A Proxy Automatic Signature Scheme Using a Compiler in Distributed Systems for (Unknown) Virus Detection

Hwang, Shin-Jia and Chen, Kuang-Hsi
Department of Computer Science and Information Engineering,
TamKang University, Tamsui, Taipei Hsien, 251, Taiwan, R.O.C.
sjhwang@mail.tku.edu.tw and kuanghsi@gamil.com

Abstract

To detect any (unknown) virus, automatic signature schemes are proposed to be embedded in honestly-made compilers. But compiling load is centralized on the compiler makers. To distribute compiling load with the help of distributed servers, proxy automatic signature schemes are proposed for the distributed compilers. However, Lin and Jan's proxy automatic signature scheme is insecure and has length restriction of source programs. To remove these flaws, Hwang and Li also proposed their scheme. However, two signatures are used for the agreement of compiler makers and servers, respectively. But only the signature for the proxy agreement of compiler makers can be validated by anyone. To remove this inefficient flaw, a new efficient proxy automatic signature scheme is proposed. Except the efficient advantage, the proxy agreement being researched both by the compiler maker and servers can be validated by anyone at the same time. Only one signature is used to show the agreement. The correctness of compilers and executable programs can be validated without releasing source codes. Moreover the moderator can easily find out infection sources.

Keywords: *Compilers, distributed system, computer virus, digital signature, proxy signatures, automatic signatures.*

1. Introduction

For the time being, there is more and more convenient in data transmission with the Internet. Unfortunately, Internet is an insecure environment, so the computer viruses, crackers, and many computer crimes may damage or modify your data in computer. Recently the computer virus has become a serious security problem. Some anti-virus packages are adopted to detect the existence of computer virus. But unknown virus cannot be detected by the anti-virus packages. A new concept for virus detection has been proposed by adopting digital signatures. Since digital signatures can guarantee the integrity of signed files, the signature is used to ensure that the executable file is not infected by (unknown) virus.

In 1993, Okamoto first proposes a cryptographic solution for detecting virus by digital signature schemes [8]. By the way of checking the consistency of the original executable program with its corresponding signature to check whether or not the executable program is infected by virus.

Another cryptographic scheme proposed by Usuda et al. [11] is the automatic signature scheme using compilers. When a compiler maker adopts honestly-made compilers to compile source programs, the compiler automatically produces the executable program and companying signature without interrupt. The automatic signature scheme can reduce the probability of infecting virus because the correctness of executable programs is validated by the companying automatic signatures. Thus any virus infection can be found out after the verification of the automatic signatures.

However, the compiler maker becomes the compiling bottleneck because any compiling tasks should be performed by the compiler maker. To distribute the compiling load, Lin and Jan [5] proposed their automatic signature scheme using a compiler in distributed systems. Because Lin and Jan's scheme adopts the signature scheme with message recovery mode, their scheme has length restriction for the source programs. Moreover, their scheme is insecure [10]. To overcome these flaws, Hwang and Li [3] proposed their proxy automatic signature scheme based on the concept of proxy signature schemes [6]. In Hwang and Li's scheme, one signature is used for the proxy agreement of servers while another signature is used to show the proxy agreement of compiler makers. It is inefficient to use two signatures for the proxy agreement between compiler makers and servers. Moreover, except compiler makers, no one can

Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)
1550-445X/05 $20.00 © 2005 IEEE

validate the agreement of servers.

To overcome the above problems in Hwang and Li's scheme, a new proxy automatic signature scheme using a compiler is proposed in Section 3. In the next section, the basic assumption and model for the new scheme is described. In Section 4, some security issues and discussions are given. The final section is our conclusion.

2. Our Basic Assumptions and Model

The basic assumptions relative our basic model is first given in the first subsection. Then the underlying basic model for our scheme is given.

2.1 Assumptions

Our assumptions are classified into three classes. One class is the set of assumptions about virus, one class is the set of assumptions about the compiler maker, and one class is the set of security assumptions [3, 5, 10-11]. Three classes are described, respectively.

Class 1 (About viruses):

This class contains three assumptions about the computer virus's operations.

(1) Viruses infect only executable files, not pure text files.
(2) Viruses damage or modify both executable files and text files.
(3) The priority of execution for compiler is higher than the execution of any virus such that virus cannot interrupt it.

