Available online www.jocpr.com

Journal of Chemical and Pharmaceutical Research, 2014, 6(6):2504-2511

Research Article

ISSN : 0975-7384
CODEN(USA) : JCPRC5

The system research on trust management in electronic commerce

Fan Wang

Shandong Youth University of Political Science, Jinan, Shandong, China

---

ABSTRACT

This thesis will identify what trust management is in the context of the electronic commerce and propose a general architecture to close the gap between trust and cryptography. I will describe two specific languages for describing trust policies and a general mechanism for evaluating whether a request for action complies with policy.

Key words: Electronic commerce, trust policies, cryptography

INTRODUCTION

Many activities of growing importance in the "information infrastructure," including electronic commerce and mobile programming, depend critically on precise and reliable ways to manage trust. Users will need to know how trustworthy information is before they act on it. For example, they will need to know where the information comes from (authentication), what kind of information it is (content), what it can do (capability), and whether it was altered during transmission (integrity). Without knowledge of what or whom to trust, users may treat a piece of potentially valuable information as yet another stream of random bits. Worse yet, malicious parties may lure users into believing that a false piece of information is trustworthy [1-3].

Many existing mechanisms and protocols address specific aspects of trust in the information infrastructure, but none provides a complete solution. For example, digital signatures allow publishers to create and distribute non-refutable proofs of authorship of documents. Public key infrastructures bind public keys to entities so that users can establish trust chains from digital signatures to signers. Metadata formats allow creators of information resources or trusted third parties to make assertions about these resources. Users can query and process the trusted assertions before deciding what to do with the information resources. Each of these mechanisms and protocols defines a subset of all potential trust problems and solves or partially solves this subset [4-7].

The goal of my research is to design a complete trust management infrastructure, in which trust is specified, disseminated, and evaluated in parallel with the information infrastructure. I have identified four major components of a trust management infrastructure: the metadata format, the trust protocol, the trust policy language, and the execution environment, which are defined in Chapter two. Under this framework of study, I discovered that most existing approaches to trust deal with metadata formats and trust protocols but lacked general trust policy languages for specifying user preferences and generic environments for evaluating them. This finding leads to my interest and involvement in REFEREE [8-10].

REFEREE is a result of collaboration among researchers from AT&T and W3C, including myself. It was designed to be a general-purpose execution environment for all Web applications requiring trust. REFEREE evaluates user policies in response to a host application's request for actions. Policies are treated as programs in REFEREE. For a given request, REFEREE invokes the appropriate user policy and interpreter module and returns to the host application an answer (with justification) to the question of whether or not the request complies with the policy [11-14].

The underlying architecture of REFEREE allows different trust policy languages and trust protocols to co-exist in one execution environment. They are treated as add-on software modules and can be installed or de-installed modularly. At the time of development, we were unable to find a suitable policy language to demonstrate all the features of REFEREE, and so we designed the Profiles-0.92 language.

In order to develop a deeper understanding of REFEREE and to demonstrate its feasibility, power, and efficiency, I built a reference implementation of the REFEREE trust management system. The implementation includes a set of the core REFEREE data types and methods, a PICS protocol, and a Profiles-0.92 policy interpreter to evaluate polices based on the PICS metadata format. In addition, I implemented another policy language called PicsRULZ and integrated it into the reference implementation, in order to demonstrate REFEREE's ability to handle multiple policy languages in particular and multiple software modules generally.

TRUST MANAGEMENT

The term trust management has received a great deal of attention in the network security community since it was first introduced in the paper "Decentralized Trust Management" by Blaze, Feigenbaum, and Lacy. People have compared and contrasted these systems and their capabilities and limitations. This article reviews the concept of "trust management" as the starting point for my thesis work. Later discussions of REFEREE in Chapter three and PicsRULZ and Profiles-0.92 in Chapter four address specific components of "trust management". As formulated by Blaze, Feigenbaum, and Lacy, trust management addresses the question "is this request, supported by these credentials, in compliance with this user policy?" The paper identified three components of trust management:

- security policies
- security credentials
- trust relationships

Security policies are local policies that an application trusts unconditionally. Security credentials are assertions about objects by trusted third parties. Trust relationships are special cases of security policies. An example in the paper illustrated the use and the interactions among the three components:

