Malaysian Journal of Computer Science, Vol. 13 No. 1, June 2000, pp.76-83

OBJECT-ORIENTED APPROACH TO SPECIFY SECRET SHARING PROTOCOL IN SECURITY CRITICAL SYSTEM USING FORMAL METHOD

Kok Meng Yew, Mohammad Zahidur Rahman and Sai Peck Lee
Faculty of Computer Science & Information Technology
University Malaya
Kuala Lumpur, Malaysia
email: zahidur@siswazah.fsktm.um.edu.my

ABSTRACT

Computers are being used increasingly in different security critical systems like electronic commerce and health care systems. The formal analysis turns out to be very useful that its application should be routine for financial and security critical systems. To win the confidence of users of a secured system, like secure secret sharing protocol, there is no other alternative than the formal method.

In this paper, we first briefly introduce the secret sharing system and Object-Z formal specification tool. Then we present our design of the secret sharing scheme. In our model, the participating user and the information sharing authority dealer are modeled. To exchange information securely between users and the dealer, private channels are used. Broadcast channel is used for open information exchange. Both types of channels have been modeled. The model is formally specified by introducing the concept of combine object for collecting secret shares and for checking whether they lie in the perfect secret sharing scheme. We finally conclude with our experience.

Keywords: Formal method, secret sharing.

1.0 INTRODUCTION

Computers are being used increasingly in different security critical systems like electronic commerce and health care systems. The formal analysis turns out to be very useful that its application should be routine for financial and security critical systems [1]. Formal specification is defined in the IEEE Software Engineering Standards Collection as [2] "a specification written and approved in accordance with established standards", and "a specification written in a formal notation, often for use in proof of correctness".

To avoid single point of failure and achieve confidentiality of users so that their vital information is not compromised, various secret sharing schemes are proposed which will be discussed in Section 2.0. The secret sharing mathematical schemes do not specify the structure of information system (IS) to which formal methods are employed.

In this paper, we investigate the issues involved in the design of the mechanism for the secret sharing. We present a generic model of secret sharing that can be used to support software engineers in reusing existing system designs and in developing new systems. The paper also shows how a formal software engineering notation, in this case Object-Z, extension of Z notation, can be used to develop models. In our model, the participating user and the information sharing authority dealer are modeled. To exchange information securely between users and the dealer, private channels are used. The broadcast channel is used for open information exchange. Both types of channels have been modeled. The model is formally specified by introducing the concept of combine object for collecting secret shares and for checking whether they lie in the perfect secret sharing scheme.

2.0 SECRET SHARING SCHEME

Formally, a secret sharing scheme can be defined as a method of sharing a secret **s** among a set of participants **P**, in such a way that qualified subsets of **P** can reconstruct the value of **s**, whereas any other (non-qualified) subset of **P** cannot determine anything about the value of **s**. Each of the participants receives a share of the secret **s**. In the rest of the text, the share means the secret share distributed among the participants **P**. Secret sharing schemes are useful in any important action that requires the concurrence of several designated people to be initiated. They are also used in the management of cryptographic keys and multi-party secure protocols.

The earliest secret sharing schemes studied were **(k,n)** threshold schemes. A **(k,n)** threshold scheme allows a secret to be shared among **n** participants, in such a way that any **k** of them can recover the secret, but any **k-1** of them have absolutely no information on the secret. Shamir [3] and Blakley [4] independently showed how to realize **(k,n)** threshold schemes. Subsequently, Ito et al [5] and Benaloh and Leishter [6] described a more general method of secret sharing. They showed how to realize a secret sharing scheme for any monotone access structure. Other threshold schemes have been devised following the contribution of Shamir and Blakley [7, 8].

The issue of efficiency (i.e. share sizes) of such schemes has been considered in several papers [9, 10, 11]. Benaloh and Leichter [6] suggested a scheme for structures represented by

monotone formulae. The span program is the most general access structure for which efficient secret sharing schemes are used. Krawczyk [12] suggested the notion of computational secret sharing.

