154. [Security] SSL과 인증서 구조 이해하기 : CA (Certificate Authority)를 중심으로

이번 포스트에서는 인증서의 구조와 동작 원리에 대해 알아보고, 이것이 실제 SSL 기반의 보안 연결에서 어떻게 사용되는지에 대해 알아본다.

사실은, 쿠버네티스에서 User (SA가 아님) 사용하다가 인증서 관리를 다시 하게 되어서..... 다 아는 내용이지만 정리해보았다.

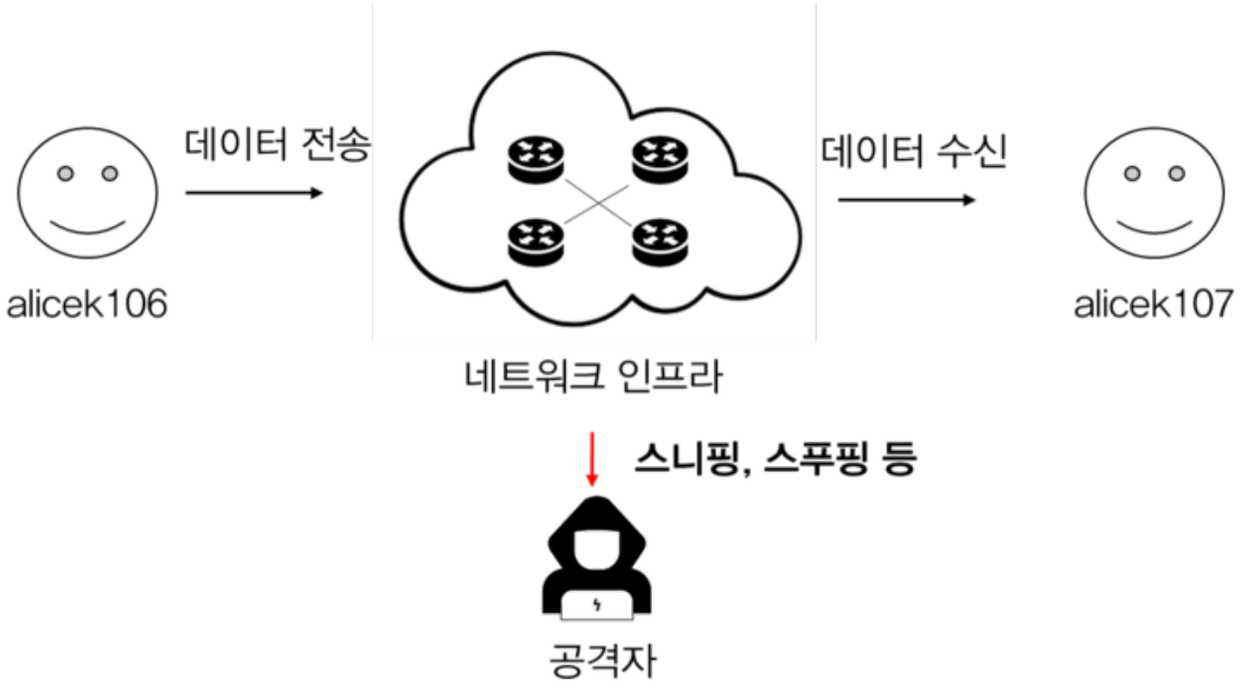
0. 들어가기 전

본 포스트는 독자가 기초적인 네트워크 및 암호화 기술 지식을 알고 있다는 가정 하에 작성되었다. 그 중 필수적으로 알아야 하는 기술은 **'비대칭 키'** 방식으로, 흔히 말하는 PKI (Public Key Infrastructure : 공개 키 기반의 암호화 시스템) 의 작동 원리를 반드시 이해해야만 한다. 비대칭 키의 동작 원리는 크게 어렵지 않기 때문에 본 포스트에서 별도로 다루지 않지만, 위키 백과사전 등에서 유용한 설명을 찾을 수 있다.

1. 왜 인증서를 사용하는가

오늘 날 많은 사람들이 네트워크를 통해 패킷을 주고 받는다. 그러나 전통적으로

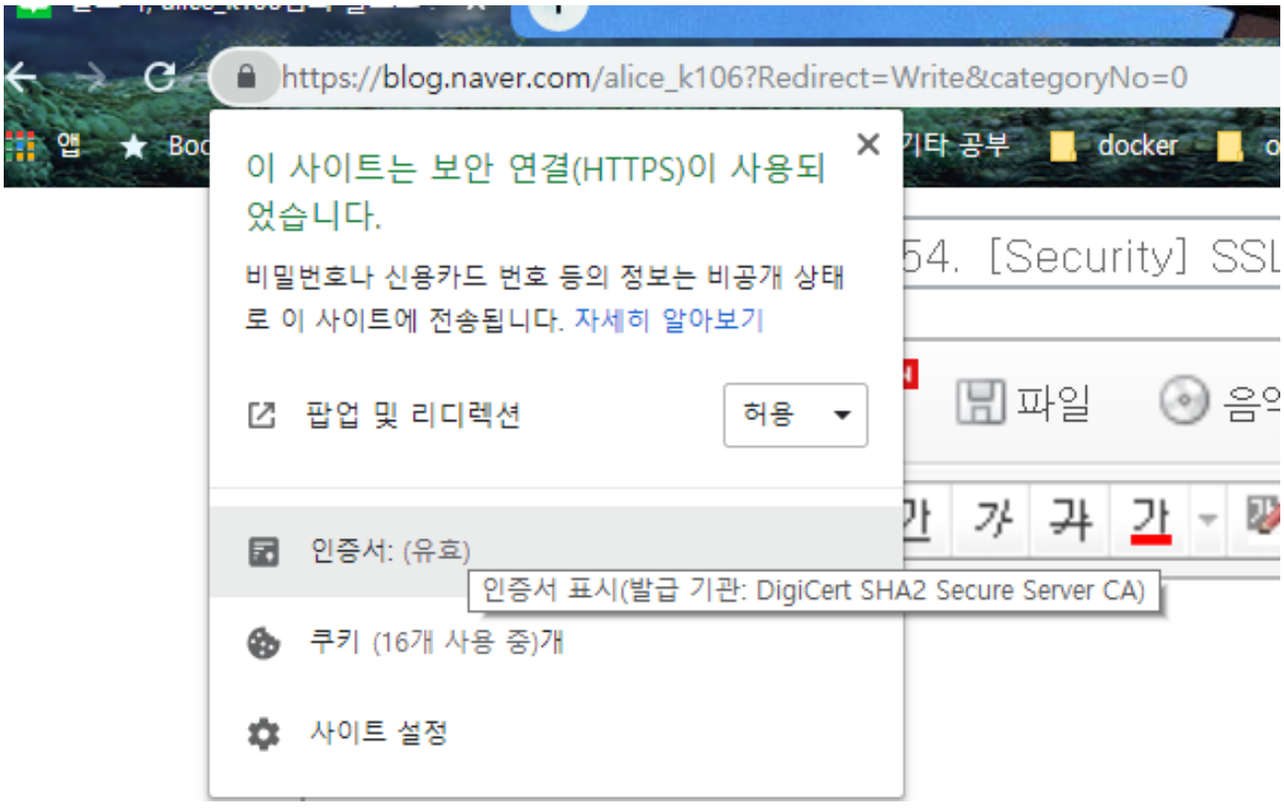
네트워크는 여러 공격자로부터의 위협이 도사리는 공간으로 간주되어 왔다. 우리가 특정 서버에 요청을 주고 받을 때 수 많은 라우터와 스위치를 거치게 되는데, 그 중간에서 누군가가 우리의 패킷을 훔쳐 (Sniffing) 볼 수 있기 때문이다. 패킷을 훔쳐 본다는 것은 결국 우리가 입력한 데이터, 예컨대 비밀번호와 같이 민감한 정보를 제 3자가 열람할 수 있다는 뜻이기 때문에 이를 미연에 방지하기 위한 많은 방법들이 연구되어 왔다.



만약 네트워크 데이터가 암호화된다면, 중간에 공격자가 패킷을 열람하더라도 데이터가 유출되는 것을 막을 수 있다. 오늘날 가장 널리 쓰이고 있는 암호화 방식이 SSL/TLS1 라는 것인데, 이 방식은 '인증서' 라고 하는 일종의 서명을 사용한다. 자세한 동작 원리는 뒤에서 다시 설명하겠지만, 인증서라는 것은 결국 **'당신이 신뢰할 수 있는 사람이냐'** 라는 것을 확인하기 위한 용도이며 이것이 패킷 데이터를 암호화하기 위한 첫 단계라고 생각하면 된다.

