JPNT 13(4), 409-423 (2024)
https://doi.org/10.11003/JPNT.2024.13.4.409

JPNT Journal of Positioning, Navigation, and Timing

QZSS 센티미터급 보정 서비스를 위한 간소화된 항법 메시지 인증 프로토콜 설계

전유진[1], 권희용[1], 노재희[2], 이문규[3†]

Design of Compact Navigation Message Authentication Protocol for Japanese QZSS Centimeter Level Augmentation Service

Youjin Jeon[1], Hee-Yong Kwon[1], Jae Hee Noh[2], Mun-Kyu Lee[3†]
[1]Department of Electrical and Computer Engineering, Inha University, Incheon 22212, Korea
[2]Korea Aerospace Research Institute, Daejeon 34133, Korea
[3]Department of Computer Engineering, Inha University, Incheon 22212, Korea

ABSTRACT

Satellite navigation systems provide services such as positioning, navigation, and timing (PNT), by transmitting navigation messages through signals. However, there is possibility that the signals can be spoofed and fake PNT information can be transmitted to the receiver. To prevent spoofing attacks, navigation messages should be authenticated. Although the Japanese Quasi-Zenith Satellite System (QZSS) recently adopted authentication services for L1C/A, L1C, and L5, it does not yet provide authentication services for its Centimeter Level Augmentation Service (CLAS). To adopt authentication functionality for the existing CLAS that already has its own message structure, only reserved fields can be used for authentication. In this study, we propose two compact navigation message authentication protocols using reserved fields of QZSS CLAS message. They are based on the Elliptic Curve Digital Signature Algorithm (ECDSA) and Timed Efficient Stream Loss-tolerant Authentication (TESLA), which have been widely considered for other signal authentication services such as GPS Chimera and Galileo Open Service Navigation Message Authentication (OSNMA). According to our experiments, the proposed ECDSA-only method needs 6 to 12 min for authentication and the proposed ECDSA-TESLA method requires 4 to 6 min.

Keywords: satellite signal, navigation message, QZSS, CLAS, authentication
주요어: 위성 신호, 항법 메시지, QZSS, CLAS, 인증

1. INTRODUCTION

위성 항법 시스템은 positioning, navigation, timing 등에 사용되는 메시지를 위성 신호를 통해 전송한다. 위성 항법 시스템은 전 세계적으로 서비스를 제공하는 Global Navigation Satellite System (GNSS)과 특정 지역에만 서비스를 제공하는 Regional Navigation Satellite System (RNSS)로 나눌 수 있다. 현재 서비스 중인 위성 항법 시스템 중 미국의 Global Positioning System (GPS), 유럽의 Galileo, 러시아의 GLObal NAVigation Satellite System (GLONASS), 그리고 중국의 BeiDou가 GNSS에 해당하며, 인도의 Indian Regional Navigational Satellite System (IRNSS)과 일본의 Quasi-Zenith Satellite System (QZSS)는 RNSS에 해당한다.

위성 항법 시스템을 활용한 대표적인 서비스에는 Position, Navigation, and Timing (PNT), Search and Rescue (SAR) 등이 있다. 이때 위성에서 제공되는 신호 대신 공격자가 자신의 가짜 신호를 위성으로부터 온 신호인 것처럼 꾸며 전송하더라도 수신자는 그것이 위성으로부터 전송된 것인지 공격자가 위조한 것인지 알 수 없게 된다. 이러한 공격을 스푸핑 공격(spoofing attack)이라 한다. 스푸핑을 방지하기 위해 일부 위성 시스템은 신호 인

Received Sep 09, 2024 **Revised** Sep 28, 2024 **Accepted** Oct 03, 2024

†**Corresponding Author** E-mail: mklee@inha.ac.kr

증 서비스를 제공하고 있으며, 일부는 제공할 예정이다. 단, 위성 시스템이 제공하는 모든 신호에 대한 인증을 제공하는 것이 아닌, 특정 신호에 대한 인증만을 제공한다 (Anderson et al. 2017, European GNSS Agency 2023, Cabinet Office 2023).

신호 인증 방법은 크게 대역 확산 코드 인증(Spreading Code Authentication, SCA)과 항법 메시지 인증(Navigation Message Authentication, NMA)으로 나눌 수 있다. SCA는 항법 메시지를 전송하는 대역 확산 코드(spreading code)에 대한 인증을 하는 방법으로, 신호에 워터마크와 같은 마커(marker)를 삽입하여, 해당 신호가 송신자로부터 전송된 것임을 확인할 수 있도록 하는 방법이다. NMA는 전송되는 항법 메시지를 이용한 인증 방법으로, 송신자는 전송하는 메시지를 이용하여 전자서명 또는 메시지 인증 코드(Message Authentication Code, MAC)를 방송하고, 수신자는 방송된 항법 메시지를 통하여 전자서명 혹은 MAC을 인증함으로써 메시지가 정상적인 송신자로부터 전송된 것인지 확인하는 방법이다.

GPS의 신호 인증 서비스는 Chips-Message Robust Authentication protocol (Chimera)로, L1C 신호에 대한 인증을 제공한다. GPS Chimera는 SCA와 NMA를 모두 제공하며, GPS의 NMA는 전자서명을 통해 수행된다. GPS의 NMA는 GPS가 보내는 항법 메시지에 대해 Elliptic Curve Digital Signature Algorithm (ECDSA) P-224를 통해 전자서명을 생성하고, 해당 서명을 검증함으로써 항법 메시지 인증을 수행한다. 또한, 해당 서명을 이용하여 마커를 생성하고, 그것을 대역 확산 코드에 삽입하여 SCA를 제공하기 때문에, 수신자는 수신한 신호에서 마커를 확인하여 신호의 유효성을 검증할 수 있다 (Anderson et al. 2017).