To avoid cheated by a dealer by not sharing a secret and to avoid the denial of service due to non-cooperation of some of participants in reconstruction phase, fault-tolerant secret sharing scheme known as verifiable secret sharing scheme (VSS) was first proposed by Chor et al [13]. After the original introduction of the concept of VSS, several VSS protocols were proposed which were motivated by various applications like secure multi-party computation [14].

In this paper, we formally specify the Shamir's (K, n) threshold secret sharing scheme. In this scheme, the Dealer sends secret shares to n Players secretly through private channels. The information through a private channel is only known to the recipient. To protect the Dealer's integrity over the share, the Dealer broadcasts a hashed value for each share to all the participants and every participant collects it for future verification. The context diagram of the secret sharing scheme is shown in Fig. 1.

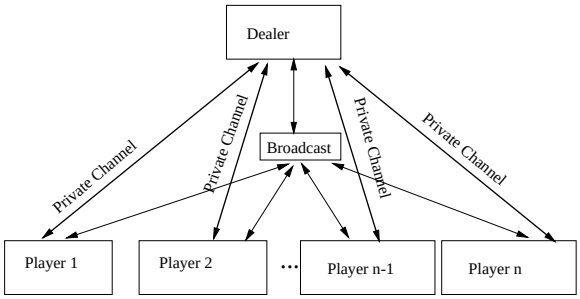

Fig. 1: Secret sharing scheme

3.0 FORMAL METHOD AND OBJECT-Z

The model presented here is written using the object-oriented extension of software engineering notation Z, Object-Z. Since Object-Z is not a typical choice of modeling language within the secret sharing community, our choice deserves some explanations. The suitability of object-Z for our task derives from some observations about secret sharing scheme. Firstly, a secret sharing scheme involves a set of users. Object-Z provides all the usual machinery of set notation. Secondly, a generic model of secret sharing scheme requires the abstract description of a function. Object-Z is well-suited in defining constraints on functions without specifying the function explicitly.

Object-Z [15, 16] is an extension to Z [17] to support specification in an object-oriented style. An object-oriented specification describes a system as a collection of interacting objects, each of which has a prescribed structure and behavior. A Z specification defines a number of state and operation schemas. In object-Z, a state schema and schemas operating on that state can be grouped together to form a class. A class is a template for objects. Each object of the class has a state conforming to the class state schema and is subject to state transitions which conform to the class operations. Classes can be related by inheritance and the inheritance can be used as a type:instances of that type. This allows objects to refer to other objects. A general class definition has the following form:

ClassName[*generic parameters*]
visibility list
inherited classes
type definitions
constant definitions
state schema
initial state schema
operation schemas
---
history invariant

An operation can refer only to the state of the object to which it belongs. The possible constituents of a class are: a *visibility list*, *inherited classes*, *type* and *constant definitions*, a *state schema*, an *initial state schema*, *operation schemas* and a *history invariant*. The *visibility list* (prefixed by $\upharpoonright$) restricts access to the listed features. If it is omitted, then all features are visible. The *inherited classes* denote those classes that are inherited by the defining class. Inheritance results in a merger of state, initial state schema Init and operation schemas having the same name. Complex classes can be specified to inherit from other classes, or to include references to instances of objects. The type and constant definitions of the inherited classes and those declared explicitly in the derived class are merged. Any schemas with the same name and the state schemas are conjoined. Also, the history invariants are conjoined. Name clashes can be resolved by renaming. The visibility lists may be used to disable inheritance of properties of a class. Redefinition of properties may be achieved by hiding and renaming of properties.

Type and *constant* definitions are same as in Z. The *state schema* is nameless and comprises declarations of state variables and state predicate. The state predicate forms the class invariant. It is implicitly included in every operation and the initial state schema. The *initial state schema* defines the possible initial states. The initial state is not unique. Operation schemas describe the methods defined for the class. The operations have a Δ-list of those individual state variables whose values may change when the operation is applied to an object of the class. State variables which are not listed in the Δ-list are unchanged by an operation. The lower part defines a predicate relating the initial and final states of the operation. The precondition and postcondition describe the effect of an operation. The *history invariant* is a predicate over histories of objects of the class. It further constraints the possible behavior of such objects. Fairness and safety properties are typical constraints of object histories. History

invariants are typically stated in temporal logic. Temporal logic provides a means of specifying certain dynamic aspects of a possibly, highly complex system.