인증서가 신뢰할 수 있다는 검증 작업을 거치고 나서야 비로소 데이터 암호화 작

업이 수행된다. 그 과정이 SSL 보안 통신이라고 불리는 것이며, 이론상 이를 해독하는 것이 거의 불가능하기 때문에 오늘날 다양한 보안 연결에서 사용되고 있다. 우리가 흔히 사용하고 있는 (그렇지만 잘 인지하지 못하고 있는) 보안 연결은 다름아닌 웹 브라우저 <-> 웹 사이트 간 연결이다.



웹브라우저의 주소 옆에 자물쇠는 인증서 (CA) 를 통해 브라우저-웹 서버 간 보안 연결이 수립되었다는 것을 의미한다. (자세한 원리에 대해서는 뒤에서 다시 설명한다). 그러나 인증서를 사용하지 않은 웹 서버의 경우 위 그림의 오른쪽처럼 '주의 요함' 이라는 문구가 출력되는데, 이는 데이터가 암호화되지 않은 채로 전송되고 있으며 네트워크에 존재할 수 있는 공격자가 스니핑, 스푸핑 등을 통해 패킷을 훔쳐볼 경우 데이터가 전부 유출될 수 있음을 뜻한다.

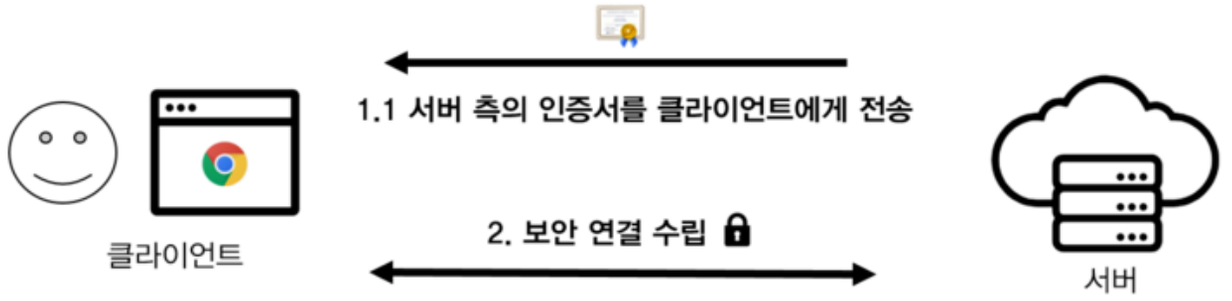
그렇다면 일반적인 사용자가 인증서 구조, 즉 본 포스트가 다루려는 SSL과 CA 등을 알 필요가 있을까? 답은 **'아니오'** 다. OOP에서 흔히 말하는 Private 명시자처럼, 아무것도 모른 채로 웹 브라우저에 내장된 암호화 기능의 혜택을 받으면 된다. 하지만 당신이 서버 관리자, 클라우드 관리자라면 이야기는 조금 달라진다. 여러 컴포넌트가 네트워크 상에서 상호 통신할 때, Payload 데이터의 유출을 막기 위해서는 보안 연결을 위한 SSL 및 인증서의 도입을 반드시 고려해야만 한다 (특히 오픈스택이나 쿠버네티스처럼 컴포넌트가 더욱 세분화되고 모듈화되어 있다면 더욱 그렇다). 사실 예전에 이와 비슷한 맥락의 포스트를 작성한 적이 있는데, 바로 도커 데몬에 보안 인증서를 적용하는 것이다. 실제 운영 환경에 있다면 이러한 작업은 필수적이다.

https://blog.naver.com/alice_k106/220743690397

물론 돈을 지불하고 보안 대행 업체에게 보안 연결 및 인증서 관리를 위임할수도 있지만, 인증서 및 보안 연결의 동작 원리는 인프라 및 네트워크 관리 측면에서 매우 유용한 지식이다. 따라서 당신이 영상처리나 머신러닝과 같이 소스코드 레벨의 알고리즘에 올인하는 개발자가 아니라면 인증서의 구조는 반드시 이해하고 넘어가는 것이 좋다.

2. 보안 연결의 순서

빠른 이해를 위해서 간단하게만 설명하자면, 서버와 클라이언트 간 보안 연결은 크게 두 가지 단계로 나누어진다. 첫 번째 단계는 클라이언트의 CA를 통해 서버 인증서가 신뢰 가능한지를 확인하는 단계이다. 간단하게 말해서, **'당신 (서버) 가 신뢰할 수 있는 사람이냐'** 라는 것을 **'인증서'** 를 통해 검증한다.



1.2 인증서가 신뢰 가능한지를 검증

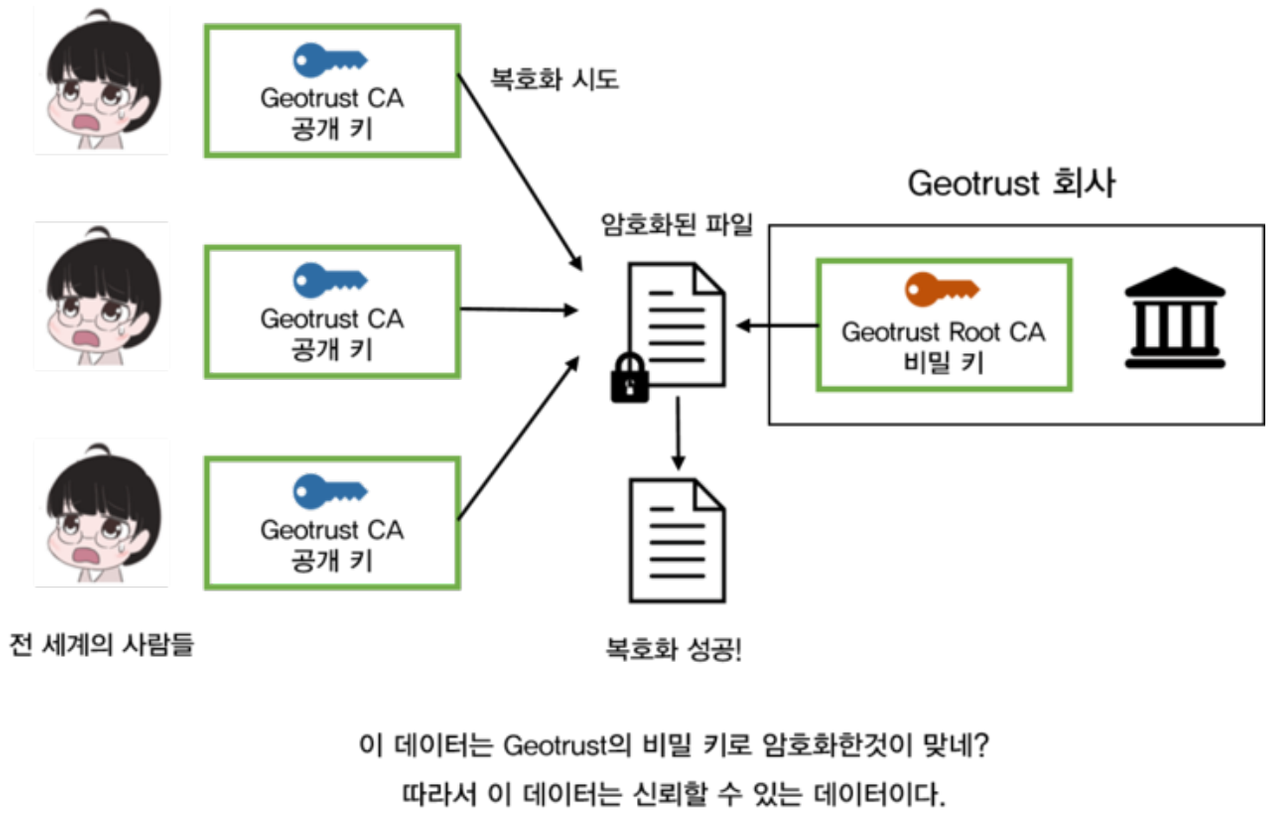
인증서 검증을 통해 서버가 신뢰할 수 있다고 판단되면, 클라이언트는 두 번째 단계인 보안 연결 수립 단계를 진행한다. 서버-클라이언트 간에 생성된 랜덤 값을 통해 대칭 키를 생성하고, 이를 통해 네트워크 데이터를 암호화해 전송한다.