An electronic banking system must enable a bank to state that at least k bank officers are needed to approve loans of $1,000,000 or less (a policy), it must enable a bank employee to prove that he can be counted as 1 out of k approvers (a credential), and it must enable the bank to specify who may issue such credentials (a trust relationship).The paper referred to the study of the three components and their interactions as the trust management problem. The authors believe that the trust management problem is a distinct and an important aspect of security in network services and that such problem can be solved using a general mechanism that is independent of any particular application or service. They propose is a trust management layer that applications and services can build on top of. Policymaker is a trust management system designed to meet the needs of this layer. It is a three-part solution: a credential format to represent authorization assertions, a security policy language to express user preferences, and an execution environment to evaluate certificates and policies. Policymaker broke new ground by expressing credentials and policies as programs. The execution environment acts like a database query engine: The host application sends to the execution environment a request for action and a user policy, and the environment returns an answer to the question of whether the credentials prove that the request complies with the policy.

TRUST MANAGEMENT INFRASTRUCTURE

A trust management infrastructure is a conceptual framework for the design of a coherent solution to various trust decisions that must be made in what is commonly referred to as the "information infrastructure". A trust management infrastructure allows parties to make trusted assertions about objects in the information infrastructure, and applications to acquire these assertions and make trust decisions based on them. The framework is independent of the trust criteria imposed by any particular application and of the type of assertions made by a trusted party.

Figure 1 shows a component dependency graph in the trust management infrastructure. Diamonds represent components in the trust management infrastructure and arrows represent dependency relations.

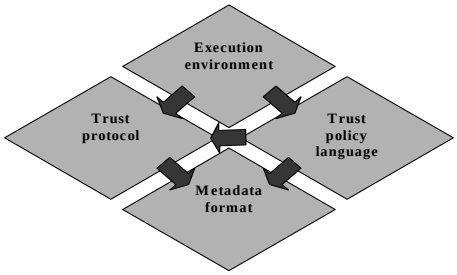

Figure 1: Dependency Graph of Trust Management Infrastructure Components

A metadata format is independent of any other components in the trust management infrastructure. It can be distributed by multiple protocols and operated on by multiple trust policy languages. For example, it is possible to represent SDSI certificates as PICS labels and use them in the SDSI public-key distribution protocol.

POLICYMAKER

PolicyMaker was the first system to take a comprehensive approach to trust problems independent of any particular application or service. It has a general metadata format ("credentials"), a trust policy language, and an execution environment. As indicated in Section **Error! Reference source not found.**, PolicyMaker does not deal with the trust protocol component of what I call the trust management infrastructure. **Error! Reference source not found.** shows a graphical representation of PolicyMaker in the trust management infrastructure.

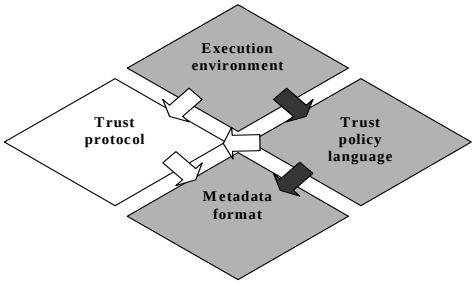

Figure 2: PolicyMaker in the Trust Management Infrastructure

Policymaker has its metadata format called credentials.

It broke new ground by treating credentials as programs. A credential is a type of "assertion." It binds a predicate, called a filter, to a sequence of public keys, called an authority structure. The form of an assertion is:

Source ASSERTS AuthorityStruct WHERE Filter: Here, source indicates the source of authority, generally a public key of an entity in the case of a credential assertion. AuthorityStruct specifies the public key or keys to which authority is granted. Filter specifies the nature of the authority that is being granted. Both AuthorityStruct and Filter are represented as programs to maximize their generality. For example, the following PolicyMaker credential indicates that the source PGP key "0x01234567abcdefa0b1c2d3e4f5a6b7" asserts that Alice's PGP key is "0xf0012203a4b51677d8090aabb3cdd9e2f".

Another major innovation in PolicyMaker is the decision to make credentials and policies the same type of object. A policy is also an assertion. The only difference is that policies are unconditionally trusted locally, and credentials are not. The source field in a policy assertion is just the keyword "POLICY", rather than the public key of an entity granting authority. Credentials are signed assertions, and the public key in the source field can be used to verify the signature.

POLICY LANGUAGE

Policy Language shows how the REFEREE execution environment processes queries, interprets trust policies and runs trust protocols in a generic, application-independent way. To prove that REFEREE is indeed a general-purpose execution environment, I implemented two different policy language interpreters as REFEREE modules, namely PicsRULZ and Profiles-0.92.