A declaration **c:C** declares c to be a reference to an object of the class **C**. A declaration does not mean that a new object is introduced nor does it mean that the object is initialized. Initialization of objects is attained by including the INIT schema of the class in an operation schema or the INIT schema of another class. Object creation is not directly provided in Object-Z. If this concept is necessary in a specification, it can be expressed by modeling a set of existing objects. Dot notation (C.var, C.op, ...) is used to refer to class properties. Objects may have object references as attributes and such references may either be individually named or occur in aggregates. If the references do not change, they can be declared as constants. However, this does not mean that the contents of the stacks remain constant.

The polymorphic operator (↓)defines the type of an attribute as any class which inherits from the specified base class. Operations applied to such an attribute must be polymorphic, i.e. regardless of the actual class of the object, the operation must be able to proceed. The containment operator (©)defines an ownership relationship between class instances. Operations may be combined using the Z schema calculus [17], including conjunction ($\wedge$) and sequential composition ($\fatsemi$). Object-Z provides two further operators: concurrency ($\|$) and angelic choice ([]). The concurrency operator merges state and predicates of operation schemas and identifies inputs and outputs. The angelic choice operator selects one of two operations according to which one can be performed. If both operations can be performed, a non-deterministic choice is made between the two.

4.0 FORMAL SPECIFICATION OF OUR MODEL

The formal specification of the secret sharing can be represented by introducing a set of registered users. The dealer is a trusted agent who mediates the secret sharing using private channels to communicate with the users and broadcasts information on a broadcast channel.

The information to be shared among the users are represented by a sequence of characters. We are not concerned with the internal detail of the text information. The encoded secret should be represented by a sequence of characters and it is specified as

$$[INFO]$$

When the text information is shared among different users, it changes to shared information and its construction is different from the text information. Hence we introduce

$$[SHARE]$$

which will be received by designated users only.

The response to any request is defined as Boolean, which is either true or false.

$$Boolean ::= true \mid false$$

The status of data in a communication channel can either be cleared, sent or received and is defined as STATUS

$$STATUS ::= clear \mid send \mid receive$$

To identify a user uniquely, two sets are introduced. Each of the participants of secret sharing has an identification number and a name. The set of unique identifiers, UID, is needed to identify a registered user.

$$[UID, NAME]$$

denotes the set of all possible identifications of users and a special case for one dealer.

The users can be represented by

$$USERS == \mathbb{P}(UID \times NAME)$$

A set of users forms a possible key group, who can reconstruct secret information. This set of users can be represented as

$$SharedUserGroup == \mathbb{P}\ USERS$$

The information of the SharedUserGroup set is to be known to the dealer for sharing the secret. The hashing of information can be abstracted by a total function of INFO and HASHVALUE so that INFO has only one HASHVALUE. The HASHVALUE can contain any value.

$$[HASHVALUE]$$

4.1 Public Key List

In the proposed specification of secret sharing scheme, the private channel uses the asymetric encryption technique. To accomodate the public keys for the users, a public key list is introduced. The public key list is open to all users. Any user can update his/her own public key, but can access anyone's public key for read only. There will be one instance of public key. Given that the type Key denotes the set of key values to which the public and private keys will be set.

$$[Key]$$

We define publicKey as Key. The Key is a large natural number.

$$publicKey == Key$$

The public key list can be represented as follows.

PublicKeyList

$\uparrow(ChangePublicKey, AddPublicKey)$

$PublicKey : \mathbb{P}\, publicKey;$
$userList : \mathbb{P}\, UID;$
$PublicKeyList : UID \nrightarrow publicKey$

$userList = \text{dom}\, PublicKeyList$

AddPublicKey

Δ
$(PublicKey, PublicKeyList)$
$user? : UID$
$PublicKey? : publicKey$

$user? \notin userList \land PublicKeyList' =$
$PublicKeyList \cup \{user? \mapsto PublicKey?\}$

ChangePublicKey

$\Delta(PublicKey, PublicKeyList)$
$user? : UID$
$oldpublicKey? : publicKey$
$PublicKey? : publicKey$