결국은 **인증서를 통한 신뢰 검증 (첫 번째 단계)** 이 완료되어야만 **보안 연결 수립 (두 번째 단계)** 을 할 수 있으므로, 인증서가 어떠한 원리로 동작하는지를 이해하는 것이 중요하다. 본 포스트의 제목이 나타내듯이, 여기서는 인증서의 동작 원리를 중점적으로 설명한다.

3. 인증서의 구조

3.1 (널리 알려진) 신뢰 가능한 기관

이 세상에는 무조건 신뢰할 수 있는 기관이 몇 군데 존재한다. 그 기관들은 보통 최상위 인증 기관이라고 불리며, Root CA라는 인증서를 발급하는 기관이다 (지금 당장은 Root CA를 몰라도 된다). 컴퓨터공학을 전공한다면 한 번쯤은 들어 봤을 만한 Verisign, Geotrust 등의 회사가 바로 그러한 기관이다. 본 포스트에서는 Geotrust를 예시로 들도록 하겠다.



이 기관들은 본인들만의 고유한 비밀 키를 가지고 있고, 이에 대응하는 공개 키를

전 세계에 배포한다. 그리고 세상 사람들은 암묵적으로 이 기관들은 신용할 수 있음을 서로 약속하고, 배포된 Geotrust의 공개 키로 복호화가 가능한 데이터는 Geotrust의 비밀키로 암호화되었기 때문에 신용할 수 있는 데이터라고 간주한다. 물론 Geotrust의 비밀 키는 철저한 보안 속에서 절대로 유출되지 않아야만 한다는 전제 조건이 필요하긴 하지만, 이러한 기관은 보안을 전문으로 하는 회사이기 때문에 보통은 안전하다고 생각하면 된다.

일반적으로 이렇게 신뢰 가능하다고 여겨지는 기관의 공개 키는 여러분의 컴퓨터에 이미 설치되어 있는 경우가 많다. Mac OS 라면 키체인에 있을 것이고, **여러분의 웹브라우저인 파이어폭스, 크롬 등에도 Geotrust의 공개 키는 이미 내장되어 있다.**

3.2 일반적인 인증서의 작동 방식

이 세상에는 '인증서' 라고 부르는, 내가 신뢰할 수 있는 사람인지를 증명하기 위한 파일이 존재한다. 그렇다면 인증서는 어떻게 내가 신뢰할 수 있음을 증명하는 것일까?

3.2.1 인증서의 내용물

상대방이 신뢰할 수 있는지 검증하기 위해 존재하는 '인증서' 라는 포맷의 파일에 대해서 먼저 알아보자. 인증서 파일 안에는 다양한 내용이 저장되어 있으며, 대표적인 내용물은 아래와 같다.

(1) 인증서의 소유자 이름,

(2) 인증서 소유자의 공개 키 (당연히 비밀 키는 소유자가 가지고 있다),

(3) 인증서의 유효 기간,

(4) 고유한 UID

(5) 인증서의 기타 모든 값들을 해시화한 값 2

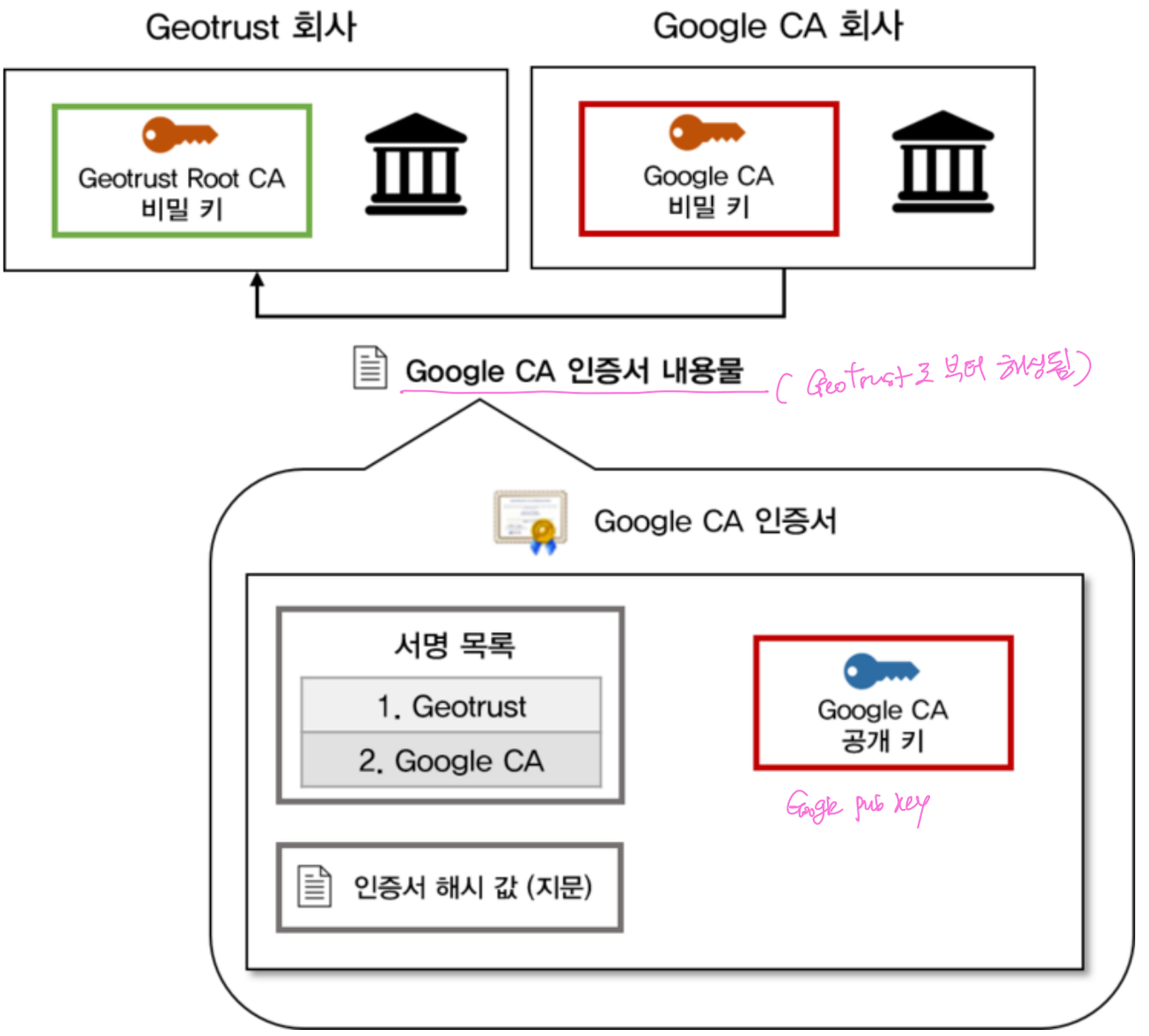
그리고 가장 중요한 것은, (5) 의 값 : **인증서의 내용을 종합해 해시화한 값을 암호화한 값 (지문)** 이 마지막으로 인증서에 기록된다. 이 지문 값에 대해서는 뒤에서 다시 언급한다. 어쨌든, 인증서라는 것은 하나의 파일이며 그 안에는 위와 같이 여러 정보가 담겨져 있다고 알고 넘어가자.

3.2.2 인증서의 구조

3.2.2.1 Geotrust <-> Google CA 예시 (2계층)

인증서는 **계층 구조**로 되어 있다. 보통 3계층 구조로 되어 있고, 가장 최상위에 위치한 인증서는 일반적으로 Root 인증서라고 불린다. 이러한 Root 인증서는 세상 사람들이 모두 신뢰하기로 약속한 기관, 예를 들어 위에서 언급한 Geotrust