Galileo의 신호 인증 서비스인 Open Service Navigation Message Authentication (OSNMA)은 E1-B Galileo Open Service 신호의 I/NAV 항법 메시지에 대한 인증을 제공한다. Galileo는 일대다 통신에서 효율적인 인증을 위해 개발된 프로토콜인 Timed Efficient Stream Loss-tolerant Authentication (TESLA)를 사용하여 메시지의 무결성을 확인할 수 있는 MAC 태그(tag)를 생성한다. 수신자는 수신한 메시지와 TESLA 프로토콜에 따라 방송된 키(key)를 이용하여 태그를 생성하고, 직접 생성한 태그가 위성으로부터 수신한 태그와 일치하는지 확인함으로써 메시지 인증을 수행한다 (European GNSS Agency 2023).

중국의 BeiDou는 D1과 D2 메시지에 대하여 인증하는 두 가지 방법이 Wu et al. (2019, 2020)에 의해 제안되었으나, 공식적인 BeiDou의 인증 프로토콜이 발표되지는 않았다. Wu et al. (2019)에서 제안된 방법은 ECDSA를 기반으로 하는 인증 방법으로 D1과 D2 메시지에 대한 인증이 가능하며, Wu et al. (2020)에서 제안된 방법은 D2 메시지를 인증하는 TESLA 기반의 인증 방법이다.

일본의 QZSS는 위성항법인증 및 보정(augmentation)을 위해 다양한 신호를 전송하나, 이 중 LNAV, CNAV, CNAV2 메시지에 대한 NMA만을 제공한다. 모든 인증은 전자서명을 통해 수행되며, 전자서명은 ECDSA P-256을 사용한다. ECDSA P-256의 서명 결과는 512 비트이며, 서명을 전부 전송한 후, 해당 서명을 생성할 때 사용한 인증 대상 메시지를 전송한다. 이때, LNAV, CNAV, CNAV2의 서명 전송 주기는 각각 다르다. LNAV는 240초마다 한 개의 서명을 전송하며, CNAV는 216초, 216초, 288초의 주기를 반복하며 매번 하나의 서명을 전송하는 반면 CNAV2는 288초, 288초, 144초의 주기를 반복하며 하나의 서명을 전송한다 (Cabinet Office 2023).

QZSS에서는 위성 항법 정밀도를 향상시키기 위해 Centimeter Level Augmentation Service (CLAS)와 같은 보정 신호 전송 서비스도 제공하고 있다. 그러나 이에 대해서는 인증을 제공하지 않고 있어 스푸핑의 위험성이 존재하므로, 이에 대한 인증 프로토콜이 필요하다. CLAS는 이미 제공되는 서비스로, 메시지 구조가 정해져 있어서 인증을 위한 메시지를 추가할 수 없기 때문에, 기존 메시지의 예비 영역(reserved field)을 사용하여 인증 프로토콜을 추가로 적용해야 한다. 본 논문에서는 QZSS CLAS의 예비 영역을 사용한 인증 프로토콜을 제안한다. 예비 영역을 사용하여 인증 프로토콜을 설계하는 경우 충분한 비트 수 확보가 불가능할 수 있기 때문에, 전체 데이터 필드의 5 퍼센트 미만의 크기인 50 비트를 인증에 사용할 수 있는 상황을 가정하여 설계하였다. 본 논문에서는 전자서명만을 사용하여 인증을 수행하는 ECDSA-only 방법과 TESLA 프로토콜을 함께 사용하여 인증을 수행하는 ECDSA-TESLA 두 가지 방법을 제안한다.

일반적으로 TESLA를 사용하는 경우 인증에 필요한 데이터가 적기 때문에 전자서명만을 사용하는 경우보다 현저히 빠른 시간 내에 인증이 가능한 것으로 알려져 있으나, 본 논문에서는 가용 비트 수가 매우 제한적인 특수한 환경의 경우 ECDSA-only와 ECDSA-TESLA의 인증 시간이 큰 차이를 보이지 않는 것을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 항법 메시지 인증을 위한 요소 기술을 설명한다. 3장에서는 QZSS CLAS 신호를 위한 두 가지 인증 프로토콜을 제안한다. 이들은 전자서명만을 사용하는 ECDSA-only와, 전자서명에 TESLA를 조합하여 사용하는 ECDSA-TESLA이며, 모두 QZSS CLAS의 예비 영역 중 50 비트 이하를 사용한다. 4장에서는 두 인증 방법의 성능을 분석하고 다른 위성 항법 시스템의 인증 서비스들과 비교한다. 5장에서는 결론 및 향후 계획을 기술한다.

2. PRELIMINARIES

2.1 QZSS L6 Message

QZSS의 L6 신호는 CLAS 메시지를 전송한다 (Cabinet Office 2022). L6 신호에서 전송되는 메시지 및 갱신 주기, 유효 기간은 Table 1과 같으며, 메시지 구조는 Fig. 1과 같다. 하나의 프레임은 여섯 개의 서브프레임으로 구성되며, 하나의 서브프레임은 다섯 개의 데이터 파트(data part)로 이루어진다. 각 데이터 파트는 1695 비트로 구성되며, 49 비트의 헤더, 1695 비트의 데이터 파트, 그리고 오류 정정을 위한 256 비트의 Reed-Solomon Code와 함께 전송된다. 따라서 하나의 프레임은 총 30개의 데이터 파트로 이루어지며, 50850 비트의 데이터를 가진다. 한 개의 데이터 파트를 방송하는 데에 1초가 걸리며, 하나의 프레임을 모두 전송하는 데에는 총 30초가 걸린다. 30초 동안 보내는 메시지는 서브프레임에 따라 결정되며, 각 서브프레임이 전송하는 메시지는

Table 1. QZSS L6 message (Cabinet Office 2023).