$user? \in userList \land PublicKeyList' =$
$PublicKeyList \oplus \{user? \mapsto PublicKey?\}$

GetPublicKey

$user? : UID$
$PublicKey! : \mathbb{P}\, publicKey$

$user? \in userList \land PublicKey! = \text{ran}$
$(\{user?\} \lhd PublicKeyList)$

isMember

$user? : UID$

$user? \in userList$

$getpublicKey \mathrel{\widehat=} GetPublicKey$
$changepublicKey \mathrel{\widehat=} ChangePublicKey$
$addpublicKey \mathrel{\widehat=} AddPublicKey$
$isMember \mathrel{\widehat=} isMember$

In the process domain, there will be only one public key list. State component *PublicKeyList* is declared as a partial function rather than a total function. The *AddPublicKey* adds a user's public key to the list if the user is not already enlisted. In some occasions, public key of a user requires to be changed and this change is performed if the user is in the list. To have a public key of a user, the user should be on the list, then only it can be extracted by restricting the user in *PublicKeyList*.

4.2 Private Channel

The private channels are for secure communication between the dealer and the users. The information being sent over the private channel can only be opened by the recipient using its private key. When a piece of information is required to send to a given user, the information is encrypted by the public key of the recipient. As the system uses a private channel to transfer data between the dealer and a particular user, this can be represented by the following specification.

PrivateChannel

$PKList : PublicKeyList;$
$publicKey : Key;$
$info : INFO;\ EcdInfo : INFO;$
$uid : UID;\ prvKey : Key;$
$infoStat : STATUS$

INIT

$infoStat = clear$

setInfo

$\Delta(info)$
$info? : INFO$

$infoStat = clear$
$prvKey' = prvKey \land infoStat' = send$

setPrvKey

$publicKey? : Key;$
$prvKey? : Key$

$prvKey' = prvKey?$

sendInfoPriv

$Ruid? : UID;$
$EcdInfo? : INFO$

$infoStat = send$
$EcdInfo? \neq info \land uid \neq Ruid?$
$\land\ infoStat' = clear$

receiveInfo

$recInfo? : INFO;$
$Ruid? : UID;$
$Suid? : UID;$
$EcdInfo! : INFO$

$infoStat = clear$
$EcdInfo' = EcdInfo! \land EcdInfo! =$
$recInfo? \land uid = Ruid? \land infoStat = receive$

decodeInfo

$Ruid? : UID;$
$EcdInfo? : INFO;$
$info! : INFO$

$uid = Ruid? \land info! \neq EcdInfo?$

$setPrvKey \mathrel{\widehat=} setPrvKey \bullet PKList.chgPubKey$
$receiveInfo \mathrel{\widehat=} receiveInfo \bullet PKList.isMember$
$sendInfo \mathrel{\widehat=} sendInfoPriv \bullet PKList.getpubKey$

In a private channel, information should be encrypted by the public key and at the receiving end, encrypted information received can only be decrypted by using the recipient's private key. This capability leads to the requirement of a private and public key pair. To change a private key, it is necessary to change the public key in the public key list. When a piece of information is ready to be sent privately, the information and the encrypted information cannot be the same. To abstract out the method of encryption, the model does not mention the mechanism. When the information is received, the encoded information can be updated, as in the private channel, only encoded information can be received. The information received should address the owner of the private channel. The received information is decoded by the decryption algorithm which is not stated to abstract out in decodeInfo. The *PKList* for the channel must be initialized at the begining of the session. The dealer uses the private channel to send data to the user. The specific private channel is only known to the dealer and the specific user.

4.3 Broadcast Channel

The broadcast channel is used to transmit data to all participants so that the receiver can reconstruct some information which can be used to verify that the dealer is fair and participating members are not being cheated. In the broadcast channel, the same information is to be sent to all the participants of the secret sharing scheme. As we are not modeling a multicast broadcast scheme, information is sent to each participant independently. the operation *receiveInfo* just adds incoming information to its state.