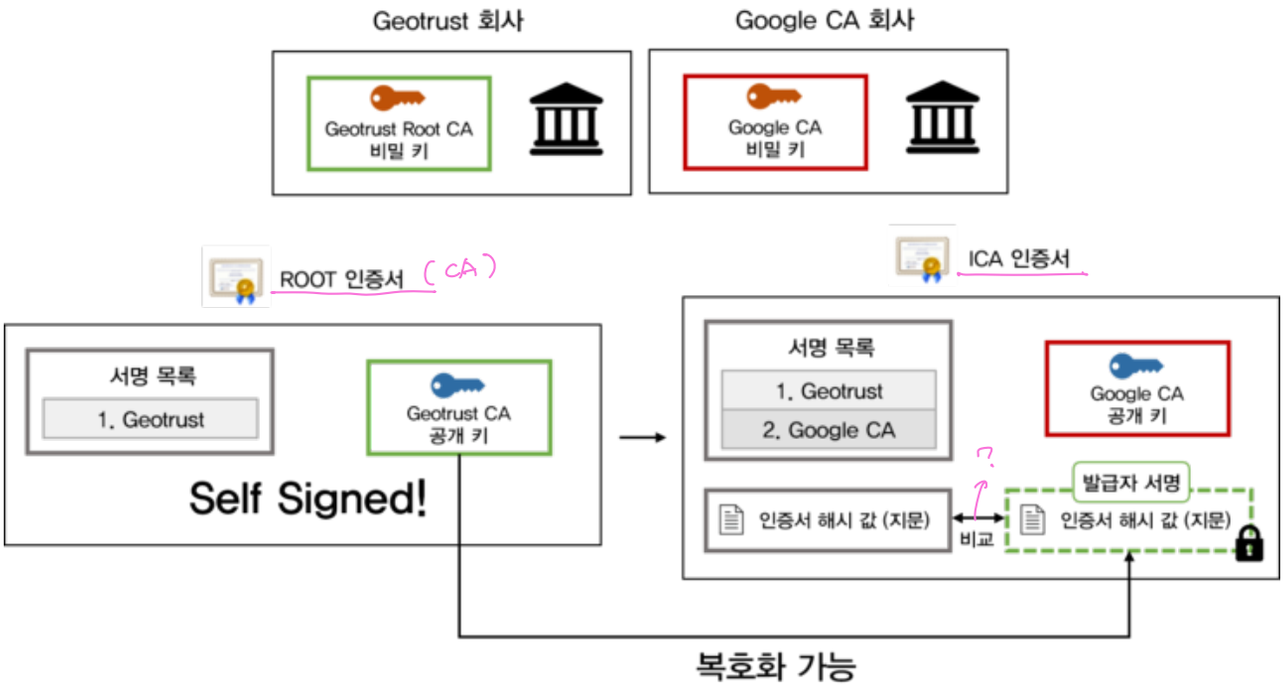
등의 기관에서 발행한 인증서가 된다. 이러한 인증서는 일반적으로 웹 브라우저 등에 미리 내장되어 있으며, 해당 인증서에 대응하는 공개 키 또한 인증서 내부에 포함되어 있다.



위 그림을 기준으로 설명하면, 가장 최상위 Root 인증서의 주인은 Geotrust 회사이다. 자신의 인증서를 만들고 싶은 Google CA 회사는 Geotrust에게 이렇게 요청한다. **'당신네들은 신뢰 가능한 회사로 알려져 있죠. 나는 Google CA 회사의 인증서를 만드려고 해요. 내 인증서의 내용물을 드릴테니, 인증서의 내용물을**

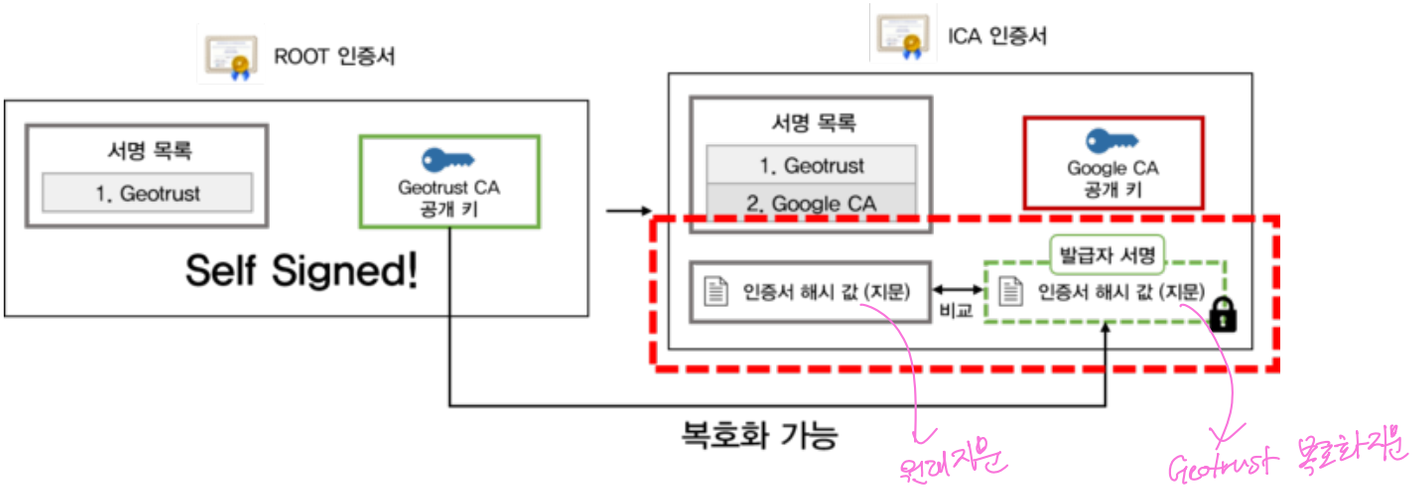
종합해 해시화한 값을 당신네 (Geotrust) 회사의 비밀 키로 암호화 해줄래요?' 345

인증서의 내용물을 종합해 해시화한 값을 Geotrust의 비밀 키로 암호화해달라고 하는 이유는 간단하지만 매우 중요하다. 세상 사람들은 Geotrust 회사의 공개 키를 전부 가지고 있고, 이 공개 키로 복호화되는 데이터는 신뢰할 수 있는 데이터임을 알고 있다. 만약 세상 사람들이 Google CA 인증서를 누군가로부터 전달받은 뒤, Google CA 인증서 내부의 암호화된 해시 값을 Geotrust의 공개 키를 이용해 복호화에 성공했다고 가정해보자. 그렇다면 **Google CA의 인증서의 내용물에 대한 해시 값을, Geotrust가 Geotrust의 비밀 키로 암호화 해준것일테니, Google CA 또한 신뢰할 수 있다**고 간주하는 것이다.



이러한 원리를 **Chain of Trust**라고 부른다. 상위 계층의 인증서 (Geotrust) 가 신뢰할 수 있는 기관이라면, 해당 인증서 (Geotrust) 의 비밀 키로 암호화된 하위 인증서 (Google CA) 또한 신뢰 가능하다고 간주한다.

그렇다면 굳이 인증서의 내용물로 해시 값을 만든 뒤, 이 값을 Geotrust의 비밀키로 암호화한 이유는 무엇일까?



사람들이 모두 갖고 있는 Geotrust의 공개 키를 이용해, Geotrust의 하위 인증서인 Google CA 인증서의 암호화된 해시값을 복호화했고, Google CA의 인증서를 생성할 당시에 사용한 해시 값을 얻었다고 가정해보자. 여기까지만 보면 Google CA의 인증서는 어쨌든 Geotrust에 의해 신뢰 가능하다고 생각될 수는 있다. 그런데, **(1) 복호화된 해시 값과 (2) 인증서에 들어 있는 내용물을 다시 해시값**으로 만들어 비교를 해보니 **동일한 값이 아니라면 어떨까?**

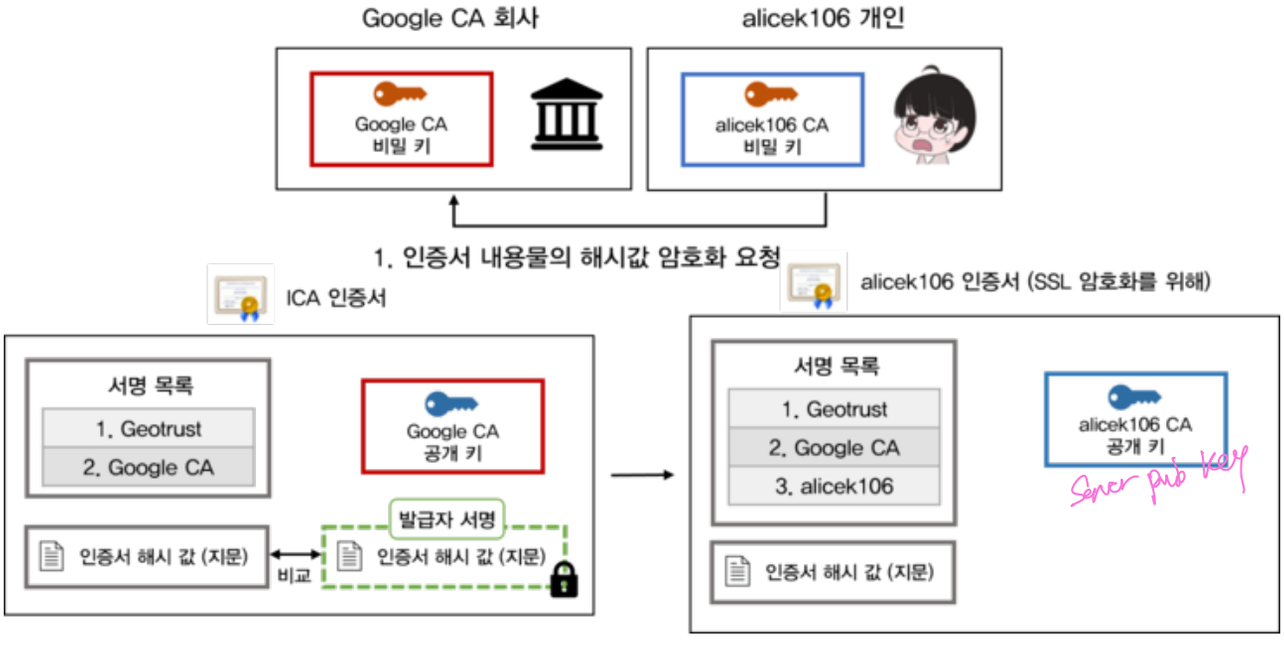
이는 곧 Google CA 인증서의 내용물이 누군가에 의해 변조되었음을 의미한다!