Both PicsRULZ and Profiles-0.92 describe trust policies based on PICS labels. While PicsRULZ is considerably simpler and easier to use and implement, Profiles-0.92 is more general and expressive. Section one identifies the design goal of a policy language. Sections two and three describe PicsRULZ and Profiles-0.92 in turn. Section four provides four sample policy scenarios and their respective PicsRULZ and Profiles-0.92 translations.

Design Goals
A policy language describes user policy in a machine-readable format. Despite its simple goal, the design of a good policy language may be more than an engineering task. The more complex the language is, the more expressive the policies can be, at the cost of being more difficult to implement, prove correctness, or build a user interface on top. This section sets asides these engineering tradeoffs, and focuses on the desired properties of a policy language.

Safe
The policy written by a policy language should not cause any undesirable side effect to its host application. That is, assuming the underlying policy interpretation is correct, there should be no way to write a valid policy that crashes the host computer.

Transferable
A profile should be transferable among different applications and platforms. This property allows not only a company to specify a profile for its employees to use on different applications and platforms, but also a user to carry his or her profile to other locations without reconfiguration.

Simple
A policy language should not be a general-purpose programming language (in the sense of Turing-complete), but a simple policy language designed specifically to describe trust policies. However, the language should have an extension mechanism to leave room for future expansion. This property comes hand in hand with the safety and the transferability of a policy language; a simpler language is easier to prove safety and more likely to be executed by an untrusted party when a policy is being transferred.

Well-defined
A policy written in a policy language should be unambiguous irrespective of its specific implementation. It is as if writing a book of law, where the citizens know exactly what is legal and not legal.

Expressive
The language construct should be expressive enough to accommodate realistic policies different users want to specify under different circumstances. The level of expressiveness may depend on programming ability of the people creating the policy, or the complexity of the user interface.

The and operator is the three-valued version of the Boolean and operator. **Error! Reference source not found.** describes the operation of and when it is given two arguments. The first row represent the truth value for the first argument, the first column represent the truth value for the second argument, and the rest of the cells represent the result of an and operation.

Table 1: Truth table for the and operator

rule1 \ rule2	true	unknown	false
true	true	unknown	false
unknown	Unknown	unknown	false
false	False	false	false

The and operator can take any number of arguments. For more than two arguments, and operator recursively reduces itself one argument at a time: (and arg1 arg2 … argn) = (and (… (and arg1 arg2) … argn)

The and of a single argument is that argument itself. The and of no argument is true by definition. If one of the arguments return false, the and rule terminates and the rule returns a false, because further evaluations will not change the outcome of a false.

The or operator: The or operator is the three-valued version of Boolean or operator. **Error! Reference source not found.** describes the operation of or when it is given two arguments:

Table 2: Truth Table for the or operator

rule1 \ rule2	true	unknown	False
true	true	true	True
unknown	true	unknown	Unknown
false	true	unknown	False

As and operator, or operator can take any number of arguments, and they are recursively reduced if more than two

arguments are present. The or of a single argument is that argument itself. The or of no arguments is false by definition. If one of the arguments is evaluated to be true, the or terminates and returns a true, because further evaluation does not change the outcome of a true.

The not operator is the three-valued version of Boolean not operator. It takes exactly one argument. **Error! Reference source not found.** describes the operation of the not operator:

Table 3: Truth Table for the not operator

	output
true	false
unknown	Unknown
false	true

The true-if-unknown operator is a projection function from three-valued logic to Boolean logic. It takes exactly one argument:

Table **4: Truth Table for the true-if-unknown operator**

	output
true	true
unknown	true
false	false

The false-if-unknown operator is also a projection function from three-valued logic to Boolean logic. It takes exactly one argument:

Table **5: Truth Table for the false-if-unknown operator**

	Output
true	true
unknown	false
false	false

EXECUTION ENVIRONMENT

An execution environment is the heart of a trust management system; it is where the local trust policies meet with the rest of the trust management infrastructure, through trust protocols and metadata formats, to make trust decisions as an interconnected entity. The primary jobs of an execution environment are two: interpret trust policies and administer trust protocols. An execution environment takes requests from its host application, and returns an answer that is compliant with trust policies. REFEREE is such an execution environment proposed by researchers from AT&T Labs and W3C, including myself. Under REFEREE, trust protocols and trust policies are represented as software modules, which can be invoked and installed dynamically. They can share other's intermediate result through a commonly agreed API. Together they divide up the trust management tasks into pieces, and solve them as a whole. At each level of computation, every aspect of REFEREE is under policy control.