BroadcastChannel

$info : INFO;$
$userlist : \mathbb{P}\ UID$

sendinfo

$info? : INFO$
$user? : UID$

$user? \in userlist$

broadcast

$info? : INFO;$
$u! : UID$

$\forall u : userlist \bullet u! = u$

receive

$info! : INFO$

$broadcast \mathrel{\widehat=} broadcast \bullet sendinfo$
$receive \mathrel{\widehat=} receive$

4.4 User

The user in this model is defined to send and receive information over both private channel and broadcast channel. The user has to verify the share received from the dealer. He/she also receives information from other users who are in the public key list. The broadcast channel is used for disseminating information to all the participants in secret sharing scheme. The private channel is used for communication between the dealer and the participant secretly. The user sends and receives information privately using the private channel. While sending information over the private channel, the method *sendInfoPriv* is used. To receive information over the private channel, *receive* method is used. To receive information over the broadcast channel, *receiveb* method is used. The *verifyShare* is used to check that the hashed value of the received share and the broadcasted hashed value by the dealer is the same. The *verify* method is used when information is to be disseminated over the broadcast channel, and only the information that is in user's possession is to be broadcasted.

User

$pchannel : PrivateChannel;$
$bchannel : BroadcastChannel;$
$MySharedUserGroup : \mathbb{P}\ SharedUserGroup;$
$Hash : INFO \rightarrow HASHVALUE;$
$hShr : SharedUserGroup \nrightarrow HASHVALUE;$
$MyShare : \mathbb{P}\ INFO$

broadcast

$share? : INFO$

receive

$share! : INFO$

sendPriv

$Ruid? : UID;$
$share? : INFO;$
$aSharedUsrGrp : SharedUserGroup;$
$aUser : USERS$

$aSharedUsrGrp \in MySharedUserGroup$
$\land\ Ruid? \in \mathrm{dom}\ aUser$
$\land\ aUser \in aSharedUsrGrp$

verify

$info? : INFO;$
$answer! : Boolean$

$info? \in MyShare \land answer! = true$

verifyShare

$info? : INFO;\ Ruid? : UID;$
$xUser : USERS;\ answer! : Boolean;$
$xSharedUsrGrp : SharedUserGroup$

$xSharedUsrGrp \in MySharedUserGroup$
$\land\ Ruid? \in \text{dom}\ xUser$
$\land\ xUser \in xSharedUsrGrp$
$info? \lhd Hash = xUser \lhd hShr$
$\land\ answer! = true$

$sendInfoPriv \mathrel{\widehat=} sendPriv \bullet pchannel.sendInfoPriv$
$broadcast \mathrel{\widehat=} broadcast \bullet bchanell.broadcastinfo$
$receive \mathrel{\widehat=} receive \bullet pchannel.receiveInfo$
$receiveb \mathrel{\widehat=} receive \bullet bchannel.receive$
$verify \mathrel{\widehat=} verify \land broadcast$
$verifyShare \mathrel{\widehat=} verifyShare$

4.5 Dealer

A dealer is modeled as an object that offers particular services, as specified by the class dealer. The dealer is actually a user, but when a user wants to share a secret among others, the user now plays the role of the dealer. At that time, the dealer will not receive any share of secret. The dealer has more responsibility than the user. The dealer starts its job when it receives a document to be shared. In verifiable secret sharing algorithm, information is shared. The dealer shares the information (secret) with a group of users who are composition from users known as *SharedUserGroup*. The dealer calculates all shares in *SharedUserGroup*. In the definition of *calculateShare*, the algorithm of sharing is abstracted out. The share should be different from information to be shared. The share for each participating user has to be sent privately. To receive information over the private channel and the broadcast channel, *receive* and *receiveb* are defined respectively.

Dealer

User

$MyInfo : INFO$

$MySharedUserGroup : \mathbb{P}\ SharedUserGroup$

calcShare

$group? : SharedUserGroup;$
$info? : INFO;$
$share! : UID \rightarrow INFO$

$group? \in MySharedUserGroup$
$\forall u : \text{dom}\ group? \bullet \text{dom}\ share! = u$
$\forall i : INFO \bullet i \in \text{ran}\ share!$
$\land\ i \neq info?$

calcHash

$info? : INFO;$
$HashValue! : HASHVALUE$

$HashValue! = info? \rhd Hash$

$sendPriv \mathrel{\widehat=} calcShare \bullet sendPriv$
$\bullet\ pchannel.sendInfopriv$
$brdcast \mathrel{\widehat=} brdcast \bullet bchanell.broadcastinfo$
$brdHash \mathrel{\widehat=} calcHash \bullet bchannel.broadcastinfo$
$receive \mathrel{\widehat=} receive \bullet pchannel.receiveInfo$
$receiveb \mathrel{\widehat=} bchannel.receive$

The dealer knows all the private channels between the dealer and the users; but, one user can know only the private channel directed towards the dealer.