Class 2 (About compiler maker)

This class contains two assumptions about the compiler maker's operation.

(1) The compilers are honestly created by the compiler maker.
(2) Compiler maker cannot refuse to reply to the requester's questions per requests.

Class 3 (About security)

This class contains three assumptions about the security of our scheme.

(1) The discrete logarithm program is a computational hard problem.
(2) The one-way hash function is strong and against finding the collisions.
(3) The distributed systems must properly execute the verification program.

2.2 Our Basic Model

The basic model for our protocol is described here. In the basic model, there are six kinds of participates: a trusted third party (TTP for short), the compiler maker (U_m), the server (U_s), the requester (U_r), customers, and a trusted moderator. Our basic model consists of five phases: Initialization phase, compiler maker-server authorization phase, sever-requester execution phase, custom verification phase, and judge phase. These phases are described, respectively.

In the initialization phase, TTP constructs the system-wild parameters and some public cryptographic functions. Each legal user randomly generates his/her private key and computes the corresponding public key. The public key of each user is certificated by TTP.

In the compiler maker-server authorization phase, a server U_s requests the compiler maker U_m a compiler, named C_R, in order to provide the compiling service on behalf of the compiler maker. The compiler maker U_m provides the server U_s with the compiler C_R which can automatically and non-interruptively generate the signature both on source programs and the corresponding executable file generated by C_R. In order to show the agreement of the compiler maker and the server, a suitable proxy delegation algorithm is cooperatively executed both by the compiler maker and the server. Then the server will obtain a proxy private key which is only computed by the server. At the same time the compiler is integrated with a proxy automatic signature generation algorithm.

The server U_s uses the verifiable compiler C_R to compile the source program M sent from the requester U_r. Then the generated executable program E is sent to the requester U_r. The requester U_r sends the executable program E when the customer buys it. In our model, servers are distributed over the Internet. When a server compiles a source program, the server adopts the compiler C_R to automatically create both the executable program and its signature. The modification of signed compilers, source programs and executable programs can be detected by checking the consistency of their accompanying signatures. To reduce the storage of signatures, being inspired of the concept of multi-proxy multi-signature schemes [4], the proxy certificate between servers and compiler makers is the signature generated by the cooperation of servers and compiler makers. Then the proxy certificate shows that the proxy agreement is made by both the original signer and servers.

Definition (Discrete-logarithm-based signature scheme [1, 2, 7])

Suppose that the signer is U_i with the public key y_i and private key x_i. A discrete-logarithm-based (DL for short) signature scheme is a signature scheme based on the discrete logarithm problem. In a DL signature scheme, there is a singing algorithm $(r, s)= sig_{x_i}(M)$ and a verification algorithm $ver_{y_i}((r, s), M) \in \{true, false\}$, where M is a message.

An example of DL signature scheme

Some DL signature schemes are proposed [1, 2, 7]. Here, the DSA in [2] is described. In DSA, TTP selects two large primes p and q satisfying $q \mid p$-1 and an element $g \in Z_p^*$ with order q. A user U_i selects a private key $x_i \in Z_q^*$ and computes a public key $y_i = g^{xi} \bmod p$. To sign a message m, the user chooses a random integer $v \in Z^*_q$ and computes $r = (g^{v} \bmod p) \bmod q$ and $s = r^{-1}(m + rx_i) \bmod q$. The signature for the message m is (r, s). To verify the signature (r, s) for m, a verifier checks whether or not $r \equiv ((g^{ms^{-1}} y_i^{rs^{-1}}) \bmod p) \pmod q$. Hereafter, the system-wild parameters p, q, and g are also suitable for the other DL signature scheme. The public key and private key of the user U_i are also y_i and x_i, respectively.

Definition (Automatic signature schemes)

Suppose that the signer is U_i with the public key y_i and the private key x_i. An automatic signature scheme is a signature scheme with the automatically signing algorithm $S = \text{autosig}_{x_i}(M)$ and the verification algorithm $\text{ver}_{y_i}(S, M) \in \{true, false\}$, where M is a message and S is the signature. The automatically signing algorithm $\text{autosig}_{x_i}(M)$ can be embedded into a executable program in such a way that the output M of the executable program and the signature generation $S = \text{autosig}_{x_i}(M)$ are executed sequentially without any interrupt.

