Anonymous Proxy Automatic Signature Schemes with Compiler Agents for (Unknown) Virus Detection

Shin-Jia Hwang* and Kuang-Hsi Chen

Department of Computer Science and Information Engineering, TamKang University, Tamsui, Taiwan 251, R.O.C.

Abstract

Many (proxy) automatic signature schemes are proposed to guard against the (unknown) virus infection with the help of honest compiler makers. In these proposed schemes, the used compiler agents' public keys should be certificated and maintained by verifiers. If verifiers only keep the compiler makers' public key, it is more convenient. So an anonymous proxy automatic signature scheme with compiler agents is proposed. In the new scheme, the compiler agents are anonymous and verifiers do not need to store compiler agents' public key. Moreover, verifiers can authenticate the source of received executable problems and detect compiler agents' deviation in advance. Our schemes are suitable for adopting any discrete logarithm based signature schemes. Our scheme has provides strong moderator's judgment to detect of virus infection sources.

Key Words: Compilers, Distributed System, Computer Virus, Digital Signature, Proxy Signature, Automatic Signature, Anonymous

1. Introduction

With the progress of network techniques, the Internet is more and more popular in our real life. By utilizing the convenience of the Internet, many services are realized on the Internet. For examples, electronic mails, electronic voting, bulletin board system, and world-wide-web are some popular services. Through the services on the Internet, Digital data can be exchanged and broadcasted over Internet.

Recently, because of unreliable environment of Internet, computer viruses have become a serious problem. Computer viruses can spread everywhere through Internet, and cause big damage to users' computer systems. The virus infection can be detected by checking the integrity of executable programs. In order to guarantee the integrity of executable programs, digital signature schemes are adopted. Any modification of signed original files is easily detected by checking the corresponding signatures. The virus detection by using signatures [1] is better than the detection by using anti-virus software, because anti-virus software cannot detect the infection caused by unknown virus [2–4].

Usuda et al. first proposed an automatic signature scheme to detect (unknown) computer viruses [5]. In their automatic signature scheme, the viruses can be detected in advance with the help of an honest compiler maker. In their scheme, the infection source can be found out. The compiler maker manages the compiling process and the automatic signature generation. Later, Lin and Jan [6] proposed an automatic signature scheme using a compiler in distributed system. The client-server model is more efficient, such that the compiler maker's load can be shared by distributed servers. Unfortunately, Tseng [7] pointed out that Lin and Jan's scheme is insecure against forgery attacks. Moreover, source programs have the length restriction. To remove these flaws, Hwang and Li [8] proposed their proxy automatic signature scheme based on the concept of proxy signature schemes. The proxy agreement between servers and compiler makers are guaranteed by two signatures in their scheme. But using two signatures between a compiler maker and a

*Corresponding author. E-mail: sjhwang@mail.tku.edu.tw server is inefficient. Moreover, only the compiler maker can validate the agreement of the server in their scheme. It is unfair for the server. To remove this inconvenience and unfairness, Hwang and Chen [9] proposed a new proxy automatic signature scheme. But, in Hwang and Chen's scheme, one flaw is that the customer cannot validate the real source of the executable program in advance. A customer may receive an executable program that is no written by legal requesters. Furthermore, the server's public key must be obtained and certificated to validate (proxy) automatic signatures. Since servers' public keys may be revoked or the authorized servers may be changed, it is inconvenient for verifiers to maintain the used authorized server's public keys.

To remove this flaw and inconvenience, an anonymous proxy automatic signature scheme with compiler agents is proposed to detect (unknown) virus. In the next section, the basic assumptions and models for the new scheme is described first. The same section also gives the security goals of an anonymous proxy automatic signature scheme with compiler agents. Because our scheme adopts Hwang and Chan's anonymous proxy signature scheme [10], their scheme is reviewed in Section 3. The new scheme and the corresponding security analysis are given in Sections 4 and 5, respectively. Section 6 is the performance analysis of the new scheme. The comparison among Lin-Jan's scheme, Hwang and Li's scheme, Hwang and Chen's scheme, and the new scheme are stated in Section 7. The last section is our conclusions.

2. Basic Assumptions and Models

The assumptions and models in our anonymous proxy automatic signature scheme are first mentioned. Then the security goals of our scheme are given.

The assumptions can be classified into the assumptions about computer viruses, the ones about compiler makers, and the ones about security.

Assumptions about computer viruses

- Viruses infect executable files but not text files.
- Viruses damage both executable and text files.

Assumptions about compiler makers

They honestly create compilers by finishing the following work.

- Create the system library, macro library, I/O library and include files.
- Sign all library files following every subroutine, procedure, and initial structure.
- Calculate the fingerprints of the included files with a one-way hash function and create the database of the fingerprints.
- Create a compiler including both a preprocessor and a linker for machine dependence and independence.

Assumptions about security

The following states three security assumptions of our scheme.

- The discrete logarithm program is the computational hard program.
- One-way hash functions are strong against finding the collisions.
- Underlying discrete logarithm based signature schemes are secure.
- The distributed systems must properly execute the verification program.

Client-server model

In order to distribute the compiling load of the compiler maker, our scheme adopts client-server model. In our scheme, there are six kinds of participates: compiler makers, servers, requesters, a trusted moderator, customers, and a trusted third party.

Compiler maker u_M

Compiler maker honestly creates compilers and is able to authorize an anonymous server to use his/her compiler to share compiling load.

Server u_S

A server first obtains the (anonymous) compiling proxy authorization from compiler makers. Then the (anonymous) server helps requesters compile source programs to generate executable programs and corresponding (anonymous) proxy automatic signatures.

Requester u_R

A requester writes and sends source programs to the server to compile source programs into executable programs. A requester then broadcasts his/her executable programs to customers.

Customer u_C

A customer gets an executable program and corresponding signatures from a requester. Then the customer detects whether or not the executable program is infected viruses with the help of the corresponding (anonymous) proxy automatic signatures.

Trusted moderators

When customers find that some executable program is infected with viruses, he/she needs the moderator's help to detect the infection source. If the infection source is some anonymous server, the trusted moderator can identify the server with the aid of compiler makers. If the infection source is the requester, the moderator can also prove this accusation. The moderator can also filter out the groundless accusation from customers.

Trusted third party

The trusted third party (TTP for short) certificates the public keys of all participates.

Figure 1 illustrates the client-server model and the relations among participates in our scheme. In Figure 1, the compiler maker honestly creates compilers, which also automatically generates signatures on its generated executable programs. The compiler maker may need many anonymous servers to share the compiling load, so the compiler maker authorizes these servers as its anonymous proxy agents by issuing anonymous proxy certificates. Each anonymous server accepts the compiling request and source programs from a requester. For the source programs, the authorized compiler being executed on the server generates the executable program and the corresponding anonymous proxy automatic signature at the same time. The server sends the executable program and proxy automatic signature to the requester. The requester first validates the anonymous proxy certificates to check the server's anonymous authorization. Then the requester checks the correctness of proxy automatic signatures to confirm the integrity of the executable programs. The requester also generates his/her signature on the executable programs and anonymous proxy automatic signatures. Then the requester sends his/her executable programs, anonymous proxy automatic signatures, and the requester's signatures to customer. Using anonymous proxy automatic signatures of executable programs, the customer checks the virus infection of executable

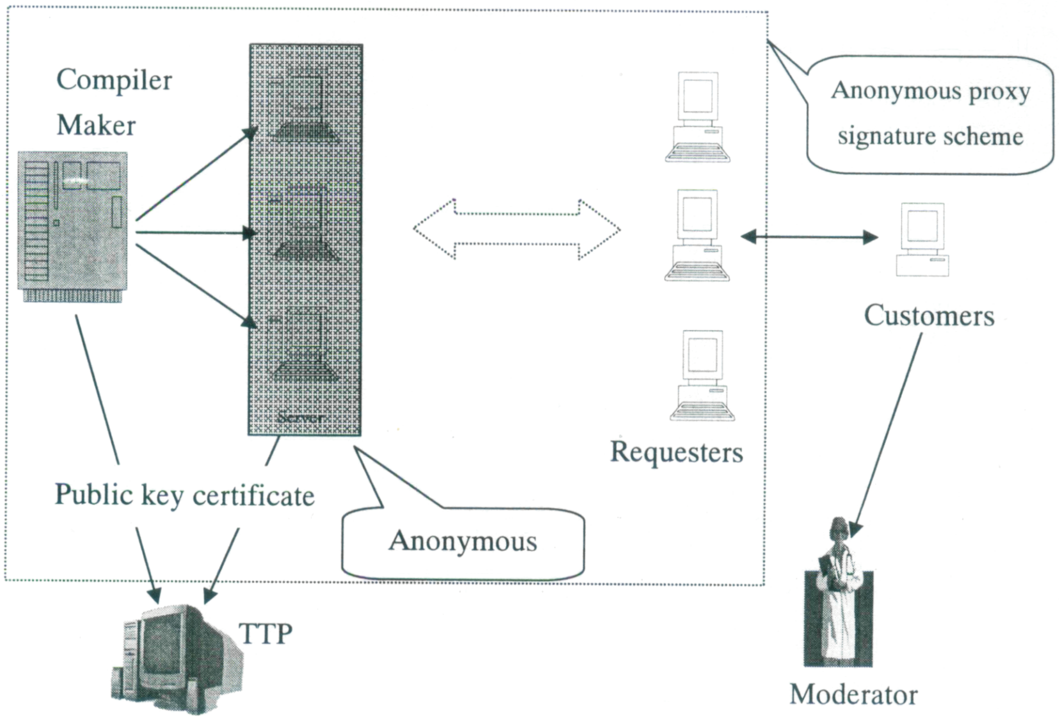

Figure 1. Client-server model in distributed systems.

programs. Customers use the requester's signatures to confirm the broadcasting source of the received executable programs. When virus infection is found, a moderator detects the infection source. In our scheme, all public keys of participates must be certificated by a trust third party.

Security goals in our scheme

The security goals for our scheme are divided into two classes. One class consists of the security goals about an anonymous proxy automatic signature schemes. Another class consists of the security goals about the virus detection and virus source identification. The security goals about an anonymous proxy automatic signature schemes are given below.

■ Anonymity: Besides the compiler maker, anyone cannot find the server's identity from anonymous proxy automatic signatures or proxy certificates.
■ Unforgeability: Only the authorized servers can generate valid anonymous proxy automatic signatures. Anyone cannot forge anonymous proxy automatic signatures even if he/she is a compiler maker.
■ Verifiability: Anyone can verify anonymous proxy automatic signatures that a valid server generated by using the server's proxy secret key.
■ Identifiability: The server's anonymity can be revoked by the compiler maker when disputes occur.
■ Undeniability: The server cannot deny his/her generation of anonymous proxy automatic signature. The compiler maker cannot deny the proxy authorization (or the generation of the proxy certificate) to the server.
■ Distinguishability: Signatures of the compiler maker, servers, requesters and proxy automatic signature signatures can be distinguished in polynomial time.
■ Server's protection: Beside the server itself, no one can impersonate the server to compile the source program and forge server's proxy automatic signatures.

The security goals about the virus detection and virus source identification are given below.

■ Server's deviation: A server cannot falsely incriminate any requester to write source programs with viruses.
■ Requester's protection: No one can impersonate the requester to distribute executable programs. The requester can promise which source codes are written by himself/herself. On the other hand, no one can success to falsely incriminate the requester writes a source code but he/she dose not.
■ Executable program's integrity: To guard against the virus infection, no one can modify the content of executable programs. Then anyone can verify the integrity of executable programs.
■ Source program's secrecy: Besides the requester and the server who compiling the requester's source code, no one can find the source code from the executable programs and anonymous proxy automatic signatures.
■ Requester's deviation: A requester cannot falsely incriminate any server to generate any executable program containing viruses.

3. Review of Anonymous Proxy Signature Scheme with Undeniable Agents

In Hwang and Chan's scheme [10], there are four participates in the anonymous proxy signature scheme: the original signer O, the proxy signer P, the verifier V, and the trusted third party TTP. In their scheme, there are four algorithms: $Auth_{APDW}$ (O, P, M_W, b, proxy certificate), $VerAuth_{APDW}$ (O, P, b′, proxy certificate), $VerCert_{APDW}$ (O, proxy certificate), and ID_{APDW} (TTP, O, y_P, M_W, b′, proxy certificate). By using these four algorithms, Hwang and Chan's scheme is realized.

An original signer O executes the algorithm $Auth_{APDW}$ (O, P, M_W, b, proxy certificate) to generate the proxy certificate on the proxy warrant M_W and the proxy public key Y to authorize an anonymous proxy signer P. Then the authorized secret value b is securely transmitted to the proxy singer P. Then the proxy signer P executes the algorithm $VerAuth_{APDW}$ (U_O, U_P, b′, proxy certificate) to validate the proxy certificate and the received authorized secret value b′. Anyone validates the proxy certificate and the proxy public key Y by the algorithm $VerCert_{APDW}$ (O, proxy certificate). When there are some disputes with the proxy signer, the original signer O can execute ID_{APDW} (TTP, O, y_P, b, proxy certificate) to revoke the anonymity of some proxy signer P, where y_P is the certificated public key of the proxy signer P.

Hwang and Chan's realization of anonymous proxy delegation scheme by warrant is described below. The following are some public system-wide parameters and functions in their scheme. The parameters p and q are two large prime numbers and $p = 2q + 1$. The parameter g is an element in Z_p^* with order q. A public function $h(\cdot)$ is a public cryptographic hash function. Each user U has a secret key $x_U \in Z_q^*$, and a certificated public key $y_U = g^{x_U}$ mod p. The proxy warrant M_W specifies the identity of the original signer O, the certificated public key y_O of the original signer O, the delegation period, and the other necessary proxy details. The four algorithms are described, respectively.

Auth$_{APDW}$(O, P, M_W, b, proxy certificate)

The original signer O randomly selects a secret integer $b \in Z_q^*$ and computes g^b mod p. Then O computes the proxy public key $Y = y_P \times (g^b)$ mod p. The original signer O uses his/her secret key x_O to generate the signature (r, s) on the digest $h(M_W, Y, h(bg^b \bmod p))$ by adopting a discrete-logarithm-based signature scheme. Then proxy certificate is $(M_W, Y, h(bg^b \bmod p), (r, s))$. Then the original signer O sends proxy certificate and the secret value b to the proxy signer O through a secure channel.

Suppose that the proxy signer received proxy certificate and the secret value b′, then the proxy signer executes the following algorithm to validate b′ and proxy certificate.

VerAuth$_{APDW}$(O, P, b, proxy certificate)

The proxy signer P first computes the proxy public key $Y' = y_P \times (g^{b'})$ mod p, and $h' = h(b'g^{b'} \bmod p)$. Then the proxy signer checks whether or not $h(M_W, Y, h(bg^b \bmod p)) = h(M_W, Y', h(b'g^{b'} \bmod p))$ and the correctness of proxy certificate by VerCert$_{APDW}$(O, proxy certificate). If VerCert$_{APDW}$(O, proxy certificate) returns true, proxy signer P computes the proxy secret key $X = x_P + b$ and return true.

VerCert$_{APDW}$(O, proxy certificate)

A verifier computes $H = h(M_W, Y, h(bg^b \bmod p))$. The verifier validates the correctness of proxy certificate by adopting the certificated public key y_O and the corresponding discrete-logarithm-based verification and H. If (r, s) is correct, then VerCert$_{APDW}$(O, proxy certificate) returns true and that the proxy certificate is validated.

ID$_{APDW}$(TTP, O, y_P, b′, proxy certificate)

The TTP validates proxy certificate by VerCert$_{APDW}$ (O, proxy certificate) to confirm that the proxy public key Y is certificated by the original signer O. TTP computes $g^{b'}$ mod p and $Y' = g^{b'} \times y_P$ mod p. Then TTP checks whether or not $h(bg^b \bmod p) = h(b'g^{b'} \bmod p)$ and $Y = Y'$. If $h(bg^b \bmod p) = h(b'g^{b'} \bmod p)$ and $Y = Y'$ hold, TTP confirms that the proxy secret key X for the proxy public key Y is only known by the one knowing x_P.

Hwang and Chan's scheme consists of the following phases.

System Set-up phase

The public system parameters and functions are the same as above. The public key and secret key of some user U_i with identity ID_i are x_i and $y_i = g^{x_i}$ mod p, respectively. The notation M_W also denotes the proxy warrant.

Proxy authorization phase

The original signer O executes Auth$_{APDW}$(O, P, M_W, b, proxy certificate) to authorize the proxy signer P on behalf of the original signer O in anonymous way. Then the proxy signer P obtains the proxy certificate and the secret value b form the original signer O in secure manners. Then the proxy signer P validates the correctness of the proxy certificate, and the secret value b by adopting the algorithm VerAuth$_{APDW}$(O, P, b, proxy certificate). If VerAuth$_{APDW}$(O, P, b′, proxy certificate) returns true, the proxy signer U_P accepts the proxy authorization and owns a valid proxy certificate $(M_W, Y, h(b \times g^b \bmod p), (r, s))$. Then the proxy secret key and proxy public key are $X = x_P + b \bmod q$ and $Y = g^X \bmod p = y_P \times (g^b) \bmod p$, respectively.

Proxy signature generation and verification phase

First of all, the proxy signer P uses the proxy secret key X to generate the anonymous proxy signature (R, S) on the message m by using any secure discrete-logarithm-based signature generation algorithm. Then the anonymous proxy signature (R, S) along with proxy certificate is sent to the verifier. After receiving m, (R, S) and proxy certificate, the verifier executes VerCert$_{APDW}$ (O, proxy certificate) to check whether or not proxy certificate and the proxy public key Y are really authorized by the original signer O. Using the proxy public key Y and the signature verification, the verifier validates the correctness of the anonymous proxy signature (R, S) on m. Finally, the verifier validates the anonymous proxy signature (R, S) on the message m and the proxy certificate authorized by the original signer O.

Proxy signer identification phase

If the anonymous proxy signature (R, S) along with the proxy certificate (M_W, Y, h(bg^b mod p), (r, s)) has any disputes, the original signer O must run the algorithm ID_{APDW}(TTP, O, y_P, b′, proxy certificate) to convince TTP that the proxy secret key X is only known by someone who has the secret key x_P. Only the singer P knowing the secret key x_P can generate anonymous proxy signature (R, S) by using the proxy secret key X for $X = b + x_P$ mod q. After adopting the proxy public key Y to validate the disputed anonymous proxy signature (R, S), TTP is convinced that the proxy signer P generates the disputed anonymous proxy signature (R, S). Otherwise, TTP does not be convinced.

The anonymous proxy signature scheme satisfies the following properties.

(1) Unforgeability: Proxy signatures can be generated only by authorized proxy signers. All unauthorized users cannot forge proxy signatures even though the original signer is included in them.
(2) Verifiability: Anyone can validate proxy signatures generated by an authorized proxy signer.
(3) Proxy signer's deviation: Each proxy signer cannot obtain the secret key of the original signer or other proxy signers. Moreover, any proxy signer cannot forge signatures of the original signer, or other proxy signers.
(4) Distinguishability: Proxy signatures, original signers' signatures and proxy signers' signatures can be distinguished in polynomial time.
(5) Identifiability: Only the original signer can determine or prove the identities of all proxy signers.
(6) Proxy protection: The original signer cannot obtain the proxy secret key of the proxy signer. When original signers illegally authorize themselves to be the anonymous proxy group, they are able to generate some illegal proxy signatures. But original signers cannot falsely incriminate any proxy signer who ever generated these proxy signatures.
(7) Undeniability: The proxy signer cannot deny the proxy signature generation. Moreover, the original signer cannot deny the generation of the proxy certificates generated by him/her
(8) Strong anonymity: Verifiers cannot directly discovery the identity of the proxy signer. Moreover, each proxy signer cannot know or discover the identity of any other proxy signer.
(9) Original signer's deviation: The original signer cannot obtain the secret key of any proxy signer. The original signer cannot forge the signatures of any proxy signer.

4. Our Realization of Anonymous Proxy Automatic Signature Schemes with Compiler Agents for (Unknown) Virus Detection

Our scheme contains these phases: Compiler-maker authorization phase, server-requester execution phase, customer verification phase, and moderator judgment phase.

The following are some public system-wide parameters and functions. The parameters p and q are two large prime numbers and p = 2q + 1. The parameter g is an element in Z_p^* with order q. A public function h(·) is a public cryptographic hash function. Each one U_i has a secret key $x_i \in Z_q^*$, and a certificated public key $y_i = g^{x_i}$ mod p. The proxy warrant M_W specifies the identity of the compiler maker u_M, the certificated public key y_M of the compiler maker u_M, the delegation period, and the other necessary proxy details.

Compiler-maker authorization phase

The server u_S sends the compiler maker u_M the compiler agent request R_{SM} for the compiling authorization. After validating the request R_{SM}, the compiler maker u_M gives the server an authorized compiler and the proxy certificate. Then the compiler maker authorizes the server as his/her compiler agents.

Step 1.

The server u_S generates the signature (r_{SM}, s_{SM}) on the digest h($u_S||R_{SM}$) by any discrete logarithm based signature scheme [11,12] and sends (R_{SM}, u_S, (r_{SM}, s_{SM})) to the compiler maker, where R_{SM} denotes the compiler agent request. Here and after, any signatures are generated and validated by adopting any discrete logarithm based signature scheme. It is better to select the same discrete logarithm based signature schemes.

Step 2.

After receiving (R_{SM}, u_S, (r_{SM}, s_{SM})), the compiler maker u_M performs the following steps:

(1) Select a secret random number $b \in Z_q^*$.

(2) Compute h.

(3) Use the secret key x_M to generate the signature (r_{MS}, s_{MS}) on h(M_W, h(C_R), Y, h(bg^b mod p)).

(4) Send the server u_S the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})), the compiler C_R, and the secret value b through a secure channel.

Step 3.

After receiving the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})), the compiler C_R', and secret value b′ from compiler maker, the server u_S validates them by executing the following steps.

(1) Compute $Y' = y_S \times g^{b'}$ mod p.

(2) Validate the secure value b′ by checking h(bg^b mod p) = h($b'g^{b'}$ mod p).

(3) Check the correctness of the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})) by validating the signature (r_{MS}, s_{MS}) on the message (M_W, h(C_R), Y, h(bg^b mod p)) with the compiler maker u_M's public key y_M.

If the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})) is correct, then the proxy secret key is $X = x_S + b$ mod q and the proxy public key is $Y = g^X$ mod p.

Server-requester execution phase

Suppose that a requester writes a source program and wants to compile it with an anonymous server's aid.

Step 1.

The requester validates the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})) by verifying the signature (r_{MS}, s_{MS}) on the message (M_W, h(C_R), Y, h(bg^b mod p)). If the proxy certificate is not correct, the requester stops.

Step 2.

The requester generates the signature (r_{RS}, s_{RS}) on h(P, ($u_R||R_{RS}$)) and sends (P, ($u_R||R_{RS}$), (r_{RS}, s_{RS})) to the server, where R_{RS} denotes the compiling request of the requester.

Step 3.

After receiving (P, ($u_R||R_{RS}$), (r_{RS}, s_{RS})), the server validates the signature (r_{RS}, s_{RS}). If the signature is correct, the server accepts requester and prepares to compile the requester's source program; otherwise the server refuses the requester.

Step 4.

The server checks the correctness of the compiler C_R' by verifying the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p)) and h(C_R') = h(C_R).

Step 5.

If the above tests pass, the server compiles the source program P by C_R to generate the executable program E. The compiler C_R not only generates E but also uninterruptedly adopts the proxy secret key X and discrete logarithm based signature scheme to generate the proxy automatic signature (r_{SR}, s_{SR}) on h(u_R, E, h(C_R), h(P), (r_{MS}, s_{MS})). Then the server sends (u_R, E, (r_{SR}, s_{SR})) to the requester.

Step 6.

After receiving (u_R, E, (r_{SR}, s_{SR})), the requester checks the signature (r_{SR}, s_{SR}) on h(u_R, E, h(C_R), h(P), (r_{MS}, s_{MS})). If the signature is correct, the requester begins to publish his/her program to customers.

Step 7.

The requester generates the signature (r_{RC}, s_{RC}) on h((r_{SR}, s_{SR}), E, h(C_R), h(P)).

Step 8.

The requester sends the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})) and ((r_{SR}, s_{SR}), (r_{RC}, s_{RC}), u_R, E, h(P)) to the customer. The signature (r_{RC}, s_{RC}) can guarantee the real source of the executable program E.

Customer verification phase

Before the customer executes the program E, the software may be infected with some viruses. The customer has to check the corresponding signature of the executable program. At the first time, the customer performs the following verifications.

Step 1.

Check the signature (r_{RC}, s_{RC}) on h((r_{SR}, s_{SR}), E, h(C_R), h(P)) by using the requester u_R's public key y_R.

Step 2.

Verify the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})) by using the compiler maker u_M's public key y_M.

Step 3.

Use the proxy public key Y to verify the proxy automatic signature (r_{SR}, s_{SR}) on h(u_R, E, h(C_R), h(P), (r_{MS}, s_{MS})).

If anyone of the above verifications is failure, the customer refuses the execution of E. Otherwise, the executable program E is safe from viruses, and the customer can executes it. Later on, the customer just needs to verify the proxy automatic signature (r_{SR}, s_{SR}) on h(u_R, E, h(C_R), h(P), (r_{MS}, s_{MS})) for detecting virus infection.

Moderator judgment phase

After receiving the software published by a requester, the customer verifies the corresponding signature. If the verification is not correct, the executable program may be infected by viruses. So the customer sends some information to the moderator, and then the moderator can find out the source of the viruses. Moreover, if the source of the viruses is the server, the moderator has to revoke the anonymity of the server.

Step 1.

The customer sends the following data to the moderator.

(1) Proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})),

(2) Anonymous proxy automatic signature (r_{SR}, s_{SR}) on h(u_R, E, h(C_R), h(P), (r_{MS}, s_{MS})), and

(3) u_R, E and h(P).

Step 2.

The moderator checks the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})) by using the signature (r_{MS}, s_{MS}) on the digest h(M_W, h(C_R), Y, h(bg^b mod p)) and the compiler-maker's public key y_M. The moderator also checks the correctness of (r_{SR}, s_{SR}) on the digest h(u_R, E, h(C_R), h(P), (r_{MS}, s_{MS})) by using the proxy public key Y. If both (r_{MS}, s_{MS}) and (r_{SR}, s_{SR}) are correct, the moderator continues this process; otherwise, the moderator rejects the customer's request for the requester or the server is framed by the customer.

Step 3.

The moderator obtains the source program P′ and the signature (r_{RS}, s_{RS}) from the server. The moderator validates the signature (r_{RS}, s_{RS}) by using the requester's public key. If the signature (r_{RS}, s_{RS}) is correct, the received source program P′ is indeed written by the requester; otherwise the moderator runs the identification process ID_{APDW}(TTP, u_M, y_S, b, proxy certificate) to revoke the anonymity of anonymous servers.

Step 4.

The moderator gets the original authorized compiler C_R′ from the compiler-maker.

Step 5.

The moderator applies C_R′ on the source program P′ to generate E′.

Step 6.

The moderator validates whether or not h(P) ≠ h(P′), h(C_R) ≠ h(C_R′), and h(E) ≠ h(E′). If anyone of h(P′) ≠ h(P), h(C_R) ≠ h(C_R′), and h(E) ≠ h(E′) occurs, the moderator runs the identification process ID_{APDW}(TTP, u_M, y_S, b, proxy certificate) to revoke the anonymity of anonymous servers.

Step 7.

If h(P′) = h(P), the moderator checks P line by line to find out whether or not the requester writes a program with viruses.

5. Security Analysis

The security of our scheme is based on the underlying discrete logarithm signature scheme and Hwang and Chan's anonymous proxy signature scheme [10]. By adopting secure discrete logarithm signature schemes, all the secret key of each user and secret random numbers are secure in our scheme. Moreover, the signatures used in our scheme cannot be forged and repudiated. In Hwang and Chan's scheme [10], the anonymity of proxy singers is guaranteed by the security of the secret value b. Fortunately, the secret value b is protected by the discrete logarithm problem and one-way function. In Hwang and Chan's scheme [10], the unforgeability of proxy signatures is based on the fact that the proxy secret key is only known by the proxy singer. Since the proxy secret key is the sum of the secret value b and the proxy signer's secret key, only the authorized proxy singer knows the proxy secret key.

Some security considerations for our scheme are listed below.

(1) Anonymity: Since our scheme adopts Hwang and Chan's anonymous proxy signature scheme, the distributed servers' identities cannot be detected besides the compiler maker. The anonymity is guaranteed by Hwang and Chan's scheme.

(2) Unforgeability: The discrete logarithm signatures and the proxy automatic signatures are both based on the discrete logarithm problem, so all the signatures are unforged.

(3) Verifiability: In our scheme, all signatures can be verified by the corresponding verification equation and signers' public keys. Proxy automatic signatures can also be verified by using the compiler-maker's public key because the underlying Hwang and Chan's anonymous proxy signature scheme satisfies the verifiability property.

(4) Identifiability: The moderator can identify the anonymous servers with the aid of compiler maker. Because the anonymity of severs can be revoked in Hwang and Chan's scheme, the moderator can find out the real server that compiles the source program in the moderator phase.

(5) Undeniability: The proxy automatic signature and discrete logarithm based signature must be generated by the signer's secret key. If the verification of signatures is satisfied, the signer cannot deny he/she generated the verified signatures.

(6) Distinguishability: The proxy automatic signatures, the server's original signatures, the compiler maker's original signatures are generated by different secret keys, respectively. These signatures have to be validated with different public keys, respectively. So these signatures can be distinguished for the verification using different public keys.

(7) Server's protection: A malicious user wants to falsely incriminate the server that he/she compiles a source program with virus and generate the corresponding proxy automatic signature. That is the malicious user has to forge the proxy automatic signatures or has the proxy secret key X of the server. Fortunately the proxy secret key $X = x_S + b \bmod q$ is known only by the server and proxy automatic signatures are unforged. So the proxy automatic signatures can be used to guarantee that the executable program is generated by the server itself.

(8) Server's deviation: In the moderator judgment phase, the moderator can confirm the correct source of the source program and compiler. So the server cannot falsely incriminate any requester writing programs with virus.

(9) Requester's protection: The requester sends $((r_{SR}, s_{SR}), (u_R, E, h(C_R), h(P)))$, the signature (r_{RC}, s_{RC}) on $((r_{SR}, s_{SR}), (E, h(C_R), h(P)))$ and the proxy certificate to the customer. The customer can confirm that the program is distributed from some requester by validating the signature (r_{RC}, s_{RC}). Because the signature (r_{RC}, s_{RC}) is not forgeable, no one can falsely incriminate the requester writes source codes but he/she does not.

(10) Executable program's integrity: The integrity of executable programs is protected by proxy automatic signatures. Anyone can verify the proxy automatic signature by using the proxy public key Y to detect modification of executable programs.

(11) Source program's secrecy: Because h(P) is used to generated signatures (r_{SR}, s_{SR}) and (r_{RC}, s_{RC}), signatures (r_{SR}, s_{SR}) and (r_{RC}, s_{RC}) can be verified by only using h(P) without releasing the content of the source program P. Except the server and the requester, no one knows the source program P of the executable program E.

(12) Requester's deviation: The integrity of executable programs is guaranteed by the proxy automatic signatures. Since proxy automatic signatures are unforged, the requester cannot incriminate any server to generate any executable program containing viruses.

6. Performance Analysis and Discussion

The performance analysis of our scheme is stated below. First, some notations are defined below.

T_{mp}: The computation cost for one modular multiplication modular p.

T_{mq}: The computation cost for one modular multiplication modular q.

T_{exp}: The computation cost for one modular exponentiation modular p.

T_{hash}: The computation cost for one-way hash functions.

T_{inv}: The computation cost for inverse operation modular q.

T_{sig}: The computation cost for the discrete logarithm based signature generation, which includes one T_{hash} to generate the message digest.

T_{ver}: The computation cost for the discrete logarithm based signature verification, which includes one T_{hash} to generate the message digest.

Because the underlying signature scheme is not specified in our scheme, the signature generation cost T_{sig} including one T_{hash} and the signature verification cost

T_{ver} including one T_{hash} denote the signature and verification cost of the underlying signature scheme, respectively. Now the computational cost of our scheme is given phase by phase.

Figure 2 shows the computational cost in the compiler-maker authorization phase. First, one T_{sig} is used to generate the signature (r_{SM}, s_{SM}) on the proxy request R_{SM} from the server to the compiler maker and one T_{ver} is used to verify (r_{SM}, s_{SM}). After the verification of signature (r_{SM}, s_{SM}), the compiler maker computes (M_W, h(C_R), Y, h(bg^b mod p). This computation costs $T_{exp} + 2T_{mp} + 1T_{hash}$. Then the compiler maker needs one T_{sig} to generate the signature (r_{MS}, s_{MS}) on the digest of (M_W, h(C_R), Y, h(bg^b mod p). After receiving the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})), the secret value b, and the compiler maker C_R, the verification of the signature (r_{MS}, s_{MS}) and the validation of (b, C_R) cost the server one T_{ver} and $T_{exp} + 2T_{mp} + 2T_{hash}$, respectively.

Figure 3 shows the computational cost in the server-requester execution phase. The first one T_{ver} is used to check the correctness of the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p), (r_{MS}, s_{MS})). Then the generation of the signature (r_{RS}, s_{RS}) on the digest of (P, ($u_S||R_{RS}$)) costs the requester one T_{sig}. After receiving (P, ($u_S||R_{RS}$), (r_{RS}, s_{RS})), the verification of the signature (r_{RS}, s_{RS}) costs one T_{ver}. If the verification of signature (r_{RS}, s_{RS}) is correct, the server needs $T_{ver} + T_{hash}$ to check the compiler C_R by verifying the proxy certificate (M_W, h(C_R), Y, h(bg^b mod p)) and h(C_R) = h(C_R). Then one T_{sig} is used to generate the proxy automatic signature. After receiving the proxy automatic signature (r_{SR}, s_{SR}), the requester needs one T_{ver} to verify (r_{SR}, s_{SR}). If (r_{SR}, s_{SR}) is correct, the re-

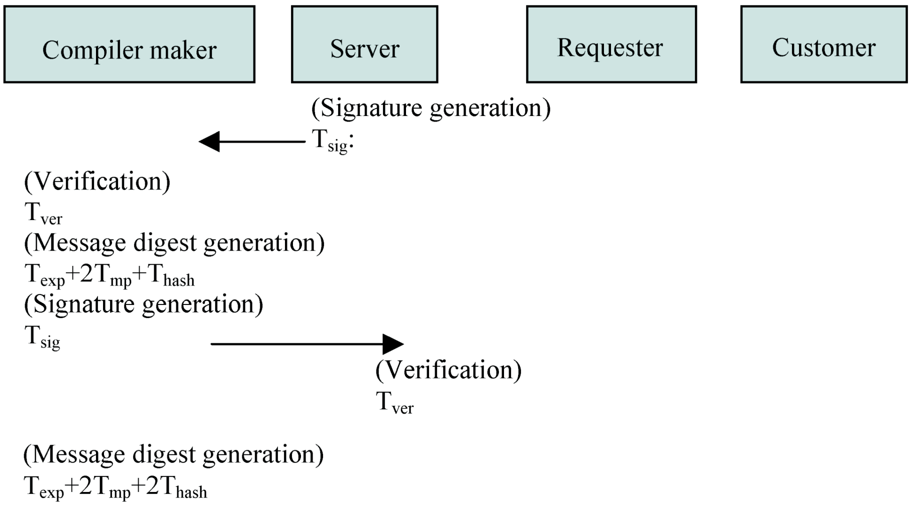

Figure 2. Computational cost in compiler-maker authorization phase.

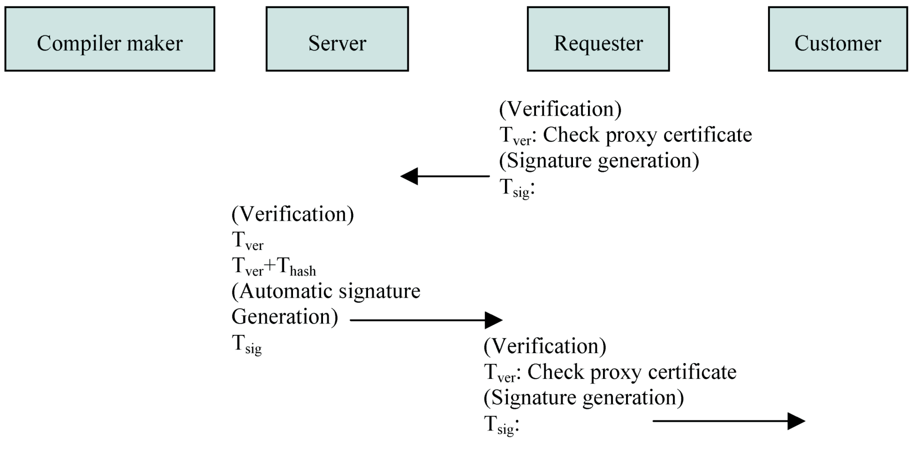

Figure 3. Computational cost in server-requester execution phase.

quester needs one T_{sig} to generate the signature (r_{RC}, s_{RC}).

Figure 4 shows the computational cost in the customer verification phase. The first time verification, the customer totally needs $3T_{ver}$ to verify signature (r_{RC}, s_{RC}), the proxy certificate (M_W, $h(C_R)$, Y, $h(bg^b \bmod p)$, (r_{MS}, s_{MS})) and the automatic signature (r_{SR}, s_{SR}). Later, the customer only needs one T_{ver} to verify the proxy automatic signature (r_{SR}, s_{SR}) before executing the received program.

In the moderator phase, the verification of three signatures (r_{MS}, s_{MS}), (r_{SR}, s_{SR}) and (r_{RS}, s_{RS}) costs $3T_{ver}$. Then the moderator coasts $3T_{hash}$ and one T_{sig} to generate the proxy automatic signature on the C_R, P, and E. If the moderator finds out the source of infection is server, he/she needs $T_{exp} + 2T_{mp} + T_{hash}$ to revoke the anonymity of the server. Therefore, the upper bound for the computational cost in the moderator phase is $2T_{exp} + 4T_{mp} + 5T_{hash} + 3T_{ver} + T_{sig}$.

Table 1 summarizes the computational cost of each kind of participates in our scheme.

7. Comparison

There are many related signature schemes for virus detection had been proposed previous. Table 2 gives the comparison among our scheme, Hwang and Chen's scheme [9], Hwang and Li's scheme [8], and Lin and Jan's scheme [6].

By adopting Hwang and Chan's anonymous proxy signature scheme, our new proxy automatic signature scheme can provide servers anonymity while the other three schemes do not. By the anonymous property, verifiers can store only compiler makers' public keys to verify proxy certificates. Then verifiers obtained certificated proxy public key to verify automatic signatures. Because the other three schemes do not provide server anonymity, public keys of servers should be also stored and certificated. Moreover, because using the signature

Table 1. Computational cost of participates in our scheme

Participates	Computational cost
Compiler-maker	$T_{exp}+2T_{mp}+2T_{hash}+T_{ver}+T_{sig}$
Server	$T_{exp}+2T_{mp}+3T_{hash}+3T_{ver}+2T_{sig}$
Requester	$2T_{ver}+2T_{sig}$
Customer	[1] $3T_{ver}$ (First time) [2] T_{ver} (Afterward)
Moderator	$2T_{exp}+4T_{mp}+5T_{hash}+3T_{ver}+T_{sig}$ (Upper bound)

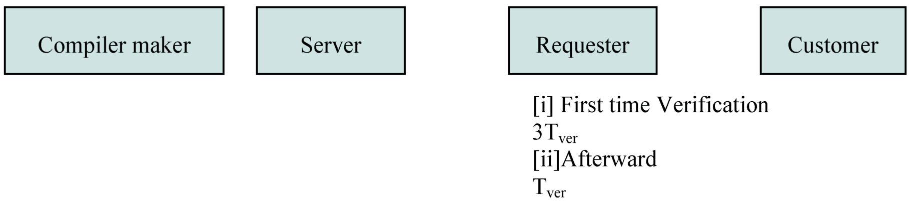

Figure 4. Computational cost in customer verification phase.

Table 2. Comparison among four automatic signature schemes

	Our Scheme	Hwang and Chen's Scheme	Hwang and Li's Scheme	Lin and Jan's Scheme
Anonymity	√	×	×	×
Source confirmation of executable programs in advance	√	×	×	×
Storage only for compiler maker's public key and proxy certificate	√	×	×	×
Pre-detection of servers' deviation	√	×	×	×
Suitability for any discrete logarithm based signature schemes	√	√	×	×
Judgment capability	√	√	√	×
No forgery attack	√	√	√	×
No length restriction	√	√	√	×
Client-server models	√	√	√	√

(r_{RC}, s_{RC}), the server's deviation can be detected in advance and the customer can confirm the source of the receiving program. In the other three schemes, the server's deviation and program sources cannot be detected and confirmed in advance. Only our new scheme and Hwang and Chen's scheme do not specify underlying discrete logarithm signature schemes while the other two schemes do. Both our new scheme and Hwang and Chen's scheme have the advantage that any discrete logarithm based signature scheme can be used. This advantage provides flexibility for the practical implementation. Furthermore, only the first three schemes have the moderator phase to exactly detect the source of infection except Lin and Jan's scheme. So only the first three schemes have judgment capability. Among the four schemes, only Lin and Jan's scheme has length restriction on source codes and is vulnerable under forgery attacks [7]. According to Table 2, our scheme provides more functions and services than the other three schemes.

8. Conclusion

A new anonymous proxy automatic signature scheme with compiler agents is proposed to detect infection of (unknown) virus. In the new scheme, customers can always verify the proxy automatic signature by using the proxy public key which is first validated by only the compiler maker's public key. Even if the authorization of some server is terminated, customers still easily verifies proxy automatic signatures because only the compiler maker's public key is necessary. The source of the executable program can be validated by customers when they received. Moreover, the new scheme provides robust infection detection of viruses. If some viruses exist, the infection source can be exactly found out. Our scheme has the properties about protection and deviation for servers and requester mentioned in Section 2. Finally, our scheme satisfies all the other security goals stated in Section 2 for a virus detection system.

References

[1] Okamoto, E., "Integrated Security System and its Application to Anti-viral Methods," *Proc. 6th Virus and Security Conf* (1993).

[2] Hedberg, S., "Combating Computer Viruses: IBM's New Computer Immune System," *Parallel & Distributed Technology: Systems & Applications, IEEE,* Vol. 4, pp. 9–11 (1996).

[3] Nachenberg, C., "Computer Virus-antivirus Coevolution," *Communications of the ACM,* Vol. 40, pp. 46–51 (1997).

[4] Subramanya, S. R. and Lakshminarasimhan, N., "Computer Viruses," *Potentials, IEEE,* Vol. 20, pp. 16–19 (2001).

[5] Usuda, K., Mambo, M., Uyematsu, T. and Okamoto, E., "Proposal of an Automatic Signature Scheme Using a Compiler," *IEICE Transactions Fundamentals,* Vol. E79-A, pp .94–101 (1996).

[6] Lin, W.-D. and Jan, J.-K., "An Automatic Signature Scheme Using a Compiler in Distributed Systems," *IEICE Transactions on Communications,* Vol. E83-B pp. 935–941 (2000).

[7] Tseng, Y.-M., "Cryptanalysis and Restriction of an Automatic Signature Scheme in Distributed Systems," *IEICE Transactions on Communications,* Vol. E86-B pp. 1679–1681 (2000).

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

[9] Hawng, S.-J. and Chen, K.-H., "A Proxy Automatic Signature Scheme Using a Compiler in Distributed Systems for (Unknown) Virus Detection," *Advanced Information Networking and Applications 2005,* Taiwan, R.O.C., pp. 649–654 (2005).

[10] Chan, C.-C., "Anonymous (Multi-) Proxy Signature Schemes with Undeniable Agents," Master Thesis, Tamkung University, Taiwan, R.O.C. (2005).

[11] ElGamal, T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm," *IEEE Transactions on Information Theory,* Vol. 31, pp. 469–1985 (1985).

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

Manuscript Received: Jan. 6, 2006
Accepted: May. 17, 2006