4.6 Proposed Specification of Secret Sharing Scheme

We define a secret sharing scheme as a class which contains one dealer and a set of users. To define temporal specification of the secret sharing scheme, we follow $\|$ operation defined in Object-Z. The dealer first calculates share and sends privately to all the users. At the same time, all the users receive information secretly over the private channel between the dealer and the particular user. The dealer broadcasts hashed value of share for all the users and all the users receive all information broadcasted by the dealer simultaneously. All the users verify the share they receive and broadcast the result. This time the dealer receives all the responses from the users over the broadcast channel.

SecretSharingScheme

$myDealer : Dealer;$
$myUser : \mathbb{P}\ User$

distribSecret

$myDealer.sendSharePrivately$

secretReceive

$\forall i : User \bullet i.receive \land i \in myUser$

brdcastHash

$Dealer.broadcastHash$

hashReceive

$\forall i : User \bullet i.receiveb \land i \in myUser$

brdcastVerify

$\forall i : User \bullet i.verifyShare$

verifyReceive

$myDealer.receiveb$

$step1 \mathrel{\widehat=} distribSecret \parallel secretReceive$
$step2 \mathrel{\widehat=} brdcastHash \parallel hashReceive$
$step3 \mathrel{\widehat=} brdcastVerify \parallel verifyReceive$

In this context, the cocept of combine object has to be introduced. The role of the *combine* object is to collect shares and checks whether they lie in the perfect sharing scheme. If a share does not comply with the sharing scheme, the *combine* object discards the share. Using the valid shares, the *combine* object recombines the shares and extracts the secret.

Combine

Dealer

$share$: seq $INFO$;
$secret$: seq $INFO$;
$limit$: $\mathbb{N}$;
$bchannel$: $BroadcastChannel$;
$MySharedUserGroup$: $\mathbb{P}$ $SharedUserGroup$

INIT

$secret = \varnothing$

getShare

$\Delta(share)$
$group?$: $SharedUserGroup$;
$share?$: $UID \rightarrow INFO$

$group? \in MySharedUserGroup$
$\forall u$: dom $group?$ • dom $share? = u$
$\forall i : INFO$ • $i \in$ ran $share? \land i \neq info?$

reCombine

$\Delta(secret)$

$\#share \geq limit$

$collectShare \mathrel{\widehat=} user? \bullet getshare$
$reCombineShare \mathrel{\widehat=} reCombine$

5.0 CONCLUSION

The case study demonstrates that formal methods can be useful for describing the specification of a secure information system like secret verifiable information sharing. The case study shows that Object-Z can be used to structure a secure information system to fulfill its security requirements, and at the same time, gracefully isolates the underlying VSS protocol from the specification. The proposed generic formal model of secure ISs can be used to support software engineers in reusing existing system designs and in developing new systems.

ACKNOWLEDGEMENTS

We would like to thank Wendy Johnston whose Wizard 1.5, the type checker for object-Z, is used to type-check the specifications presented in this paper. The specification is written in LaTeX document setting format and passed directly to Wizard to be type-checked. Wizard is actually a type-checking system for Object-Z specifications that work with the LaTeX text formatter on Unix platforms. The wizard program works on the same input file as LaTeX. It extracts and parses the formal text to check for syntax errors and then type-checks the specification to find typing and scope errors. It is available from the Software Verification Research Center's ftp area. This ftp area is located at: ftp://ftp.cs.uq.edu.au/pub/SVRC/software .

REFERENCES

[1] J. Bowen and M. Hinchey, "The Use of Industrial–Strength Formal Methods," In *21st International Computer Software and Application Conference, COMPSAC'97*, 1997, pp. 332–337, IEEE Computer Society Press.