Section one lists the design goals of an execution environment in a trust management system. Section two introduces REFEREE, our proposed solution. Section three and four describes the REFEREE internal architecture and primitive data types. Sections five and six provide a standard procedure to bootstrap and query REFEREE. The REFEREE execution environment is an extensible and self-modifiable execution environment, although it appears to the host application as a monolithic tri-value decision box. The basic computing unit in REFEREE is a module. A REFEREE module is an executable block of code that processes input arguments and asserts additional statements. It can also defer subtasks to other modules and make trust decisions based on returned assertions. Together the interconnected REFEREE modules can process requests from the host application and produce a recommendation.

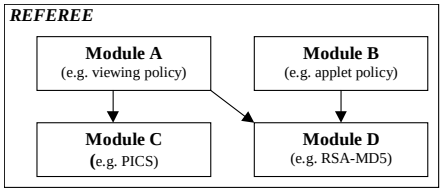

Figure 3: Sample block diagram of REFEREE internal structure.Ease of Use

The separation of duty among REFEREE modules has several advantages. First of all, existing modules can be

updated without affecting other modules, as long as the upgraded modules keep the API backward-compatible. For example, module A and B do not care how module D verifies RSA-MD5 signatures. Therefore module D can be updated with more optimized code without changing module A and B. Moreover, new modules can be introduced dynamically. For example, if module C starts returning PICS labels with DSA-SHA1 signatures that module A cannot verify, module A can upload a module to handle the verification. Other modules in REFEREE, including module B, can then share the new module transparently.

Zooming in, a REFEREE module looks like the following figure 4:

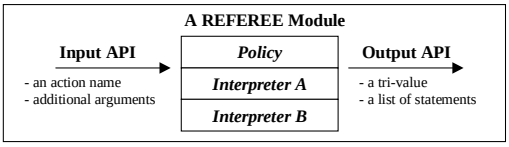

Figure 4: Required interface for every REFEREE module

REFEREE REFERENCE IMPLEMENTATION

To verify the REFEREE design as described in the thesis, I produced a working REFEREE reference implementation as part of my thesis. I chose Java as my REFEREE underlying execution environment. The implementation work included the core REFEREE primitive data types, REFEREE API, PicsRULZ and Profiles-0.92 interpreters, and a user interface for demonstration purpose. REFEREE was ported to Jigsaw proxy as its host application.

Jigsaw was originally designed as a Web server, whose purpose was to provide a basis for experimenting new server-side features. Recently Jigsaw introduced a Client API, which manages requests and performs filtering on behalf of a client. The Jigsaw proxy extracts pieces from both the Jigsaw Server and the Client API. The proxy front end, responsible for accepting network requests and managing them as a pool of threads, is taken from the Jigsaw Server front end. The proxy back end, responsible for redirecting requests to Web servers, is taken from the Jigsaw Client API. This section focuses on the Jigsaw proxy back end, in which REFEREE is embedded.

The Jigsaw Client API is very simple: it takes in a request (generally a URL) and returns a reply (the content of the URL). The API aims to replace the Java standard java.net.URLConnect class, with the advantages of being more robust and modular. A simplified architectural figure is shown below.

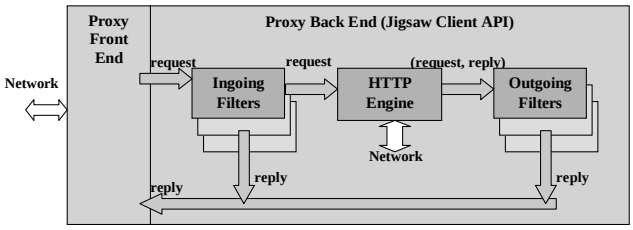

Figure 5: Jigsaw Proxy Architecture

REFEREE is implemented as an ingoing filter in the Jigsaw proxy. When a request is received, the REFEREE filter constructs an equivalent REFEREE request object, which includes an action name and the URL of interest. The request object is then sent to the REFEREE execution environment for evaluation. If the output of the evaluation is true, the filter returns null, allowing the request to flow through without interruption. If the output is unknown or false, the filter returns a default HTML document expressing the request is blocked, along with the justifications returned by REFEREE.