따라서, 상위 인증서 기관의 공개 키로 하위 기관의 인증서 해시 값을 복호화 한다는 것은 두 가지 효과를 얻을 수 있음을 알 수 있다. 먼저, (1) Chain of Trust의 원리에 의해 하위 인증서가 신뢰할 수 있는지를 알 수 있으며, (2) 하위 인증서의 내용물이 변조되었는지를 알 수 있게 된다. 참으로 똑똑한 발상이 아닐 수 없다. 세상엔 변태들이 참 많다고 생각했다.

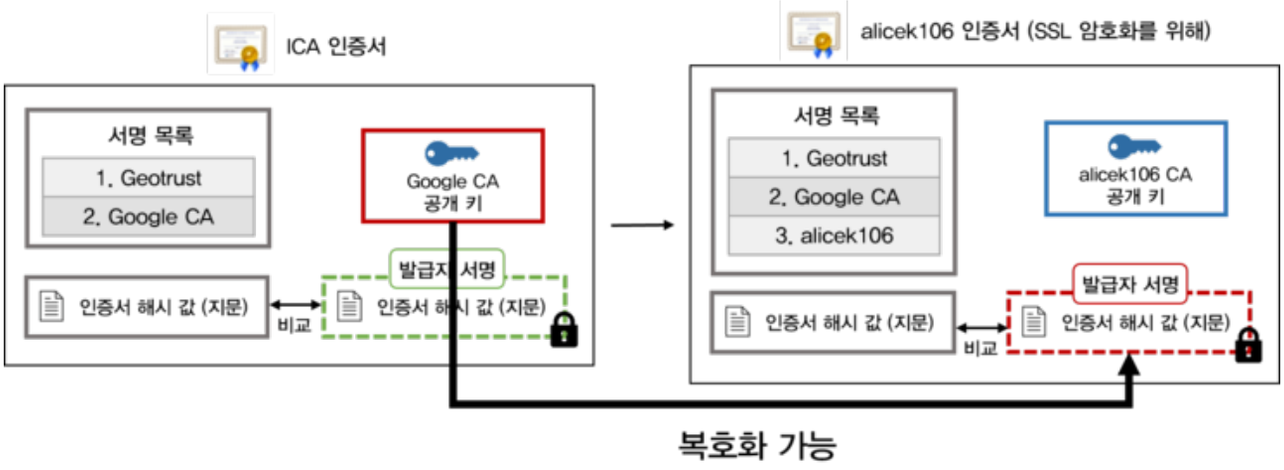
3.2.2.2 Geotrust <-> Google CA <-> 개인 인증서 예시 (3계층)

위 예시는 Geotrust와 Google 간의 예시를 보여주었지만, 실제로 웹 브라우저에는 Geotrust와 Google CA의 인증서가 신뢰할 수 있도록 미리 등록되어 있다. 최상위 인증서 기관과 중간 인증서 기관까지는 사실상 신뢰할 수 있도록 설정되어 있기 때문에 우리가 Geotrust와 Google CA에 대해 의심할 필요는 없다고 보면 된다. 즉 중간 인증서 기관은 모두 신뢰할 수 있으며 Root CA 인증서와 구분짓기 위해 **ICA (Intermediate CA)** 라고 부른다. 이러한 ICA 또한 일반적으로 신용도가 높은 회사 (예를 들면 Google) 로 구성되어 있기 때문에 신뢰할 수 있는 인증서 목록에 기본적으로 ICA가 등록되어 있다.

그렇다면 이번에는 Google과 같은 큰 회사의 ICA가 상위 인증서가 되는 경우를 살펴보도록 하자. 나와 같은 개인이 인증서를 발급받으려면 일반적으로 Root 인증서가 아닌 ICA에 인증서의 사인 (인증서 내용물 해시값의 암호화) 를 요청하게 된다.



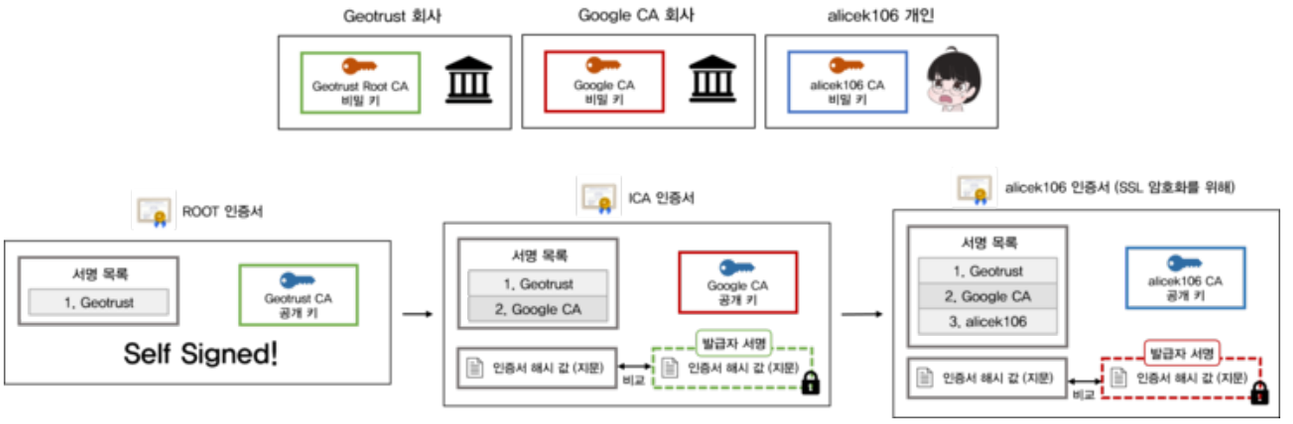
필자와 같은 개인이 Google CA에 인증서 생성을 요청 (해시값의 암호화 요청)을 보내면 Google CA는 Google CA의 비밀키를 이용해 해시값을 암호화함으로써 인증서에 서명하게 된다. Geotrust와 Google CA 예시에서 보았던 것처럼, alicek106 인증서에 저장된 암호화된 해시 값은 Google CA로만 복호화할 수 있다.



우리는 Google CA가 신뢰할 수 있는 기관임을 Geotrust의 공개 키 (Geotrust의 인증서) 를 통해 검증했다. 그런데 Google CA의 공개 키를 통해 alicek106 인증서에 내장된 암호화된 해시 값의 복호화에 성공했다면, alicek106의 인증서 또한 Google CA에 의해 신뢰할 수 있음을 인증하게 된다.

여기까지의 내용을 종합해 그림으로 정리하면 아래와 같다.

최종적으로 alicek106은 Geotrust에 의해 신뢰될 수 있는 인증서를 갖게 된다.



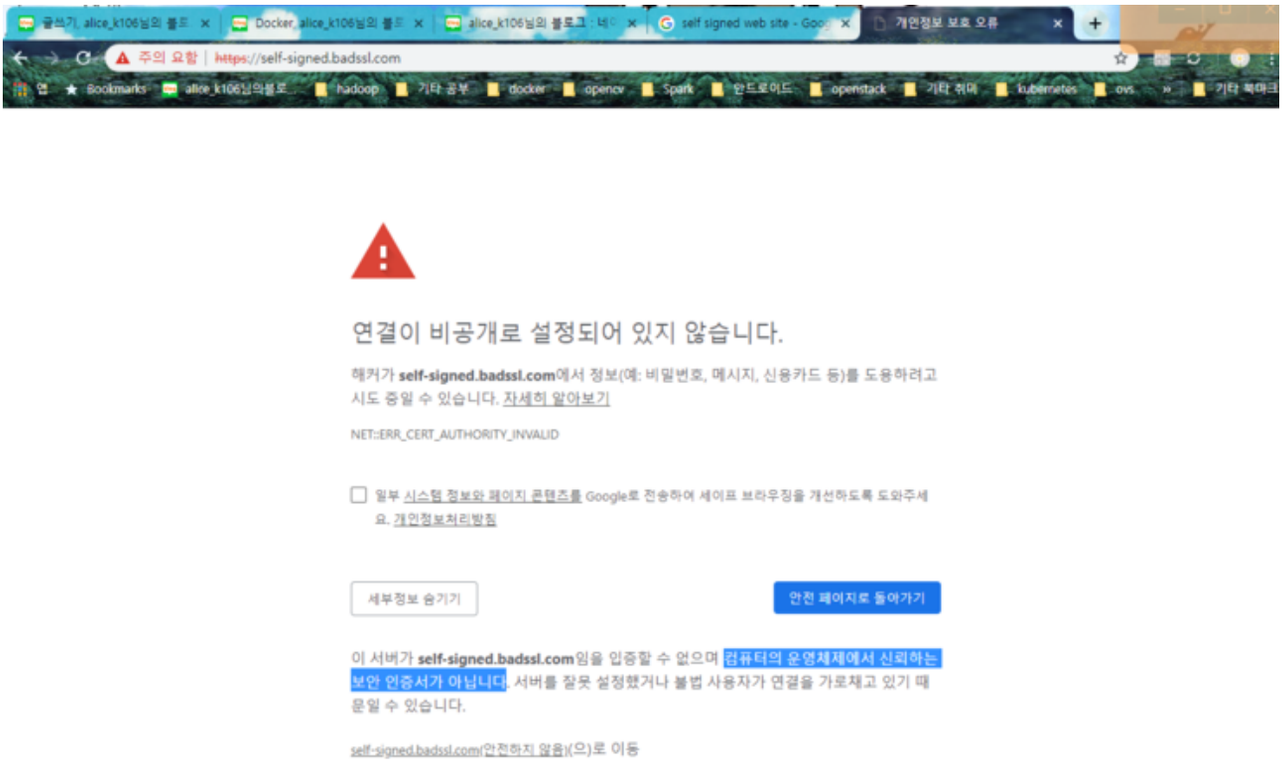
여기서 한 가지 알아둘 점은, 최상위 인증서인 Root 인증서는 스스로를 사인해줄 상위 기관이 없기 때문에 스스로 보증 (Self-signed) 하게 된다. Geotrust와 같은 Root CA가 발행하는 인증서가 이에 해당하며, 이러한 최상위 인증 기관은 모두가 믿기로 약속했기 때문에 크게 문제가 되지 않는다고 한다.

여기서 한 가지 의문점이 들 수 있는데, 왜 굳이 ICA라는 중간 단계가 존재하는 것일까? Root CA가 개인 인증서까지 서명하면 좀 더 직관적이고 쉬운 구조가 될 수도 있는데 말이다. Quora6 및 Stack Overflow의 답변에 의하면, 인증을 수행하는 기관의 인증서가 손상되거나 비밀키가 유출될 경우 이를 철회 (revoke) 할 수 있도록 ICA라는 중간 단계를 두었다고 한다. Root 인증서가 직접 개인 인증서에 서명하는 것에 비해 한 단계 인증 Layer가 더 존재하게 되므로, 더욱 강력한 보안 체계를 구축할 수 있게 된다.

참고로 인증서가 훼손되거나 손상될 가능성이 있을 경우 철회 (Revoke) 하는 과정 또한 존재하며, 이에 대한 자세한 설명은.... 본문이 너무 길어지기에 별도로 설명하지 않는다. 그러나 내가 읽어보니, 이 또한 잘 갖춰진 시스템이라는 생각이 들기에... 관심이 있다면 찾아서 읽어보는 것을 권장하고 싶다.

3.2.2.3 만약 alicek106의 인증서가 신뢰할 수 없다면?

필자의 인증서인 alicek106을 다른 사람의 웹 브라우저가 전달받았고, alicek106 인증서에 대한 신뢰성을 검증하려고 한다. 그러나 웹 브라우저에 미리 등록되어 있는 어떠한 인증서 (공개 키) 도 alicek106 인증서에 저장된 암호화된 해시값을 복호화할 수 없다고 가정해보자.

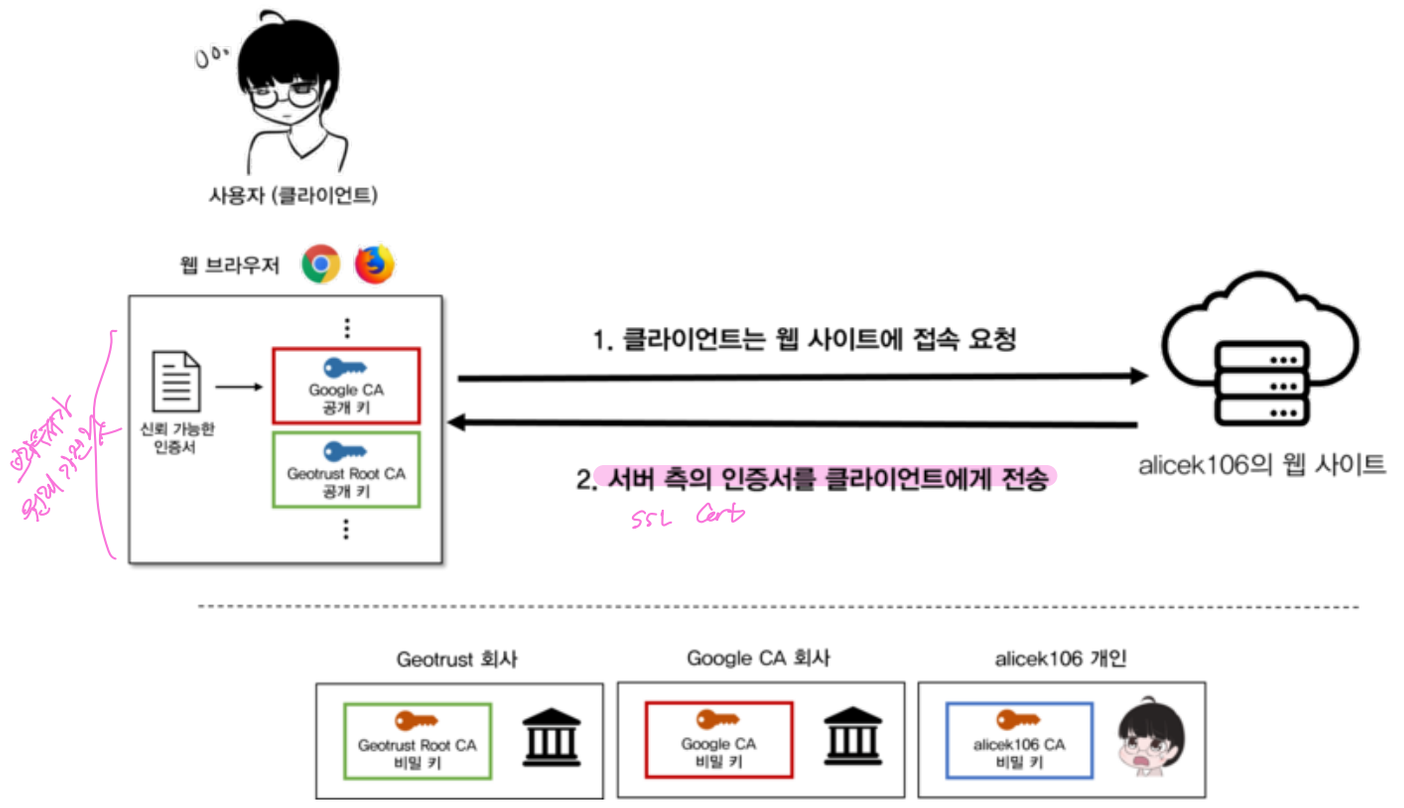


지겨울 정도로 여러 번 말했지만, 우리가 신뢰하는 기관은 미리 정해져 있으며 Geotrust 등이 이에 해당된다고 위에서 언급했었다. 신뢰할 수 없는 인증서라는 것은, Geotrust과 같은 Root CA를 제외한, 우리가 신뢰하지 않는 제 3의 기관이 발급한 인증서 (공개 키) 라는 뜻이다. 신뢰할 수 없는 기관의 인증서를 사용하는 웹페이지에 접속할 경우, 암호화된 패킷이 제 3자에 의해 다시 복호화될 수 있기 때문에 보안 상의 취약점이 발생하게 된다. 따라서 웹 브라우저는 **'신뢰할 수 없는 인증서'** 라는 경고를 출력하거나 접근 자체를 허용하지 않는다. (이는 지금도

조금 헷갈리는데, 신뢰할수 없는 CA라 하더라도 사실 암호화 연결은 가능하다. 그러나 인증서 변조 등과 같은 여러가지 문제가 발생할 수 있다고 한다.)