An example of automatic signature schemes

In essential, an automatic signature scheme is a signature scheme embedded in to an executable program. If the underlying signature scheme is a DL signature scheme, then $S = (r, s) = \text{autosig}_x(M)$ and $\text{ver}_y((r, s), M) \in \{true, false\}$. To set up system parameters, TTP generates the system-wild parameters p, q, and q. TTP also publishes a cryptographic one-way function $h(\cdot)$.

Suppose that U_A writes a source program M and needs U_B's help to compile her source program with signature for the executable program. U_A first sends a request and his/her source program M to U_B. After getting the executable program E on M by compilers, the automatic signature signing algorithm $\text{autosig}_{x_B}(E \| M)$ is immediately performed to generate the automatic signature (r, s) on E and M.

Definition (Proxy Signature scheme)

In the proxy signature scheme [6], an original signer is allowed to authorize a designate person as his proxy signer. Then the proxy signer is able to generate signatures on messages on behalf of an original signer. Suppose that the original signer is U_O and the proxy signer is U_P. The authorizing algorithm $C = \text{Autho}(w, x_O, y_O, x_P, y_P)$, can generate a proxy certificate C on the proxy warrant w for the proxy signer U_P. Then everyone can validate C by using $\text{CertV}(w, C, y_O, y_P)$.

To generate the signature on the message M, the proxy signer can use the proxy signing algorithm $S = \text{ProxySig}(C, x_P, y_P, M)$ to generate the proxy signature (S, C) of M. Then the proxy signature is (S, C) can be validated by $\text{ProxysigV}(S, C, y_O, y_P, M)$

An example of a proxy signature scheme

Being inspired of the multi-proxy multi-signature scheme in [4], our proxy signature scheme is proposed below. One of the advantages of the scheme in [4] is that the proxy authorization is based on the agreement from not only original signers but also proxy signers. Hence it is efficient to check the agreement of original signers and proxy signers at the same time. The system parameters and public functions are the same as those in the DL signature scheme.

The authorizing algorithm $C = \text{Autho}(w, x_O, y_O, x_P, y_P)$ is given here, where w is the proxy warrant. To construct proxy authorization, U_O and U_P first select random numbers k_O and $k_P \in Z^*_q$, respectively. U_O and U_P compute $K_O = g^{k_O} \bmod p$ and $K_P = g^{k_P} \bmod p$, respectively. Then U_P sends K_P to U_O and U_O sends K_O to U_P. Both U_O and U_P compute $K = K_O \times K_P \bmod p$ by themselves. Then U_O finds $v_O = h(w)x_O y_O + k_O K \bmod q$ and U_P finds $v_P = h(w)x_P y_P + k_P K \bmod q$. U_O sends v_O to U_P while U_P sends v_P to U_O. U_O validates v_P by checking $g^{v_P} \equiv (y_P{}^{y_P})^{h(w)} \times (K_P)^K \pmod p$ and U_P validates v_O by checking $g^{v_O} \equiv (y_O{}^{y_O})^{h(w)} \times (K_O)^K \pmod p$. Finally both they obtain the proxy certificate $C = (K, V)$, where $V = v_O + v_P \bmod q$. The proxy certificate $C = (K, V)$ can be validated by adopting the equation $g^V \equiv (y_P{}^{y_P} \times y_O{}^{y_O})^{h(w)} \times (K)^K \pmod p$. Therefore, $\text{CertV}(w, C, y_O, y_P)$ is to check whether the equation $g^V \equiv (y_P{}^{y_P} \times y_O{}^{y_O})^{h(w)} \times (K)^K \pmod p$ holds or not.

To generate the signature on a message M, the proxy singer U_P first selects a random integer t and computes $r = g^t \bmod p$. Then U_P computes $s = (Vt + x_P y_P rh(M)) \bmod q$. Then the proxy signature of the message M is $(w, (K, V), (r, s))$. The proxy signature is verified by using the equations $g^V \equiv (y_P{}^{y_P} \times y_O{}^{y_O})^{h(w)} \times (K)^K \pmod p$ and $g^s \equiv r^V (y_P{}^{y_P})^{rh(M)} \pmod p$.

3. Our Realization of Our Proxy Automatic Signature Scheme Using a Compiler in Distributed Systems

Our realization is described phase by phase in the following.