Sub type	Message name	Nominal update interval [sec]	Nominal validity periods [sec]
1	Compact SSR mask	30	1
2	Compact SSR GNSS orbit correction	30	60
3	Compact SSR GNSS clock correction	5	10
4	Compact SSR GNSS satellite code bias	30	60
5	Compact SSR GNSS satellite phase bias	30	60
6	Compact SSR GNSS satellite code and phase bias	30	60
7	Compact SSR GNSS URA	30	60
8	Compact SSR STEC correction	30	60
9	Compact SSR gridded correction	30	60
10	Compact SSR service information	(N/A)	(N/A)
11	Compact SSR GNSS combined correction	5 or 30	10 or 60
12	Compact SSR atmospheric correction	30	60
-	Null message	(N/A)	(N/A)

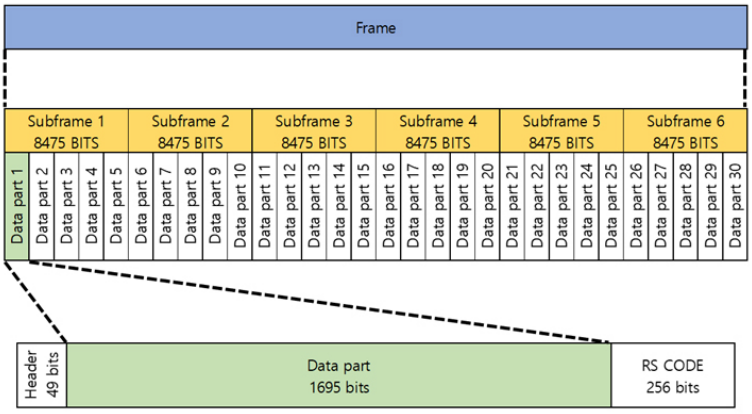

Fig. 1. QZSS L6 Message Structure (Cabinet Office 2022).

Table 2. QZSS L6 transmission pattern of sub type (Cabinet Office 2022).

Sec	Subframe number	Sub type
0 - 4	1	1, 2, 3, 11, 4, 6, 7, 12
5 - 9	2	3, 11, 6, 12
10 - 14	3	3, 11, 6, 12
15 - 19	4	3, 11, 6, 12
20 - 24	5	3, 11, 6, 12
25 - 29	6	3, 11, 6, 12

Table 2와 같다.

서브프레임 6의 서브타입(sub type)별 전송 패턴은 Fig. 2와 같으며, 본 논문에서는 각 프레임의 마지막인 서브프레임 6의 데이터 파트 30의 예비 비트를 인증에 사용하는 방법을 제안한다.

2.2 Authentication

인증(authentication)은 인증성(authenticity)과 무결성(integrity)을 보장하기 위해, 수신자가 받은 메시지가 적법한 송신자가 전송한 원래의 메시지와 동일한지 확인하는 작업이다(Stallings 2023). 인증성은 데이터를 보낸 주체가 적법한지 확인하는 것으로, 공격자가 위조된 신호를 만들어서 전송하는 것을 방지한다. 무결성은 수신자가 받은 데이터가 변조가 되지 않았음을 확인하는 것으로, 공격자의 신호 변조를 탐지할 수 있다. 인증은 메시지 인증 코드와 전자서명을 통해 수행된다.

메시지 인증 코드(Message Authentication Code, MAC)는 짧

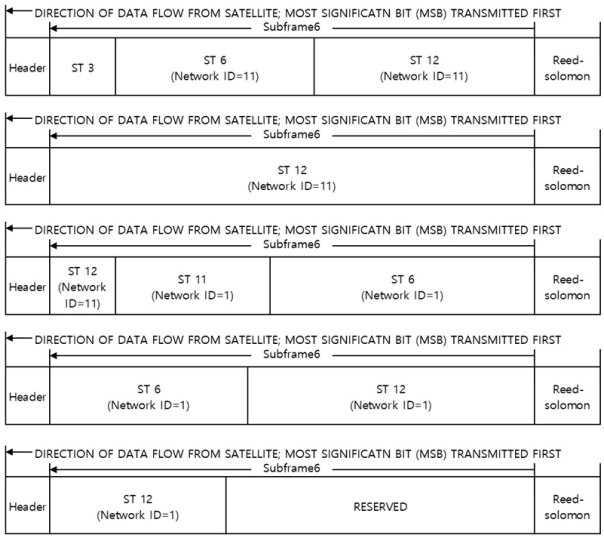

Fig. 2. Subframe 6 broadcast pattern (Cabinet Office 2022).

은 길이의 태그로 메시지의 인증성 및 무결성을 보장한다. MAC의 대표적인 방법에는 해시 기반 메시지 인증 코드(Hash-based Message Authentication Code, HMAC), 암호 기반 메시지 인증 코드(Cipher-based Message Authentication Code, CMAC)가 있다. HMAC은 Secure Hash Algorithm (SHA) 등의 암호학적

Algorithm 1 Key Pair Generation (Hankerson et al. 2006)

Input: Domain parameters $D = (q, FR, S, a, b, P, n, h)$.
Output: Public key Q, private key d.
1: Select $d \in_R [1, n-1]$.
2: Compute $Q = dP$.
3: **Return** (Q, d).

Algorithm 2 ECDSA signature generation (Hankerson et al. 2006)

Input: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, private key d, message m.
Output: Signature (r, s).
1: Select $d \in_R [1, n-1]$.
2: Compute $kP = (x_1, y_1)$ and convert x_1 to an integer $\overline{x}_1$.
3: Compute $r = \overline{x}_1 \bmod n$. If $r = 0$ then go to step 1.
4: Compute $e = H(m)$.
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6: **Return**(r, s).

Algorithm 3 ECDSA signature verification (Hankerson et al. 2006)

Input: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, public key Q, message m, signature (r, s).
Output: Acceptance or rejection of the signature.
1: Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails then return ("Reject the signature").
2: Compute $e = H(m)$.
3: Compute $w = s^{-1} \bmod n$.
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5: Compute $X = u_1 P + u_2 Q$.
6: If $X = \infty$ then return("Reject the signature");
7: Convert the x-coordinate x_1 of X to an integer $\overline{x}_1$; compute $v = \overline{x}_1 \bmod n$.
8: If $v = r$ then **return**("Accept the signature");
Else **return** ("Reject the signature").