One observation here is that my implementation of the Jigsaw filter has a self-regulating "policy" to respond to the outcome the REFEREE evaluation. That policy is application specific; it is neither controlled nor evaluated by REFEREE, but by the application itself. This observation reinforces the fact that REFEREE is recommendation-based; the burden to enforce the trust management decision is on the application itself. I will discuss more on this aspect in Section **Error! Reference source not found.**.

Recall from Chapter 3, there are two stages in REFEREE. The bootstrap stage corresponds to the initialization of the REFEREE filter. The query stage corresponds to the invocation of the REFEREE filter. In addition, Jigsaw provides a method (callback) to fetch information from the network. The fetch callback is implemented as the Jigsaw Client API itself, except it does not have the REFEREE filter installed.

Label loader is a PICS trust protocol implemented as a REFEREE module. There are four required input arguments, a statement-list, a URL of interest, a rating service URL, and a list of label sources. The input statement-list contains a set of cached PICS labels. When there is a cache hit, Label Loader returns the label without fetching it from the network. The URL of interest and the rating service URL specify what PICS labels to fetch. A list of places to find labels includes embedded in a document (by the keyword "EMBEDDED"), via the HTTP header stream (by the keyword "ALONG-WITH"), or from a list of label bureaus.

The returned value is true if any label is found, unknown if all label bureaus cannot be contact or false if label bureaus can be contacted but no label is returned. The returned statements are parsed PICS labels, which are restructured in a way that are easy for pattern matching and other operations. The exact syntax is in **Error! Reference source not found.**. An example is illustrated below, assuming a Profiles-0.92 policy calls Label Loader (local name "load-label") in the following line:

(invoke "load-label" STATEMENT-LIST URL "http://www.musac.org/v1.0"
(EMBEDDED "http://www.bureau.com")
if Label Loader finds only an embedded PICS label, the returned tri-value is true, and the returned statement-list looks like the following:
(("load-label") (("load-label" "http://www.w3.org/Overview.html" EMBEDDED) ((version "PICS-1.1") (service "http://www.musac.org/v1.0") (by "mailto:mstrauss@research.att.com") (original (PICS-1.1 "http://www.musac.org/v1.0"
labels by "mailto:mstrauss@research.att.com" ratings (s 1 v 0)) (ratings (s 1) (v 0)))))

As the thesis is written, the implementation of the Load Label module does not have a running PICS protocol. Instead the inputs of the raw PICS labels are provided by an input stream from the REFEREE filter. The implementation does parse PICS labels and turn them into REFEREE statements.

CONCLUSION

The REFEREE implementation in the Jigsaw proxy provides many insights on how a trust management system should work in a real-world application. This section attempts to address some of the concerns raised during the implementation, and explain how my particular implementation deals with them.

The first concern is the order in which REFEREE is placed with respect to other tasks in a host application that are concurrently affecting the behavior of the application. Caching is such a conservative task in the Jigsaw proxy, where the order to do trust management and caching matters. I place the REFEREE filter in the highest precedence, so it always gets evaluated. It is considered the most conservative approach, because it would accurately observe time-dependent trust elements, such as expired or revoked certificates, and make correct trust decisions based on them. However, if performance is more critical than accuracy, an application should place the caching filter in front of the REFEREE filter. Determining how the trust management task interacts with other processes in a host application is in a way another "trust policy", and it is outside the scope in which REFEREE can evaluate.

The second concern is whether actions issued during the evaluation of trust management are subject to the same trust management evaluation. For example in the Jigsaw proxy, download-applet requires Label Loader to call the Jigsaw Client API and fetch PICS labels from the network. The act of fetching PICS labels itself may be considered as a trust management problem and be subjected to a label-fetching trust management policy. My current implementation does not invoke another level of trust management during a trust management evaluation. I reject this idea for two reasons. First, it may introduce deadlock if the label-fetching policy in turn requests the same labels before the label can be fetched. The same Label Loader would be called recursively without making any progress. Second, I treat the action of fetching PICS labels as a trust protocol that is safe, secure, without any judgment of trust, therefore the action needs not be subjected to a trust management decision.