Initialization phase

In this phase, TTP and each participate generate their parameters listed below.

(1) TTP selects a public large public prime p, a public prime factor q of p-1, and a public element $g \in Z^*_p$ with order q.
(2) TTP publishes a cryptographic hash function $h(\cdot)$.
(3) Each participate U_i selects his/her private key $x_i \in Z^*_q$ and computes his/her public key $y_i = g^{x_i} \bmod p$. Then the public key y_i is certificated by TTP.

Some notations used in our realization are defined below.

C_R: C_R denotes the executable compiler created by the honest compiler maker.

w: w denotes the proxy warrant w between the compiler marker and servers. The proxy warrant w specifies the necessary proxy details. The proxy details at least include the identities of the original signers and proxy signers, the public keys of the original singer and proxy signers, the compiler C_R with the corresponding information and the authorization period.

P: P denotes the source program sent from the requester U_r.

E: E denotes the executable program on the source program M from the requester U_r.

Compiler maker-Server authorization phase

Suppose that both the compiler maker U_m and the server U_s make an agreement of the proxy warrant w in advance. In order to show that they both agree of proxy authorization, both U_m and U_s cooperatively generate the proxy certificate for the proxy signer (server U_s) on the proxy warrant w and the compiler C_R. At the same time, the compiler maker U_m sends the compiler C_R requested by the server U_s. By using the compiler C_R, the server is authorized to automatically sign source programs and executable programs compiled by C_R.

Step 1: The compiler maker U_m selects a random integer $k_m \in Z^*_q$, computes $K_m = g^{k_m} \bmod p$ and sends K_m to the server U_s. At the same time, the server U_s selects a random integer $k_s \in Z_q^*$ and computes $K_s = g^{k_s} \bmod p$. Then K_s is sent to the compiler maker U_m.

Step 2: The server U_s and the compiler maker U_m computes $K = K_m K_s \bmod p$.

Step 3: The compiler maker U_m computes $v_m = h(w \| h(C_R)) x_m y_m + k_m K \bmod q$. U_m sends v_m and $h(C_R)$ to the server U_s.

Step 4: The server U_s validates v_m and $h(C_R)$ by the equation $g^{v_m} \equiv y_m^{y_m^{h(w\|h(C_R))}} K_S^K \pmod p$. If the above equation holds, U_s computes $v_s = h(w \| h(C_R))\, x_s\, y_s + k_s\, K \bmod q$, and sends v_s to the compiler maker U_m.

Step 5: Compiler maker U_m verifies the correctness of v_s by the equation $g^{v_S} \equiv y_s^{y_S^{h(w\|h(C_R))}} K_s^K \pmod p$. If the above equation holds, the compiler maker U_m send the server U_s the compiler C_R .

Step 7: Server U_s checks the correctness of C_R by using the digest $h(C_R)$. Both U_s and U_m computes $V = v_m + v_s \bmod q$.

At last the proxy certificate on the proxy warrant w and the compiler C_R is (K, V). Both the compiler maker and the server reach an agreement to authorize the server as a compiler proxy agent.

Server-Requester execution phase

The requester U_r sends the request and the source program P to the server U_s in order to compile P with the aid of the server U_s. Then U_s sends the executable program E for P and the corresponding automatic proxy signature to the requester U_r.

Step 1: The requester U_r generates his/her digital signature $(e_r, s_r) = sign_{x_r}(h(U_r \| P))$ adopting a DL signature scheme [1-2, 7]. Then he/she sends $(U_r, P, (e_r, s_r))$ to the server U_s.

Step 2: The server U_s validates (e_r, s_r) on the digest $h(U_r \| P)$ by performing $ver_{y_r}((e_r, s_r), h(U_r \| P))$.

Step 3: If (e_r, s_r) is correct, then U_s first validates his/her compiler C_R by the equation $g^V \equiv (y_s^{y_s} \times y_m^{y_m})^{h(w\|h(C_R))} \times K^K \pmod p$. The server U_s feeds his/her private key x_s, the proxy certificate (K, V) and the program P into the validated compiler C_R. After generating the executable program E on P, the compiler C_R immediately and automatically generates the signature (R, S) on the digest $h(E, K, V, h(P))$ by adopting a suitable DL signature generation algorithm. During the compiling process, the code and data memory belonging to C_R should be protected from any unauthorized modification except C_R and the operation system. Finally, U_S sends

$(w, (K, V), (R, S))$, $h(C_R)$, and E to the request U_r.