해시 함수(cryptographic hash function)를 사용하여 임의의 길이의 인증 대상 메시지에 대한 고정된 길이의 값을 출력하고, 출력값 중 원하는 길이를 추출하여 MAC 태그를 생성한다. CMAC은 Advanced Encryption Standard (AES) 등의 블록 암호를 사용하여 인증 대상 메시지에 대한 암호문을 얻고, 그 암호문에서 임의의 길이를 추출하여 MAC 태그를 만들어서 인증에 사용한다 (Stallings 2023).

전자서명은 공개키 암호 방식(public key cryptosystem)을 이용하여 디지털 세계에서 서명을 구현한 것으로, 송신자는 개인키를 사용하여 서명을 생성하며, 수신자는 공개키를 사용하여 해당 서명을 검증한다. 개인키는 송신자만이 가지고 있기에 인증성이 보장되며, 또한 데이터가 변조되는 경우 서명 검증에 실패하기 때문에 무결성이 보장된다. 대표적인 전자서명 방법에는 리베스트-샤미르-애들먼(Rivest-Shamir-Adleman, RSA)을 사용한 RSA Probabilistic Signature Scheme (RSA-PSS)과 타원 곡선 전자 서명 알고리즘(Elliptic Curve Digital Signature Algorithm, ECDSA)이 있다 (Stallings 2023).

2.3 ECDSA

ECDSA는 타원곡선 기반의 전자서명 방법이다. 개인키와 공개키의 키 쌍을 생성하고, 개인키는 서명을 생성하는 주체가 보관하며, 공개키는 서명을 확인하여 데이터의 신뢰성을 확보하고자 하는 대상 모두에게 배포한다. 송신자는 개인키를 이용하여 데이터에 대한 서명을 생성하고, 수신자는 공개키를 이용하여 수신 데이터에 대한 서명을 검증한다. ECDSA는 타원곡선 이산 대수 문제(Elliptic Curve Discrete Logarithm Problem, ECDLP)에 기반한 알고리즘이다. ECDLP는 타원곡선 상의 두 점 Q와 P가 주어졌을 때, $Q = kP$를 만족하는 자연수 k를 찾는 문제이며, 이는 다항시간(polynomial time) 내에 해결하는 것이 어려운 문제로 잘 알려져 있다.

Algorithm 1은 ECDSA의 키 쌍 생성 알고리즘으로, $[1,n-1]$ 범위의 임의의 정수 d를 선택하여 개인키로 사용한다. 다음으로, $Q=dP$를 계산하고 Q를 공개키로 하여 ECDSA의 키 쌍 (Q,d)를 반환한다. 개인키 d는 서명을 생성하는 주체만이 가지며, 공개키 Q는 서명 검증을 수행할 모든 대상에게 배포한다. 이때 입력값인 도메인 파라미터(domain parameter)는 타원 곡선 체계에 대한 매개변수로, q는 유한체의 원소의 개수, FR은 유한체 원소의 표현 방식(field representation), S는 타원 곡선이 무작위로 생성된 경우 사용된 시드를 나타낸다. a와 b는 타원 곡선 식 $y^2=x^3+ax+b$에서 사용된 계수이며, P는 기저점(base point)을, n은 위수(order)를, h는 여인자(cofactor)를 의미한다.

Algorithm 2는 서명 생성 알고리즘으로, 개인키 d를 사용하여 메시지 m에 대한 서명 (r,s)를 생성한다. P-256 커브를 사용하여 서명을 생성한 경우, r과 s는 각각 256 비트의 크기를 가진다. 본 논문에서는 개인키 d를 사용하여 메시지 m에 대한 서명을 생성하는 과정을 $Sign_d(m)$으로 나타낸다.

ECDSA 전자 서명은 Algorithm 3을 사용하여 검증할 수 있다.

https://doi.org/10.11003/JPNT.2024.13.4.409

서명 검증에는 서명 생성 시에 사용된 개인키 d에 대응되는 공개키 Q, 메시지 m, 그리고 m에 대한 서명 (r,s)가 사용된다. 서명 s와 메시지 m, 공개키 Q를 사용하여 v를 생성하고, 그것이 서명 r과 일치하는 경우에는 인증에 성공하며, 일치하지 않는 경우에는 인증에 실패한다. 본 논문에서는 공개키 Q를 이용하여 메시지 m에 대한 서명 (r,s)를 검증하는 과정을 $Verify_Q\,(m,r,s)$로 나타낸다.

2.4 TESLA

TESLA는 일대다 통신 시 효율적인 메시지 인증이 가능하도록 하는 프로토콜로, Perrig et al. (2000)에 의해 제안되었다. 데이터의 무결성을 확인하기 위해 사용되는 MAC은 송신자와 수신자가 각 세션마다 비밀키를 공유하고, 해당 비밀키로 MAC 태그 생성 및 검증을 수행하는 방법이다. 따라서 일대일 통신의 경우 송신자와 수신자가 안전한 방법으로 비밀키를 공유하고, 그것을 이용하여 메시지의 무결성을 검증한다. 하지만 일대다 통신의 경우, 송신자와 다수의 수신자가 각각의 비밀키를 공유하는 것은 매우 비효율적이다. 또한, 송신자와 모든 수신자가 동일한 비밀키를 사용하는 경우, 비밀키를 가진 수신자가 송신자의 신원을 가장하여 가짜 MAC 태그를 만들어 인증을 통과하도록 공격하는 것이 가능하게 된다. 이러한 문제를 해결하기 위해 제안된 방법이 TESLA이다. TESLA는 해시 함수를 사용하여 MAC 키 사이의 의존 관계를 형성하는 단방향 키 체인을 Eq. (1)과 같이 생성하여, 체인의 각각의 값을 키로 사용한다. 이때, $\boldsymbol{H}(x)$는 해시를 생성하는 SHA(NIST 2015) 등이 사용된다.

$$K_i = \boldsymbol{H}(K_{i+1}) \tag{1}$$

TESLA 키 체인 생성 과정을 구체적으로 설명하면 다음과 같다. 먼저 난수 시드(random seed) σ를 생성하고, 여기에 해시를 적용하여 키 $K_{k-1}=\boldsymbol{H}(\sigma)$를 생성한다. 다음에는 Eq. (1)을 반복적으로 적용하여 $K_{k-2}=\boldsymbol{H}(K_{k-1})$, $K_{k-3}=\boldsymbol{H}(K_{k-2})$, ⋯, $K_0=\boldsymbol{H}(K_1)$로 추가로 k–1개의 키를 생성하고, 마지막으로 생성된 K_0를 루트 키(root key)로 정의한다. 시드로부터 여러 번의 해시를 수행하여 루트 키를 생성한 것이므로, 키 체인의 모든 값들은 최대 k번의 해시를 수행하여 루트 키에 도달하게 된다. 사용자는 루트 키인 K_0부터 K–1번 키 K_{k-1}까지 총 k개의 키를 MAC 태그 생성에 순서대로 사용할 수 있다. 즉, 키 생성 순서와 사용 순서는 서로 반대이다.

