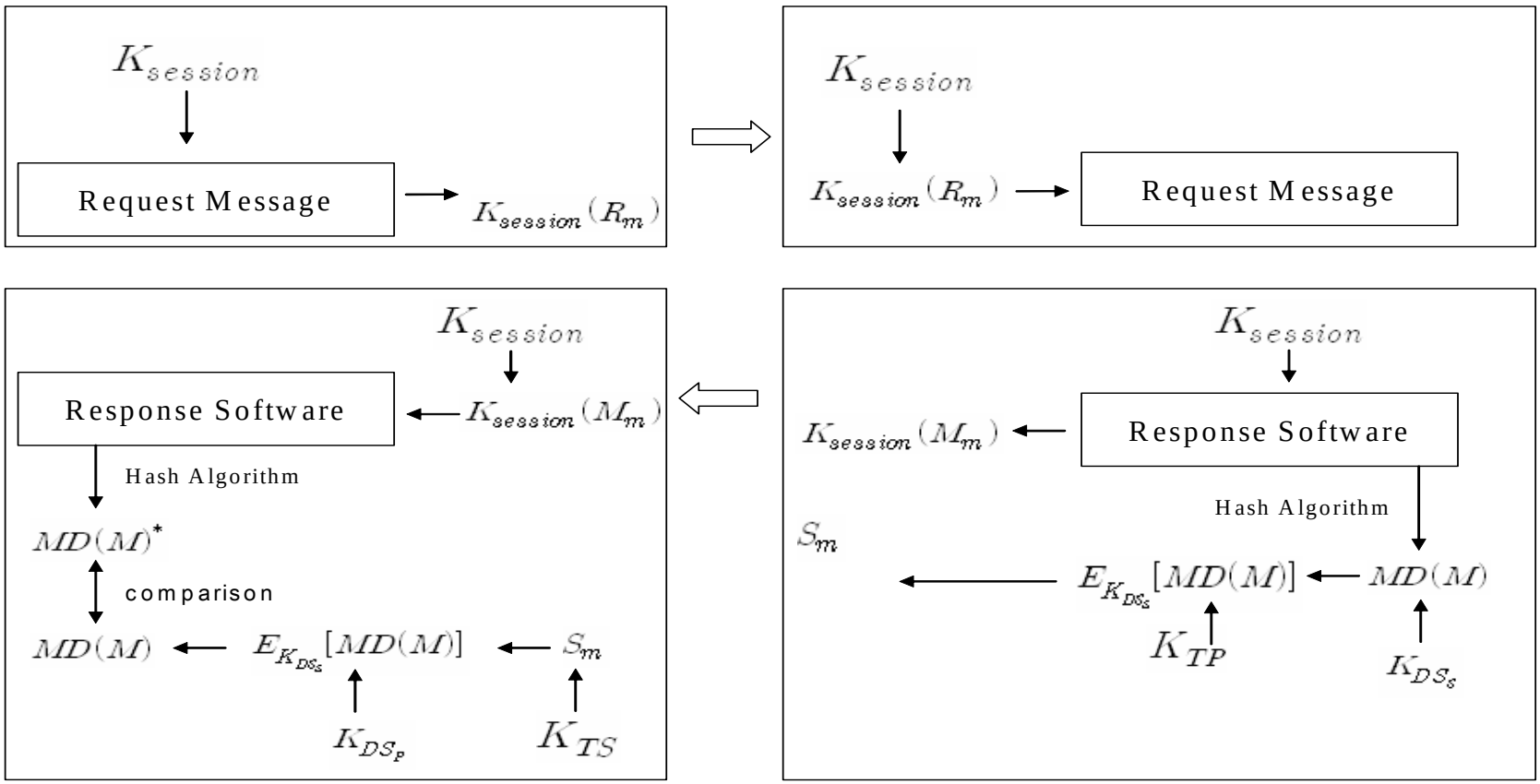


SDR Terminal

DS

3. Design of security architecture for SDR system

- 3.3.3 DS Transmission Security
  - Encryption method including node's signature function



4. Conclusion

- This paper proposes the certificate process using PKC for SDR Terminal
- S/W access method using AC
- The transmission method for the integrity of S/W
- We will consider the efficiency of Terminal based on this paper
- Continue to research about lightening PKC and the limit of AC access control scope
- Furthermore, in aspect of transmission security , we will continue to research about Key administration, establishment and distribution.