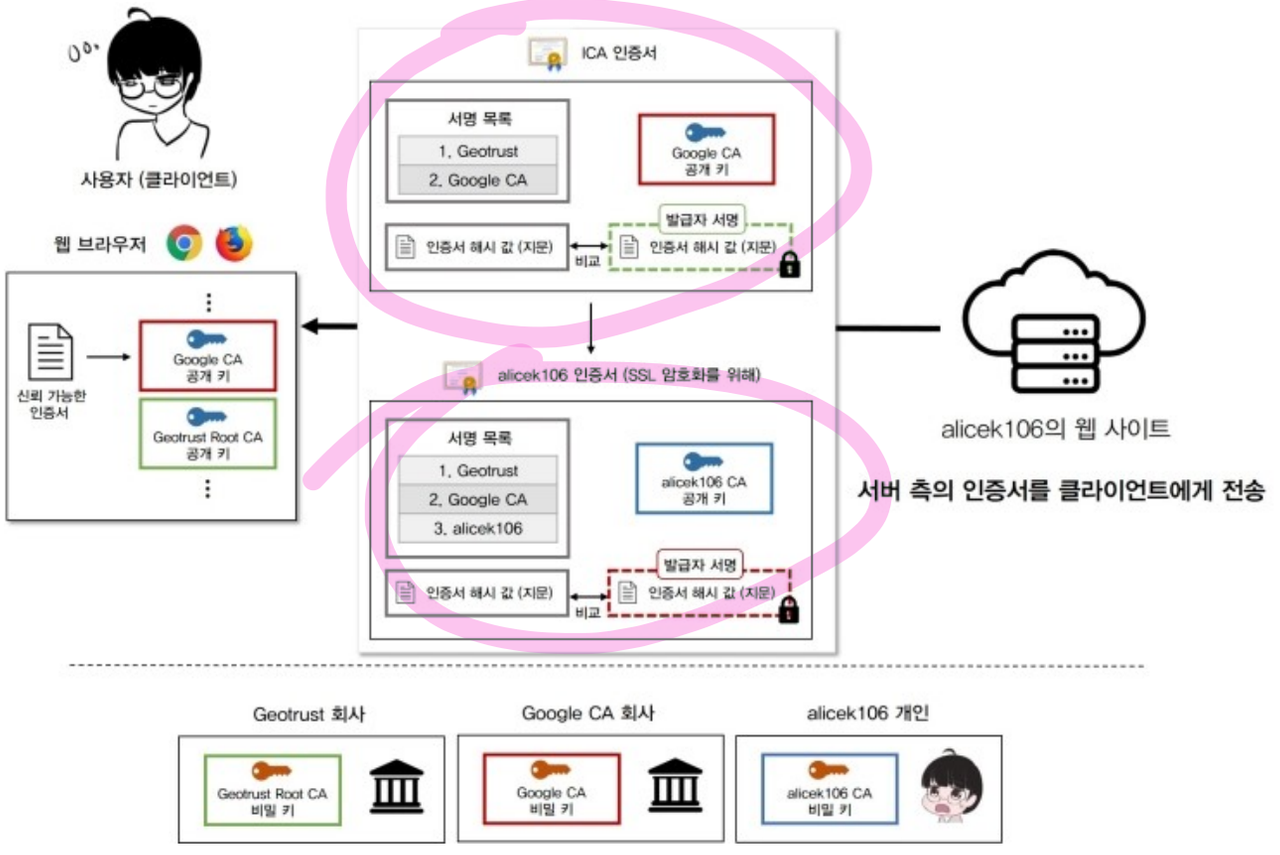
3.3 보안 연결 수립 과정

개인이 ICA로부터 인증서를 발급받았고, 이를 웹 사이트에 등록했다. 그리고 사용자 (클라이언트) 가 해당 웹 서버에 접속해 데이터를 주고받으려고 할 때, 보안 연결 수립을 위해 어떠한 과정이 발생하는지 알아보자.



이 글의 [2. 보안 연결의 순서] 항목에 있는 그림을 좀 더 구체화하면 위와 같이 표현할 수 있다. 클라이언트가 웹 사이트에 접근하면 서버 측에서는 자신의 인증

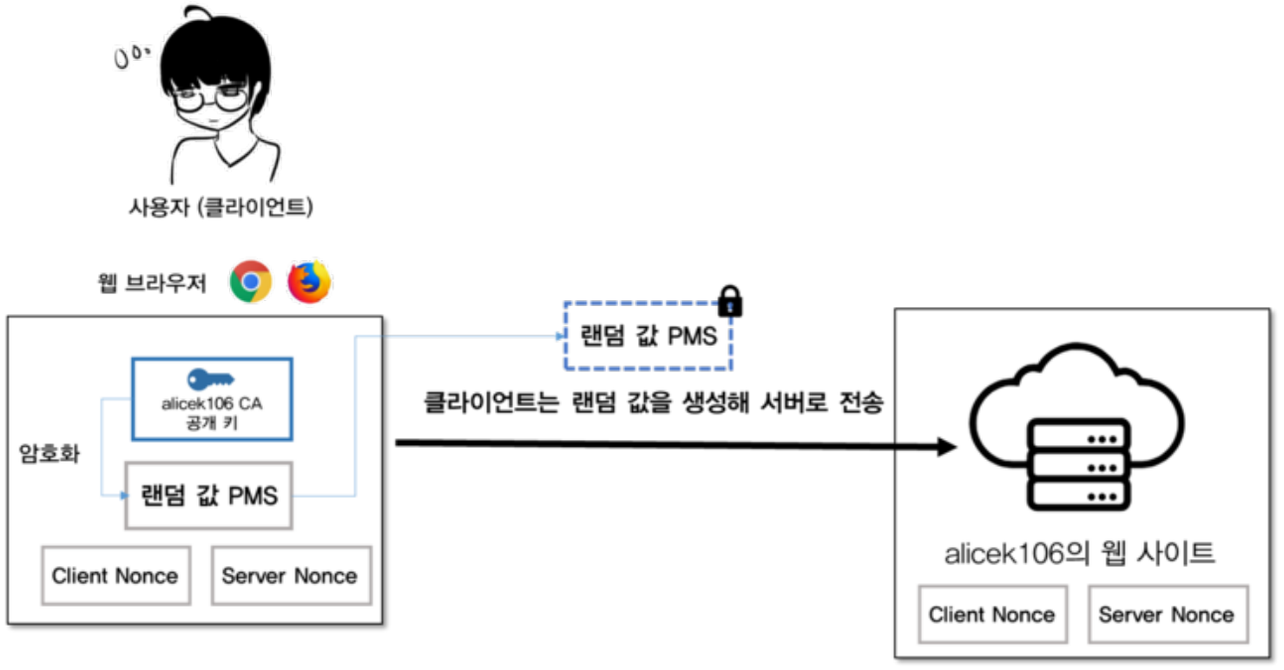
서 및 ICA 인증서를 클라이언트에게 전송한다. 물론, 클라이언트의 웹 브라우저에는 해당 인증서가 신뢰할 수 있는지를 확인하기 위해 '신뢰할 수 있는 인증서' (공개 키) 가 미리 등록되어 있다. 위에서 말했다시피, '신뢰할 수 있는 인증서' 목록에는 Geotrust와 같은 Root CA의 인증서, Google CA같은 ICA의 인증서가 포함되어 있다.