The third concern is whether REFEREE should introduce an explicit caching mechanism for performance reason. Currently REFEREE does not have one, and my implementation has no mechanism explicitly for caching purpose. However, my implementation transparently inherits the benefit of Jigsaw's internal caching mechanism (caching filter). When Label Loader needs to fetch labels from the network, it calls the Jigsaw Client API, where the cache filter is activated. The subsequent call to get the same label will be caught by the caching filter, and hence Label Loader gets caching for free. The observation implies that caching is supposed to be transparent from the rest of the processes in the application, and REFEREE needs not implement an explicit caching mechanism.

The fourth concern is whether REFEREE can be application independent as promised. As demonstrated in this implementation, the only two pieces that are "Jigsaw-centric" are the Jigsaw filter and the network fetcher. The Jigsaw filter traps requests from its host and bootstraps REFEREE. The network fetcher fetches information from the network, which are already in place for most network applications. Both of them are considered minimal for an application to do trust management. The rest of the code can be ported to other applications without any modification.

The fifth concern is whether REFEREE introduces disastrous performance hit for doing trust management. My implementation takes less than half a second to evaluate of the sample policy in Section **Error! Reference source not found.**, excluding any network time (my implementation supplies all the network information through a fixed input stream). This observation implies that the bottleneck to do trust management will not be the invocation of REFEREE modules, or evaluation of trust policies. Rather, the bottleneck will be the use of network, where fetching labels from the Web may incur long delays, or the use of cryptography, where validating digital signatures may take large CPU cycles. They are however, the unavoidable steps to make any trust decisions, but the overhead of REFEREE is minimal.

My thesis identifies the trust management problems in the context of the World Wide Web and provides a two-part solution: REFEREE as the general-purpose execution environment and PicsRULZ and Profiles-0.92 as the policy languages. They utilize the existing trust protocols and metadata formats, and together, they form a complete trust management infrastructure in which trust is exchanged and established among mutually untrusting parties in an entrusted information infrastructure.

This thesis has four contributions to the area of trust management: identify the concept of the trust management infrastructure, with the four basic building blocks in the infrastructure. Study current protocols and systems involving trust and identify their strengths and weaknesses. Propose a two-part solution: REFEREE as a generic execution environment, and Profiles-0.92 as a flexible trust policy language. Implement reference versions of REFEREE and Profiles-0.92 and prove that the concept of a generic trust management infrastructure is a realistic goal. The work on REFEREE and trust management is definitive or conclusive in its current state, but rather that it is a step forward in the understanding of the intricacies of trust. Of course, more work is needed. In particular, we need network experts to build robust and yet more efficient metadata formats and trust protocols. We need language experts to define simple and yet expressive trust policy languages. We need system experts to structure secure and yet dynamic execution environment. We also need user interface experts to deliver a user-friendly and yet feature-rich user interface to take advantage of a sophisticated trust management infrastructure beneath.

REFERENCES

[1]Liu Xiao-lan. *China Sport Science and Technology*. **1984**, 29(13), 46-49.
[2]Luo Yang-chun. *Journal of Shanghai Physical Education Institute*. **1994**, 23(12), 46-47.
[3]Wan Hua-zhe. *journal Of Nanchang Junior Colle*ge. **2010**, 3, 154-156.
[4]Li Ke. *Journal of Shenyang Sport University.* **2012**, 31(2), 111-113.
[5]Zhang Shu-xue. *Journal of Nanjing Institute of Physical Education*. **1995**, 31(2), 25-27.
[6]Pan Li. *Journal of nanjing institute of physical education(natural science).* **2004**, 19(1), 54-55.
[7]Li Yu-he; Ling Wen-tao. *Journal of Guangzhou Physical Education Institute*. **1997**, 17(3), 27-31.
[8] Xu Guo-qin. *Journal Of Hebei Institute Of Physical Education*. **2008**, 22(2), 70-72.
[9] Chen Qing-hong. *China Sport Science and Technology*. **1990**, 21(10), 63-65
[10] Tian Jun-ning. *Journal of Nanjing Institute of Physical Education*. **2000**, 14(4), 149-150.
[11] Zhang B.; Zhang S.; Lu G.. *Journal of Chemical and Pharmaceutical Research*, **2013**, 5(9), 256-262.
[12] Zhang B.; *International Journal of Applied Mathematics and Statistics*, **2013**, 44(14), 422-430.
[13] Zhang B.; Yue H.. *International Journal of Applied Mathematics and Statistics*, **2013**, 40(10), 469-476.
[14] Zhang B.; Feng Y.. *International Journal of Applied Mathematics and Statistics*, **2013**, 40(10), 136-143.