키 사이의 의존관계는 Fig. 3과 같이 나타낼 수 있다. 단방향 해시함수가 사용되므로, i+1번째 키를 이용하여 i번째 키를 얻을 수 있으나, i번째 키를 이용하여 i+1번째 키를 얻는 것은 불가능하다.

송신자는 K_0부터 K_{k-1}를 순서대로 사용하며, i+1번째의 키를 사용하여 생성된 MAC 태그를 송신하는 시점에 i번째 키를 방송(broadcast)하여 모든 수신자가 받을 수 있도록 한다. MAC 태그를 생성할 때까지는 MAC 태그 생성에 사용되는 키를 송신자만이 가지고 있으므로, 다른 사람은 MAC 태그를 생성할 수 없으며, 모든 수신자는 MAC 태그가 방송된 이후에 그것을 생성할 때 사용된 MAC 키를 취득하여 해당 키로 생성된 MAC 태그를 검증할 수 있다. Fig. 4는 수신자의 MAC 태그 검증 순서를 나타낸다.

다만, 위의 과정은 이전 세션에 방송된 메시지를 현재 세션에 방송된 MAC 키를 이용하여 검증함으로써 연속된 두 세션이 같은 송신자로부터 방송된 것임을 보장하는 것일 뿐, 이러한 연속된 메시지들이 합법적인 송신자로부터 송신된 것임을 보장하는 것은 아니다. 따라서, MAC 태그 생성에 사용되는 키 체인이 합법적 송신자로부터 송신된 것임을 확인하는, 키 체인 유효성 검증 과정이 필요하다. 키들은 서로 단방향 해시 함수로 연결되어 있으므로, 루트 키를 검증하는 것으로 모든 키의 검증이 가능하다. 따라서 전체 키의 유효성 검증을 위해 루트 키에 대한 인증이 선행되어야 한다. 이를 위해 TESLA 루트 키를 ECDSA를 이용하여 인증한다. 송신자는 생성한 키 체인의 루트 키에 대한 전자서명을 생성하고, 루트 키와 전자서명을 전송한다. 수신자는 루트 키와 전자서명을 취득하고 전자서명을 검증하여 루트 키의 유효성을 확인할 수 있다.

Fig. 3. TESLA key chain.

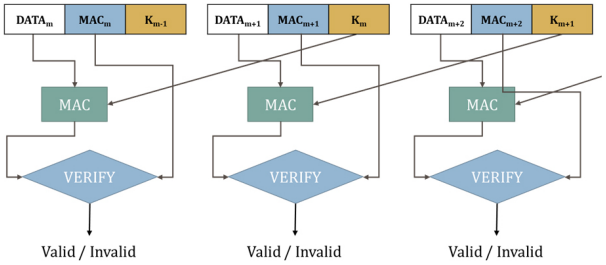

Fig. 4. Receiver MAC verification sequence.

TESLA는 송신자와 수신자가 동일한 키를 이용하는 MAC을 기반으로 하기 때문에, 송신자는 MAC 태그 생성 시 사용된 MAC 키를 그 다음 MAC 태그 전송 시에 공개하고, 수신자는 MAC 태그와 MAC 키를 모두 취득하여야 인증을 수행할 수 있다. 또한 MAC 생성 시 사용된 MAC 키는 공개적으로 방송되기 때문에, 동일한 키를 이용하여 다음 차례의 인증 대상 메시지에 대한 MAC 태그를 생성할 수 없다. 따라서 TESLA를 사용하는 인증의 경우, MAC 태그를 생성할 때마다 MAC 키를 교체해야 한다. 반면, ECDSA를 사용하는 인증 방법의 경우, 송신자는 개인키를 이용하여 인증 메시지를 생성하고, 수신자는 공개키를 이용하여 메시지에 대한 인증을 수행한다. 이를 위해 송신자와 수신자는 각각 개인키와 공개키를 사전에 준비해야 한다. 하나의 키 쌍의 개인키와 공개키는 동일하지 않으며, 공개키는 모두에게 공개적으로 배포할 수 있기 때문에, 송신자의 개인키만 유출되지 않고 안전하게 보관된다면 하나의 키 쌍을 이용하여 여러 차례의 인증을 수행할 수 있다.

3. PROPOSED METHODS

QZSS CLAS는 이미 시행 중인 서비스로, 메시지 구조가 정해져 있기 때문에 인증 프로토콜을 추가하기 위해서는 예비 영역을 사용하여 인증 메시지를 전송해야 한다. 본 장에서는 예비 영역을 활용하여 적은 비트 수로 CLAS 데이터에 대한 인증을 수행

Table 3. Comparable algorithm strength (Barker 2020).

Security bits	Symmetric key	Finite field/discrete logarithm (DSA, DH, MQV)	Integer factorization (RSA)	Elliptic curve (ECDSA, EdDSA, ECDH, ECMQV)
80	2TDEA	L = 1024, N = 160	k = 1024	160 ≤ f ≤ 223
112	3TDEA	L = 2048, N =224	k = 2048	224 ≤ f ≤ 255
128	AES-128	L = 3072, N = 256	k = 3072	256 ≤ f ≤ 383
192	AES-192	L = 7680, N = 384	k = 7680	384 ≤ f ≤ 511
256	AES-256	L = 15360, N = 511	k = 15360	f ≥ 512

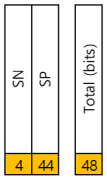

Fig. 5. ECDSA-only authentication message structure.