After obtaining $(w, (K, V), E, (R, S))$ from the server Us, the requester U_r checks the multi-proxy multi-signature as the following steps.

Step 1: Verify the warrant w and the certificate (K, V) by the equation $g^V \equiv K^K [y_s y_s \, y_c{}^{y_c}]^{h(w, h(C_R))} \pmod p$

Step 2: Check the correctness of the multi-proxy multi-signature (R, S) by the equation $g^S \equiv RV[y_c{}^{y_c}]^{Rh(E, K, V, h(P))} \pmod p$.

Customer verification phase

The customer U_c sends the request to the requester U_r for executable program E, and the request U_r send the executable program E to the customer U_c.

The customer got the executable program and its corresponding signature from the requester. After the customer receives $(w, h(P), h(C_R), (K, V), E, (R, S))$, he/she verifies it in two steps.

Step 1: Verify the warrant w, $h(C_R)$, and the certificate (K, V) by the equation $g^V \equiv K^K \, (y_m{}^{y_m} \, y_s{}^{y_s} s)^{h(w||C_R)} \pmod p$. If the equation does not hold, reject the proxy signature (R, S)

Step 2: Check the correctness of the proxy signature (R,S) and the executable program E by the equation $g^S \equiv R^v (y_s{}^{y_s} s)^{R \, h(E)} \pmod p$. If the equation holds, the executable program E had not been modified, and it can be accept.

Judge phase

When the customer finds the executable program from the request infected by virus, several possible situations are considered.

(1) The server adopts an infected compiler to generate executable programs.
(2) The request may write a program with virus.
(3) The executable program may be infected virus in customer's computing environment.

First of all, the customer sends $(w, h(P), h(C_R), (K, V), E, (R, S))$ to moderator for detecting the source of virus. The moderator performs the verification $g^V \equiv K^K (y_m{}^{y_m} \, y_s{}^{y_s} s)^{h(w||h(C_R))} \pmod p$ to check whether the server performed the invalid compiler. If the verification equation does not hold, the executable file may be infected virus in server's computing environment. Otherwise, the server is stainless.

If the server used the correct and clean compiler, then the moderator will have a suspicion that the request may have a bad intention. That is the requester writes a virus program, and sends it to the customer. To judge this suspicion, the moderator checks the signature of the executable program E by $g^S \equiv R^v (y_s{}^{y_s} s)^{R \, h(E)} \pmod p$. If the equation holds, the moderator must ask the requester to provide the source program of E and checks whether or not the source program contains virus. Finally, neither the server or the requester produces the virus, the executable program may infect virus in the customer's computing environment.

4. Security Issues and Discussions

Our scheme has some advantages. In the Compiler maker-Server authorization phase of Lin and Jan's scheme, one signature is used to guarantee the agreement of the server while one signature is used to guarantee the agreement of the compiler maker and the correctness of the compiler sent to the server. It is expensive the check these thing by perform the signature verification twice. It is not reasonable that no one is able to find out the agreement of the server in the other phases. So, in our Compiler maker-Server authorization phase, one proxy certificate is used to guarantee the agreement both form the compiler maker and servers. Moreover, this certificate is also used to guarantee the correctness of the compiler for the server and anyone. In other words, the agreement from the compiler maker and server and the correctness of the compiler can be efficiently validated by anyone.

In our verification phase and judge phase, the requester's source code and the executable program can be verified without reveal the content of source code. It can protect the program author's privacy. But if the source program contains some malicious code to infect others, the original source code must be revealed to proof its legitimate.

In our scheme, the security is base on the proxy signature scheme and one-way hash function. There some possible attacks in our scheme are discussed below.

Security of proxy signature

The security of the proxy certificate (K, V) is considered. The malicious users want to forge the individual proxy certificate (V_m). To pass the verification equation $g^{v_m} \equiv y_m{}^{h(w||h(C_R))y_m} K_m{}^K \pmod p$, the forger must generate a forged individual certificate (K'_m, V'_m). If the value K'_m is determined first, it is hard to find V'_m for the DL problem $g^{V'_m} \equiv y_m{}^{h(w||h(C_R))y_m} K'_m{}^K \pmod p$. If the value of V'_m is determined first, it is hard to find the K'_m from the equation $K'_m \equiv [g$

$^{V'_m}(y_m^{h(w||h(C_R))y_m})^{-1}]^{K'_m{}^{-1}K_s^{-1}} \pmod p$. So, (K_m, V_m) can not be forged. By the similar analysis, it is also hard to forge another individual proxy certificate (K_s, V_s). Therefore, the proxy certificate (K, V) can't be forged.