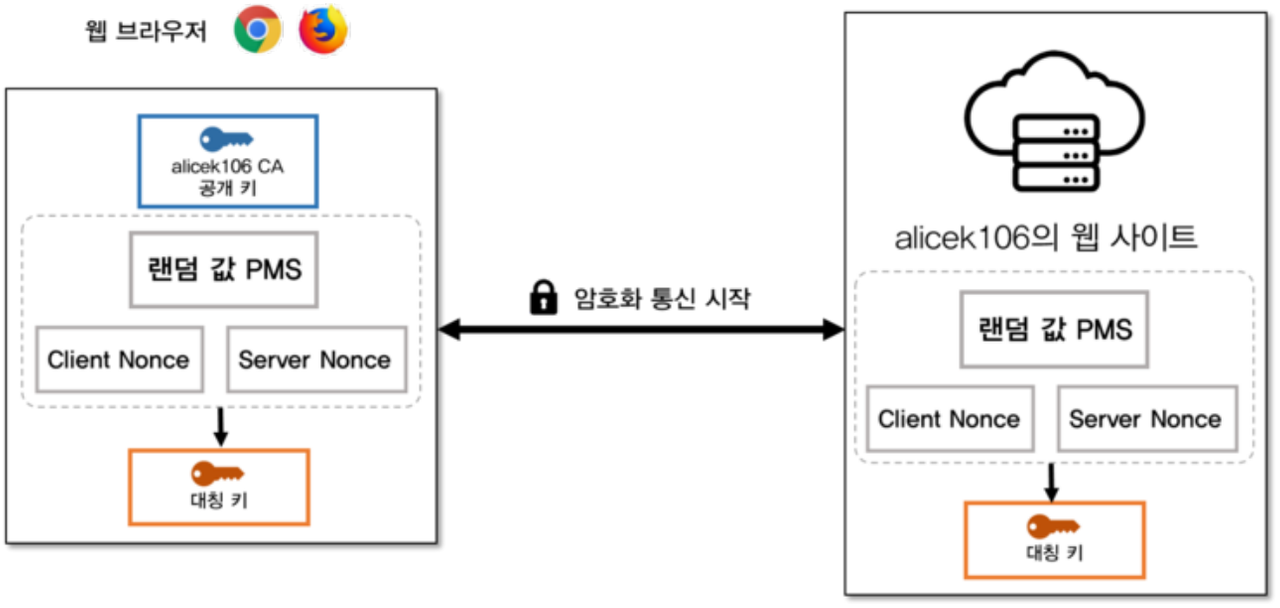
위 그림과 같이 클라이언트에게 ICA의 인증서와 alicek106의 인증서가 함께 전달되고, 웹 브라우저에 등록된 인증서 (공개 키) 에 의해 alicek106 인증서는 신뢰할 수 있는지를 확인하는 작업을 수행한다.7 alicek106 인증서가 상위 인증서 (Google CA)에 의해 신뢰될 수 있음을 확인하고 나면 클라이언트는 **alicek106 인증서에 내장되어 있는 alicek106의 공개 키를 꺼내 사용하게 된다.**

결론부터 말하자면, 비대칭키의 암호화/복호화는 대칭키에 비해 자원을 많이 사

용하기 때문에, 클라이언트와 서버가 최초로 비밀 값을 주고 받을 때 (Client Key Exchange 단계) 에만 비대칭키를 통해 암호화한다. **즉 서버와 클라이언트 간에 처음 연결을 수립할 때에만 비대칭키를 사용하고, 보안 연결 수립 뒤에는 대칭 키로 패킷을 암호화해 전송한다.** 그 과정을 나타내면 아래 그림과 같다.



클라이언트와 서버는 연결을 수립하기 이전에, Handshake 과정에서 각각 Nonce (Seed 값) 를 주고 받는다. 즉 클라이언트는 클라이언트의 Nonce 값을 생성해서 서버로 전송하고, 서버는 서버의 Nonce 값을 생성해 클라이언트로 전송한다. 여기까지는 아직 데이터가 암호화되어서 전송되지 않는다! 그리고 나서야 클라이언트는 PMS (Pre-master secret) 라고 불리는 일종의 난수 값을 생성하고, **인증서로부터 획득한 alicek106의 공개 키를 이용해 PMS 값을 암호화한 뒤 이를 서버로 전송한다.8**

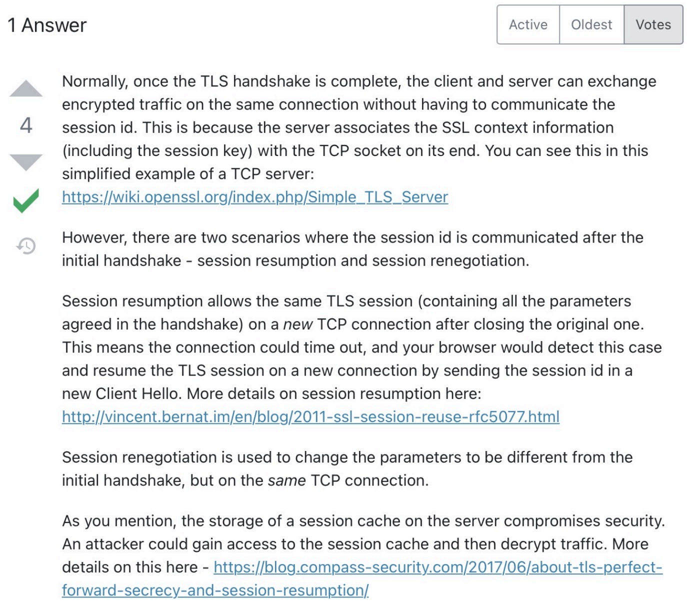


서버와 클라이언트는 {클라이언트/서버 Nonce 값, PMS 값} 을 통해 비로소 대칭 키를 생성하게 되고, 서버와 클라이언트는 이 대칭 키를 이용해 데이터를 암호화하게 된다. 길고 긴 암호화 과정이 끝났기에, 여기서부터 본격적으로 암호화된 보안 통신이 시작된다!910

사실 openssl로 csr, key 파일 발급하고 사인하고 self-signed Root cert 만드는 것도 하려고 했는데.. 시간이 너무 늦어 여기까지만 작성한다. 이 부분까지 직접 openssl 명령어로 쳐보고 나면 어느 정도 감이 올 것이니, 직접 해보는 것을 추천한다. (나의 경우에는 쿠버네티스 CA로 테스트했다)

끝.

1. 엄연히 말하자면 TLS/SSL은 다른 개념이다. SSL을 좀 더 정교하게 만든 것이 TLS이지만 일반적으로 openssl과 같은 명령어를 사용할 때는 TLS/SSL의 구현체를 사용하게 되므로 두 가지를 함께 취급하기도 한다.
2. 인증서에는 일련 번호, 서명 알고리즘, 버전 (x509 version 3 등), 유효 기간,


1 Answer

Active Oldest Votes

4

Normally, once the TLS handshake is complete, the client and server can exchange encrypted traffic on the same connection without having to communicate the session id. This is because the server associates the SSL context information (including the session key) with the TCP socket on its end. You can see this in this simplified example of a TCP server: https://wiki.openssl.org/index.php/Simple_TLS_Server

However, there are two scenarios where the session id is communicated after the initial handshake - session resumption and session renegotiation.

Session resumption allows the same TLS session (containing all the parameters agreed in the handshake) on a *new* TCP connection after closing the original one. This means the connection could time out, and your browser would detect this case and resume the TLS session on a new connection by sending the session id in a new Client Hello. More details on session resumption here: http://vincent.bernat.im/en/blog/2011-ssl-session-reuse-rfc5077.html

Session renegotiation is used to change the parameters to be different from the initial handshake, but on the *same* TCP connection.

As you mention, the storage of a session cache on the server compromises security. An attacker could gain access to the session cache and then decrypt traffic. More details on this here - https://blog.compass-security.com/2017/06/about-tls-perfect-forward-secrecy-and-session-resumption/

급 대상 등이 포함되어 있다. 뒤에서 다시

일이 존재한다. 인증서의 정보 (Common
함해 인증서 서명 요청 파일을 작성해
이를 사인해 완성된 인증서를 돌려주는 식
DN (Distinguished Name) 이라고 부른
봐야 감이 온다..
된다. 자세한 내용은 여기를 참고.
https://www.lesstif.com/pages/viewpage.action?pageId=26083556

5. csr에서 공개 키를 추출하는 것도 가능하다. https://superuser.com/questions/1233571/how-to-export-public-key-from-certificate-signing-request 참고
6. https://www.quora.com/What-is-the-main-purpose-of-the-intermediate-certification-authorities
7. 자세한 설명은 이 동영상을 참고. https://www.youtube.com/watch?v=heacxYUnFHA
8. https://www.tuwlab.com/ece/26967 를 참고한다.
9. 이 뒤에 Handshare Integrity Check라고 하는 별도의 무결성 검증 과정이 존재하지만, 본 포스트에서 자세히 다루지는 않는다.
10. 더 자세한 과정에 대해서는 https://blog.naver.com/PostView.nhn?blogId=nttkak&logNo=20130246586&categoryNo=19&viewDate=¤tPage=1&listtype=0 를 참고한다.