[2] IEEE, "IEEE Standard Glossary of Terminology," In *IEEE Software Engineering Standards Collection*, IEEE press, 1991.

[3] A. Shamir, "How to Share a Secret," *Communications of the ACM*, Vol. 22, No. 11, 1979, pp. 612–613.

[4] G. Blakley, "Safeguarding Cryptographic Keys," In *National Computer Conference*, Vol. 48, 1979, pp. 313–317, AFIPS.

[5] A. S. M. Ito and T. Nishizeki, "Secret Sharing Schemes Realizing General Access Structure," In *Global Telecommunication Conference, Globecom'87*, 1987, pp. 99–102.

[6] J. Benaloh and J. Leichter, "Generalized Secret Sharing and Monotone Functions," In *Advances in Cryptology-CRYPTO'88*, Vol. 403 of *LNCS*, 1988, pp. 27–36, Springer-Verlag.

[7] C. Asmuth and J. Bloom, "A Modular Approach to Key Safeguarding," *IEEE Transaction of Information Theory*, Vol. IT-30, 1983, pp. 208–210.

[8] J. G. E. Karnin and M. Helman, "On Sharing Secret Systems," *IEEE Transaction of Information Theory*, Vol. IT-29, 1983, pp. 35–41.

[9] E. F. Brickell and D. Davenport, "On the Classification of Ideal Secret Sharing Schemes," In *Advances in Cryptology-CRYPTO'89*, Vol. 435 of *LNCS*, 1990, pp. 278–285, Springer-Verlag.

[10] L. G. C. Blundo, A. De Santis and U. Vaccaro, "On the Information Rate of Secret Sharing Schemes," In *Advances in Cryptology-CRYPTO'92*, Vol. 740 of *LNCS*, 1992, pp. 148–167, Springer-Verlag.

[11] A. Beimel and B. Chor, "Universally Ideal Secret Sharing Schemes," In *Advances in Cryptology-CRYPTO'92*, Vol. 740 of *LNCS*, 1992, pp. 183–195, Springer-Verlag.

[12] H. Krawczyk, "Secret Sharing Made Short," In *Advances in Cryptology-CRYPTO'93*, Vol. 773 of *LNCS*, 1994, pp. 136–146, Springer-Verlag.

[13] S. M. B. Chor, S. Goldwasser and B. Awerbuch, "Verifiable Secret Sharig and Achieving Simultaneity in Presense of Faults," In *26th IEEE Symposium on Foundations of Computing Science*, 1985, pp. 383–395, IEEE.

[14] I. D. Ronald Cramer and U. Maurer, "Span Programs and General Secure Multi-Party Computation," *Brics*, 1997.

[15] G. R. Roger Duke, Paul King and G. Smith, "The Object-Z Specification Language: Version 1.," Tech. Rep. Technical Report 91-1, Software Verification Research Centre, Department of Computer Science, The University of Queensland, Australia, April 1991.

[16] G. R. Roger Duke and G. Smith, "Object-Z: A Specification Language Advocated for Description of Standards," Tech. Rep. Technical Report 94-45, Software Verification Research Centre, Department of Computer Science, The University of Queensland, Australia, December 1994.

[17] J. M. Spivey, *The Z Notation: A Reference Manual.* Computer Science, Pretice Hall International, 2 ed., 1992.

BIOGRAPHY

Kok Meng Yew was an Associate Professor of the Faculty of Computer Science and Information Technology, University of Malaya.

Mohammad Zahidur Rahman received B.Sc. degree in Electrical and Electronics from Bangladesh University of Engineering and Technology, Bangladesh in 1986 and M.Sc. in Computer Science and Engineering from the same university in 1989. He is currently pursuing his PhD in the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. His research interests are security in E-commerce, electronic voting and Formal Methods.

Sai Peck Lee obtained her Master of Computer Science from University of Malaya in 1990, her D.E.A of Computer Science from University of Paris VI Pierre et Marie Curie in 1991 and her PhD of Computer Science from University of Paris I Panthéon-Sorbonne in 1994. She is an Associate Professor at the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia.