SSL을 위한 Signcryption 기반 Ciphersuite

서병국[0,*] 박언탁[*] 정재학[*] 심경아[**]
[*]소프트포럼㈜, [**]한국정보보호진흥원
bksuh@softforum.co.kr

Signcryption-based ciphersuite for SSL

Byungkuk Seo[0,*] Untak Park[*] Jaehak Jung[*] Kyungah Shim[**]
[*]Softforum Co., LTD, [**]KISA

요 약

Signcryption 알고리즘은 전자서명과 암호화를 한번의 연산으로 수행할 수 있는 알고리즘으로 기존의 방식인 전자서명과 암호화를 개별적으로 수행하는 방식에 비해 효율성이 우수하다고 아려져 있다. 본 논문에서는 Signcryption 알고리즘을 SSL에 적용하는 방법을 기술하고 적용된 방법이 효율성 측면에서 우수함을 보인다.

1. 서 론[1]

Secure Socket Layer (SSL, [1]) 프로토콜은 Web 보안을 위한 업계 표준 보안 프로토콜이며 현재 대부분의 Web 서버와 브라우저가 지원하고 있다. 본 논문에서는 SSL 프로토콜의 효율성을 향상 방안으로 효율성이 뛰어나다고 알려진 Signcryption ([2]) 알고리즘을 SSL에 적용하는 방안 및 효율성 향상 효과를 기술한다. 본 논문에서 제안하는 signcryption 기반 알고리즘을 이용할 경우 기존의 방식에 비해 6배 정도의 효율성 향상 효과를 얻을 수 있다.

본 논문의 순서는 다음과 같다. 2장에서는 SSL 프로토콜을 소개하고, 3장에서는 signcryption 방식을 소개한다. 4장에서는 signcryption 이용 ciphersuite의 정의 및 알고리즘 적용방안을 기술하고, 5장에서는 효율성 향상 결과를 보인다. 그리고 6장에서 끝을 맺는다.

2. SSL 프로토콜

SSL 프로토콜은 크게 HTTP 데이터의 암호화를 담당하는 레코드 프로토콜과 레코드 프로토콜에서 사용할 암호 알고리즘 및 키의 설정을 담당하는 핸드쉐이크 프로토콜로 구분할 수 있다. 본 논문에서 고려하는 signcryption 알고리즘은 핸드쉐이크 프로토콜에서 키의 교환 및 객체 인증을 위해 이용된다. 우리가 이용할 핸드쉐이크 프로토콜의 구조를 기술하면 그림 1과 같다. 사용되는 알고리즘 종류에 따라 메시지들은 선택적으로 보내어 진다.

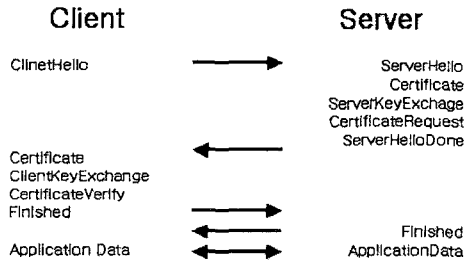

그림 1. SSL 핸드쉐이크 프로토콜

ClientHello, ServerHello 메시지는 클라이언트와 서버가 사용할 알고리즘을 선택하기 위해 이용되며, 서버가 보내는 Certificate 메시지부터 클라이언트가 보내는 CertificateVerify 메시지까지는 객체 인증 및 키 교환을 위해 이용된다. 그리고 클라이언트와 서버가 보내는 Finished 메시지는 핸드쉐이크 프로토콜이 올바르게 수행되었는지 여부를 검사하기 위해 사용된다. Finished 메시지의 교환이 끝난 이후 Application Data인 HTTP 데이터는 레코드 프로토콜을 이용하여 보호된다.

SSL 프로토콜은 새로운 알고리즘을 추가하는 것이 용이하도록 구성되어 있다. SSL 프로토콜에서는 클라이언트가 사용할 수 있는 알고리즘의 묶음인 ciphersuite들을 모두 보내면 그 중의 하나를 서버가 선택하는 방법으로 사용할 ciphersuite를 확정한다. 따라서 signcryption 기반 알고리즘을 추가하기 위해서는 SSL에서 알고리즘을 표현하기 위해 사용하는 ciphersuite의 종류에 signcryption 기반 ciphersuite를 추가하면 된다. 그 이후 클라이언트와 서버는 ClientHello 메시지와 ServerHello 메시지의 교환을 통하여 추가된 알고리즘을 이용하는 것이 가능하다.

본 연구는 KISA (No.2001-S-092)의 지원을 받았음

3. Signcryption 알고리즘

Signcryption 방식은 전자서명과 공개키 암호화를 한 번에 수행할 수 있는 방식으로서 독립적으로 암호화와 전자서명을 하는 방식에 비해 효율성이 우수하다고 알려져 있다. 본 논문에서는 Zheng이 제안한 DSS 전자서명을 이용한 signcryption방법을 개략적으로 소개한다. 보다 자세한 내용은 [2]를 참조 바란다.

3.1 사용기호

- p, q : DSA의 소수 조건을 만족하는 두 소수
- g : 위수가 q인 Z_p^*상의 생성원
- x_C, $y_C(=g^{x_C} \bmod p)$: Client의 개인키와 공개키
- x_S, $y_S(=g^{x_S} \bmod p)$: Server의 개인키와 공개키
- m : 메시지
- (r, s) : 서명
- E, D : 암호화 함수와 복호화 함수
- MAC : MAC 생성을 위한 함수

3.2 메시지 m에 대한 signcryption 생성

클라이언트가 서버에게 signcryption을 이용한 전자서명 및 암호화의 수행 순서는 다음과 같다.

1. 난수 k를 생성한다.
2. 서버의 공개키 y_S를 이용하여 $(y_S)^k \bmod p$ 값을 계산한 후, 이 값을 k_1, k_2 두 개로 나눈다. (k_1은 메시지 암호화에 이용되는 키 이고, k_2는 MAC의 생성에 이용되는 키이다. 나누는 방법은 같은 길이로 나누거나 또는 사용되는 알고리즘에 적절한 방법을 활용한다.)
3. 메시지 m의 암호문 $c=E_{k_1}(m)$ 를 계산한다.
4. 메시지 m의 MAC 값인 $r=MAC_{k_2}(m)$을 계산한다.
5. $s=(k/(r+x_C)) \bmod q$ 값을 계산한다.
6. (c, r, s) 값을 전송한다.

3.3 메시지 m에 대한 signcryption 검증

서버가 클라이언트로부터 받은 signcryption 값을 이용하여 서명 검증 및 복호화의 순서는 다음과 같다.

1. client의 공개키 y_C를 얻는다.
2. 받은 서명 (r, s), 그리고 y_C를 이용하여 다음의 식을 계산한다.

$$v=(y_C \times g^r)^{s \cdot x_S} \bmod p$$

3. v 값을 이용하여 k_1, k_2 값을 얻는다.
4. k_1을 이용하여 c의 복호화된 값인 $m'=D_{k_1}(c)$ 값을 계산한다.
5. k_2를 이용하여 $r'=MAC_{k_2}(m')$ 값을 계산한다.
6. r'값과 client로부터 받은 r 값이 일치할 경우, Signcryption이 검증된다.

4. Signcryption 의 SSL 적용

Signcryption을 SSL에 적용하기 위해서는 다음의 2가지가 필요하다. 하나는 signcryption을 표현할 수 있는 ciphersuite를 정의하는 것이며 다른 하나는 signcryption 알고리즘을 SSL 핸드쉐이크 프로토콜에 정의된 메시지 형태로 재구성하는 것이다

우리는 다음의 ciphersuite를 signcryption을 위하여 새로이 정의하였고, ClientHello, ServerHello 메시지의 교환을 통하여 사용될 수 있도록 하였다.

SSL_SC-DSS_WITH_AES_CBC_SHA

ServerHello 메시지에서 위의 ciphersuite가 선택되었을 경우 SSL 프로토콜은 제안한 signcryption 알고리즘을 적용하여 키 교환 및 객체 인증을 수행한다. 이 경우 클라이언트와 서버는 모두 DSS 인증서를 가지고 있어야 하며 signcryption 알고리즘의 특성으로부터 공개키에 포함되는 p, q, g 값은 서로 동일해야 한다.

우리는 3장에서 기술된 signcryption 알고리즘을 SSL 프로토콜에 맞도록 부분적으로 변형하여 사용한다. 원래의 Signcryption 방식에서는 전송하고자 하는 메시지를 별도의 대칭키 암호 알고리즘을 이용하여 암호화 하였다. 그러나 우리가 Signcryption을 SSL에 적용할 경우, 키의 교환만을 목적으로 하므로 별도의 암호화는 필요하지 않으며 SSL은 signcryption에서 이용되는 데이터 암호화용 키를 직접적으로 이용할 수 있다.

Signcryption을 이용할 경우 핸드쉐이크 프로토콜 메시지 중에서 ServerKeyExchange, CertificateVerify 메시지는 그 의미를 갖지 않으며 ClientKeyExchange 메시지를 이용하여 키 공유 및 클라이언트 인증이 수행된다. Signcryption을 적용하였을 경우 ClientKeyExchagne 메시지의 생성 및 서버의 처리과정은 다음과 같다.

4.1 ClientKeyExchange 메시지 생성

클라이언트가 서버에게 signcryption을 이용한 전자서명 및 암호화의 수행 순서는 다음과 같다.

1. 난수 k를 생성한다.
2. 서버의 공개키 y_S를 이용하여 $key=(y_S)^k \bmod p$ 값을 계산한다.
3. handshake의 MAC 값인 $r=MAC_{key}(handshake)$을 계산한다.
4. $s=(k/(r+x_C)) \bmod q$ 값을 계산한다.
5. (r, s) 값이 ClientKeyExchange 메시지이다.

Client는 key를 premaster secret으로 활용한다. master secret을 얻는 과정은 Diffie-Hellman 알고리즘과 동일한 방법을 활용한다.

4.2 ClientKeyExchagne 메시지 검증

서버가 클라이언트로부터 받은 signcryption 값을 이용하여 서명 검증 및 복호화의 순서는 다음과 같다.

1. client의 공개키 y_C를 얻는다. (Certificate 메시지)
2. ClientKeyExchagne 메시지인 (r, s), 그리고 y_C를 이용하여 다음의 식을 계산한다.

$$key' = (y_C \times g^r)^{s \cdot x_S} \bmod p$$

3. key'을 이용하여 $r' = MAC_{key'}(handshake)$ 값을 계산한다.
4. r'값과 client로부터 받은 r 값이 일치할 경우, Signcryption이 검증된다.

Server는 key'를 premaster secret으로 활용한다. master secret을 얻는 과정은 Diffie-Hellman 알고리즘과 동일한 방법을 활용한다.

5. 효율성

본 논문에서 제안한 signcryption 기반 알고리즘과 안전성 측면에서 가장 유사한 것은 DHE_DSA 기반 키 공유 알고리즘이며 ciphersutie 이름은 다음과 같다.

SSL_DHE_DSS_WITH_AES_CBC_SHA

두 알고리즘은 perfect forward secrecy 측면에서 두 알고리즘의 안전성이 다르므로 DHE_DSA 기반 키 공유 알고리즘을 실행할 때, 서버가 하나의 DH 임시키를 생성하여 여러 세션에서 공통적으로 이용하도록 하는 방법을 이용해야 한다.

우리는 기존의 SSL 기반라이브러리인 OpenSSL ([3])을 확장하여 signcryption을 적용하였으며 실험 데이터 역시 OpenSSL에서 제공한 암호 모듈을 이용한 것이다. 그리고 실제적인 안전성을 고려하여 두 알고리즘 모두 1024-bit 키 교환을 이용하도록 하였다.

실험 환경은 다음과 같다.

- 클라이언트 및 서버: Enterprise 3500, CPU: UltraSparc 400Mhz X 6, Memory: 4GB

클라이언트 및 서버 어플리케이션은 OpenSSL에서 제공하는 s_server, s_client를 이용하였다. 클라이언트와 서버간의 통신은 동일한 플랫폼에서 루프백 회선을 이용 네트웍 영향을 최소화 시켰다. 그리고 SSL 핸드쉐이킹을 50번 반복 수행 후 각 알고리즘별로 평균시간을 구한 결과는 다음과 같다.

항목	Signcryption	DHE
시간	200 msec	1360 msec
키교환	Signcryption	DHE 1024 bits
서명	DSA	DSA
암호화	AES	AES
해쉬	SHA-1	SHA-1
인증	상호인증	상호인증

위의 실험 결과에서 볼 수 있듯이 SSL 프로토콜에서 Signcryption을 이용할 경우 기존의 방식을 이용하는 것에 비해 6배 이상의 효율성 향상 효과를 얻음을 알 수 있다.

6. 결 론

본 논문에서는 SSL의 효율성 향상을 위하여 효율성 측면에서 우수하다고 알려진 Signcryption 기반 알고리즘을 SSL에 적용하는 방식을 제안하였다. 본 논문에서 제안하는 방법을 적용하였을 경우 SSL에서 자체적으로 제공하는 방법을 적용하는 것에 비해 6배 정도의 효율성 향상 효과를 얻는다.

향후 과제로는 제안된 SSL용 signcryption 방식의 안전성 검증이 요구된다.

7. 참고문헌

[1] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL protocol version 3.0," November 18, 1996

[2] Y. Zheng, "Digital Signcryption or How to Achieve Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption)," Advances in Cryptology -- Crypto'97, Springer-Verlag, 1997

[3] http://www.openssl.org