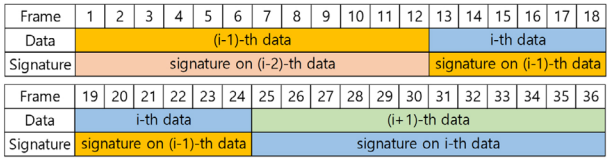

Fig. 6. ECDSA-only transmission pattern.

하는 프로토콜을 설계한다. 예비 영역 활용 시 인증을 위한 충분한 비트 수 확보가 어렵기 때문에 데이터 파트의 5 퍼센트 미만의 크기인 50 비트만을 사용한다. Chimera, OSNMA 등 기존의 다른 항법 인증 프로토콜에서 많이 고려된 ECDSA 기반의 방법인 ECDSA-only 방법과 TESLA 기반의 ECDSA-TESLA 방법 두 가지를 고려한다.

3.1 ECDSA-only Method

이번 절에서는 제안하는 방법 중 ECDSA-only 방법에 대해 기술한다. ECDSA-only 방법은 인증 대상 메시지에 대하여 서명을 생성하여 전송하는 방법으로, QZSS LNAV, CNAV, CNAV2의 인증 방법과 유사하다. ECDSA는 사용되는 타원 곡선에 따라 다른 안전성을 가지며, 미국 국립표준기술연구소(National Institute of Standards and Technology, NIST)의 SP 800-57 (Barker 2020)에 따라 Table 3과 같은 안전성을 가진다. NIST SP 800-57에 따라, 현재 최소 112 비트의 안전성을 만족해야 하지만 2030년 이후에는 128 비트 이상의 안전성만 허용되므로 본 논문에서는 안전한 서명을 위하여 128 비트의 안전성을 만족하는 ECDSA P-256을 사용한다 (Barker 2020).

ECDSA P-256의 서명 크기는 512 비트로, 본 논문에서 가정한 상황과 같이 가용 비트 수가 제한적인 상황에서는 여러 프레임에 걸쳐서 하나의 서명을 전송하기 때문에, 서명 전송에 필요한 시간이 길어진다. 이에 따라 인증 주기가 길어지므로, 한 번에 인증해야 하는 메시지의 길이 또한 길어진다.

각 프레임에서 50 비트 이하를 사용하여 512 비트의 서명을 전송하고자 하는 경우, 하나의 프레임에 서명 전체가 포함될 수 없으므로 서명은 여러 조각으로 나뉘어 전송된다. 제안 방법에서는 하나의 프레임 당 44 비트를 할당하여 총 12개의 프레임에 서명을 균일하게 배분하되, 마지막 프레임에는 서명 조각을 위해 28 비트를 할당하고 나머지 16 비트는 패딩(padding)으로 채운다. 또한, 각 프레임에는 현재 프레임에 포함된 서명 조각이 몇 번째 조각인지를 나타내기 위해 서명 조각의 순서를 나타내는 4 비트 필드를 추가하여, 하나의 프레임 당 총 48 비트를 사용한다. 각 프레임에서 전송하는 인증 메시지의 구조는 Fig. 5와 같다. Fig. 5의 SN은 signature part number로, 현재 프레임에서 전송하는 서명의 순서를 나타내며, SP는 signature part로, 전체 512 비트 중 44 비트의 서명 조각을 의미한다.

3.1.1 ECDSA-only transmitter

본 논문에서 제안하는 방법은 QZSS L6 신호에서 전송하는 실시간 보정 메시지에 대한 인증이므로, 전송할 메시지를 사전에 취득하여 서명을 생성할 수 없다. 따라서 실시간으로 메시지를 전송한 후, 전송된 메시지에 대한 사후 인증을 한다. 송신자는 하나의 서명을 12개의 프레임에 나누어서 전송한다. 이때 전송되는 전자서명은 해당 서명이 전송되기 직전의 12개의 프레임에서 전송한 메시지에 대한 인증을 수행한다. 예를 들어, 프레임 1에서 프레임 12에 전송된 메시지는 프레임 13에서 프레임 24에 걸쳐 전송되는 전자서명을 통해 인증된다. Fig. 6은 ECDSA-only 방법의 데이터 전송 패턴을 보인다. 수신자가 한 번의 인증을 수행하기 위해서는 12개의 프레임 동안 전송되는 인증 대상 메시지와 이후의 12개의 프레임동안 전송되는 서명을 모두 취득해야 한다.

전자서명은 12개의 프레임에 나뉘어 전송되므로, 전자서명의 일부와 함께 현재 전송하는 값이 서명의 몇 번째에 위치하는 값인지를 수신자가 알 수 있도록 해야 한다. Fig. 5의 4 비트의 SN은 전송하는 서명 조각이 몇 번째 조각인지 나타내는 값으로, 0부터 11의 값을 가진다. 512 비트의 서명에서 MSB를 0, LSB를 511번 비트라고 할 때, SN이 0인 경우는 현재 전송하는 44 비트의 값이 전자서명의 0번부터 43번까지의 44 비트임을 나타내며, SN이 1인 경우에는 44번부터 87번까지의 44 비트를 전송함을 나타낸다. 각 전자서명의 전송이 시작될 때는 SN이 0을 가지며, 서명의 전송이 끝날 때에는 SN이 11을 가진다.