The proxy signature can be used to protect the original signer and the proxy signer. The original signer must delegate authority to the proxy signer, and only the proxy signer can generate the proxy signature. If someone wants to generate the proxy signature without the original signer giving authority, he/she must have original signer's private key x_s to generate v_m. Without the original signer's private key, he/she must forge v_m and pass the equation $v_m = h(w||C_R)x_m y_m + k_m K \bmod q$. But to solve the equation is difficult, so the proxy signature can protect original signer.

On the other hand, the proxy signature can also protect proxy signer. The proxy signature of the message E is (R, S), $S = (V \cdot t + x_s y_s Rh(E)) \bmod q$. The original signer have no proxy signer's private key, so he/she can't generate the proxy signature that pass the equation.

Security of private keys

The malicious may want to forge the private key from the public key, he/she must solve the equation $y_i = g^{x_i} \bmod p$. But it is a discrete logarithm problem.

5. Conclusion

The automatic signature scheme using a compiler in distributed system is first proposed by Lin and Jan in 2000[5]. But their scheme cannot withstand forgery attack and has restriction of the source program [10]. Although Hwang and Li [3] proposed their improvement, Hwang and Li's scheme is inefficient to use two signatures for the proxy agreement between compiler makers and servers. To remove this inefficient problem, a new proxy automatic signature scheme is proposed. By the aid of automatic signature schemes, any modification of original programs can be found in advance by verifying the signature of original programs. In our scheme, only one signature is used to show the agreement between the compiler maker and servers. Besides, in the server-requester execution phase, any DL signature scheme is suitable to adopt. This property makes our scheme more and more freely for many conditions. Moreover, the origin of the infection can be specified to identify the responsibility.

References

[1] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithm," *IEEE Transactions on Information Theory*, Vol. 31, No. 4, 1985, pp. 469-1985.

[2] FIPS PUB 186, February 1991, Digital signature Standard.

[3] S.-J. Hwang and E.-T. Li, "A Proxy Automatic Signature Scheme Using a Compiler in Distributed Systems," 2004 Information Security Conference, Taipei, Taiwan, R.O.C., Jun. 10-11, 2004, pp. 345- 352.

[4] S. J. Hwang, Chiu-Chin Chen, "New multi-proxy multi signature schemes," *Applied Mathematics and Computation* Volume: 147, Issue: 1, January 5, 2004, pp. 57-67.

[5] W.–D. Lin and J.–K. Jan, "An automatic signature scheme using a compiler in distributed systems," *IEICE Transactions on Communications*, Vol. E83-B No. 5, May 2000, pp. 935-941.

[6] M. Mambo, K. Usuda, E. Okamoto, "Proxy signatures: delegation of the power to sign message," *IEICE Transactions Fundamentals*, E79-A, No. 9, 1996, pp. 1338-1354.

[7] K. Nyberg and R.A. Rueppel, "Message recovery for signature scheme based on the discrete logarithm problem," Design, Codes and Cryptography, Vol. 7, No. 1-2, 1996 , pp.61-81.

[8] E. Okamoto, "Integrated security system and its application to anti-viral methods," Proc. 6th Virus and Security Conf, 1993.

[9] C. P. Schnorr, "Efficient identification and signatures for smart cards," Advances in Cryptology-CRYPTO'89, LNCS 435, Springer-Verlag, 1990, pp.239-252.

[10] Y.–M. Tseng, "Cryptanalysis and restriction of an automatic signature scheme in distributed systems," *IEICE Transactions on Communications*, Vol. E86-B No. 5, May 2000, pp. 1679-1681.

[11] K. Usuda, M. Mambo, T. Uyematsu, and E. Okamoto, "Proposal of an automatic signature scheme using a compiler," *IEICE Transactions Fundamentals*, Vol. E79-A, No. 1, pp .94-101, January 1996.