ECDSA-only 방법에서 송신자는 Algorithm 4에 따라 방송할 인증 메시지를 생성한다. 하나의 서명으로 12개의 프레임에 걸쳐 전송된 메시지에 대한 인증을 수행하므로, 인증 대상인 12개의 프레임 $(v_1,\cdots,v_{11})$를 연접(concatenate)한 후 Algorithm 2를 사용하여 전자서명을 생성한다. 전자서명 생성 시, 개인키 d로 V를 연접한 메시지 m에 대한 서명을 생성한다. 하나의 프레임에서 인증 메시지는 44 비트의 서명 조각을 전송하므로, (r,s)를 연접한 뒤 44 비트씩 추출한다. Zero_padding(m,n)은 입력 메시지 m을 n 비트로 만들기 위해 m의 LSB 뒤에 n 비트가 되도록 0을 채워 패딩

Algorithm 4 ECDSA-only transmitter authentication message generation

Input: Frame list $V = (v_0, ..., v_{11})$, an ECDSA private key d.
Output: Authentication message m_{auth}.

$m \leftarrow v_0||...||v_{11}$
$(r, s) \leftarrow Sign_d(m)$ with ECDSA P-256 Domain parameter D
$sig \leftarrow r||s$
$sig \leftarrow$ Zero_padding$(sig, 528)$
for $i \leftarrow 0$ to 11 **do** ▷ Generate transmitting authentication message.
    $temp[i] \leftarrow$ Extract_bits $(sig, i \times 44, 44)$
    $temp[i] \leftarrow i||temp[i]$
$m_{auth} \leftarrow temp[0]||...||temp[11]$
Return m_{auth}

Algorithm 5 ECDSA-only receiver signature verification

Input: Signature parts $P = (p_0, ..., p_{11})$, frame list $V = (v_0, ..., v_{11})$, public key Q.
Output: Acceptance or rejection of signature.

$sig_{pad} \leftarrow p_0||...||p_{11}$
$(r, s) \leftarrow$ (Extract_bits$(sig_{pad}, 0, 256)$, Extract_bits$(sig_{pad}, 256, 256)$)
$m \leftarrow v_0||...||v_{11}$
$Verify_Q(m, r, s)$ with ECDSA P-256 Domain parameter D

(padding)을 하는 함수로, 연접된 512 비트의 서명을 44 비트씩 전송할 수 있도록 패딩하여 사용한다. Extract_bits(*m,i,n*)는 입력 메시지 *m*에서 *i*번째의 비트부터 *n*개의 비트를 추출하는 것을 의미하며, 빅 엔디안(big-endian)을 따른다. 하나의 전자서명을 순서대로 44 비트씩 절삭(truncate)하고, 서명의 순서를 나타내는 4 비트 숫자를 44 비트의 서명 조각 앞에 연접하여 인증 메시지를 생성한다. 알고리즘의 결과로써 생성된 메시지는, 송신자가 전송할 인증 메시지가 연접된 형태를 가지며, 송신 시 48 비트, 즉, 6 바이트씩 절삭하여 전송할 수 있다.

3.1.2 ECDSA-only receiver

ECDSA-only 방법의 수신자는 12 프레임의 메시지와 이후 12 프레임의 서명을 받아서 한 번의 인증을 수행할 수 있다. 수신자는 인증의 대상이 되는 12 프레임의 메시지를 받아서 저장한 후, 이어서 전송되는 12 프레임에서 전송된 512 비트의 ECDSA P-256 서명의 값을 추출한다. 각 프레임에서 인증을 위해 사용되는 데이터는 총 48 비트이며, 48 비트 중 최초의 4 비트는 서명 조각의 순서를 나타내고, 나머지 44 비트는 서명의 일부이다. 따라서 수신자는 48 비트 중 앞의 4 비트가 나타내는 위치에 나머지 44 비트의 값을 붙여 넣는 과정을 12번 반복하여 하나의 전자서명을 완성한다. 완성된 전자서명은 528 비트로 상위 512 비트는 실제 전자 서명값이며, 하위 16 비트는 패딩으로 구성되어 있다.

수신자가 서명 전송 중간에 동기화를 하는 경우, 앞서 전송된 서명 조각을 취득할 수 없기 때문에, 수신자는 온전한 512 비트의 서명을 완성할 수 없으므로 동기화 시점에서 전송되는 서명에 대한 인증은 수행할 수 없다.

Algorithm 5는 ECDSA-only 방법의 수신자가 12개의 프레임 v_0,⋯,v_{11}과 프레임들로부터 추출된 서명 조각 p_0,⋯,p_{11}을 사용하여 ECDSA 전자 서명을 검증하는 과정을 보인다. 서명 조각을 모두 연접한 뒤, 최상위 256 비트를 *r*로, 그 다음의 256 비트를 *s*로 한다. 서명 검증 시에는 공개키 *Q*와 12개의 프레임을 모두 연접한 메시지 *m*을 이용하여 *r*과 *s*를 검증한다. 이때, 수신자는 서명 검증에 사용되는 공개키를 사전에 취득해야 하며, 공개키는 위성 신호가 아닌 별도의 방법으로 공개 배포된다. 공개키는 공개키 인증서를 사용하여 그 유효성을 검증할 수 있다.

3.2 ECDSA-TESLA

ECDSA-TESLA는 MAC 태그를 이용하여 메시지 인증을 수행하는 방법으로, 전자서명보다 적은 수의 비트를 이용하여 인증을 수행할 수 있기 때문에, 비교적 단시간 내에 인증이 가능한 방법이다. MAC 태그를 생성할 때 사용하는 키는 2.4절에서 설명한 TESLA 프로토콜에 따라 생성한 키 체인을 이용하며, 키 체인 검증은 루트 키에 대한 전자서명을 이용한다. 전자서명은 ECDSA-only 방법과 동일하게 128 비트의 안전성을 만족하는 ECDSA P-256을 사용한다. MAC 태그 생성 시에는 HMAC을 사용한다 (NIST 2008). HMAC은 송신자와 수신자가 공유하는 비밀키를 이용하여 메시지의 무결성을 확인하는 방법이다. 본 절에서 제안하는 방법은 HMAC-SHA-256을 사용하며, MAC 태그 생성 시 사용하는 키의 길이는 128 비트로 한다. HMAC-SHA-256을 128 비트의 키와 함께 사용하는 경우, NIST SP 800-107 (Dang 2012)에 따라, 안전성은 키 길이와 동일한 128 비트가 된다. 이때 생성되는 HMAC-SHA-256은 일정 길이로 만드는 해시를 수행하기 때문에 출력값의 길이는 256 비트이다. ECDSA-TESLA는 가용 비트 수가 적으므로, 방송하는 인증 메시지의 크기를 최소화하기 위해 MAC 태그의 길이는 NIST SP 800-107 (Dang 2012)에서 권장되는 최소 MAC 태그의 길이인 32 비트로 한다. 따라서 생성되는 출력 값 중 MSB부터 32 비트만을 절삭하여 MAC 태그로 사용한다.

Fig. 7은 이번 절에서 제안하는 ECDSA-TESLA 방법의 인증 메시지의 구조이다. ECDSA-TESLA 인증 메시지는 MAC 키 조각의 순서를 나타내는 MN 2 비트, MAC 태그 조각인 MP 8 비트, MAC 키 조각인 KP 32 비트, 루트 키와 루트 키에 대한 서명의 시작을 나타내는 플래그(flag) 1 비트, 루트 키 혹은 루트 키에 대한

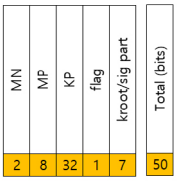

Fig. 7. ECDSA-TESLA authentication message structure.

Frame	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Data	(i-1)-th data				i-th data				(i+1)-th data				(i+2)-th data			
MAC tag	(i-2)-th tag				(i-1)-th tag				i-th tag				(i+1)-th tag			
MAC key	(i-3)-th MAC key				(i-2)-th MAC key				(i-1)-th MAC key				i-th MAC key			

Fig. 8. ECDSA-TESLA transmission pattern.

서명인 kroot/sig part 7 비트로 구성된다.

하나의 프레임에서 MAC 태그의 8 비트와 MAC 키의 32 비트를 전송하므로, 완전한 하나의 MAC 태그와 MAC 키를 전송 혹은 취득하기 위해서는 네 개의 프레임이 필요하다. 따라서 수신자가 MAC 태그와 MAC 키를 연접하기 위해서는 현재 취득한 일부의 MAC 태그와 MAC 키가 전체 중 몇 번째 부분에 해당하는지 알 필요가 있다. MN이 0인 경우는 32 비트의 MAC 태그 중 0에서 7번 비트임을 의미하며, MN이 1인 경우에는 8에서 15번, 2인 경우에는 16에서 23번, 3인 경우에는 24에서 31번 비트임을 의미한다. MAC 키의 경우, MN이 0일 때는 0에서 31번, 1일 때는 32에서 63번, 2일 때는 64에서 95번, 3일 때는 96에서 127번 비트임을 나타낸다.

Fig. 8은 ECDSA-TESLA 방법의 MAC 태그와 MAC 키 전송 패턴을 보인다. ECDSA-only 방법과 마찬가지로, 미리 인증 대상 메시지를 얻어서 MAC 태그를 생성할 수 없으므로, 4개의 프레임 동안 인증 대상 메시지를 전송하고, 그 직후 전송되는 4개의 프레임에 하나의 MAC 태그를 나누어 방송한다. MAC 키는 MAC 태그 전송 완료 후 공개해야 하므로, MAC 태그가 전송된 후 4개의 프레임 동한 해당 MAC 태그의 생성에 사용된 MAC 키를 방송한다.

수신자는 취득한 MAC 키가 올바른 지 검증하기 위해 TESLA 프로토콜에 따라 취득한 키에 대해 반복적으로 해시를 계산함으로써, 최종 해시 값이 루트 키와 동일한지 확인해야 한다. 이를 위해 수신자는 루트 키와 루트 키에 대한 서명이 필요하기 때문에 루트 키와 루트 키에 대한 서명 역시 방송되어야 한다. 전체 인증 데이터 50 비트 중 MAC 관련 데이터에 42 비트를 사용하므로, 루트 키와 서명의 전송에는 8 비트를 사용할 수 있다. 수신자가 온전한 루트 키와 서명을 취득하기 위해, 수신한 일부의 루트 키 혹은 서명의 시작이 어디인지 알아야 한다. 따라서 8 비트 중, 현재 전송하는 루트 키 혹은 서명의 일부가 첫 번째 부분임을 알리기 위하여 1 비트의 플래그(flag)를 사용하며 나머지 7 비트는 루트 키 혹은 서명의 일부를 전송한다. 현재 전송하는 부분이 시작점인 경우 플래그를 1로 설정하며, 이외의 경우에는 0으로 설정한다. 루트 키와 서명은 번갈아 전송되며, MN이 0 또는 2인 경우에는 루트 키를, MN이 1 또는 3인 경우에는 서명을 전송한다.

수신자가 MAC 키 검증을 수행할 수 있도록 전송되는 루트 키와 그에 대한 서명의 크기가 다르기 때문에 각각은 다른 전송 주기를 가진다. 루트 키는 128 비트, 서명은 512 비트이며 하나의 프레임에서 7 비트씩 전송되기 때문에, 128 비트의 루트 키를 전송하는 데에는 19 프레임이 필요하며, 서명은 74 프레임이 필요하다. 이때 루트 키와 서명은 한 번씩 번갈아 전송된다. 따라서 루트 키의 전송 주기는 38 프레임이며, 서명의 전송 주기는 148 프레임이다. 루트 키 혹은 서명을 모두 전송한 이후에도 Galileo의 OSNMA (European GNSS Agency 2023)와 마찬가지로 동일한 루트 키 혹은 서명을 반복하여 전송함으로써, 루트 키와 서명이 전송되는 중간 시점에 인증 서비스에 동기화한 사용자도 루트 키와 서명을 취득할 수 있도록 한다.

Algorithm 6 TESLA key chain generation

Input: Total key number n_t.
Output: Key chain seed s_k, key chain $K = (k_0, ..., k_{n_t-1})$.
1: Select $\sigma \in_R [0, 2^{128} - 1]$
2: $s_k \leftarrow \sigma$
3: **for** $i \leftarrow 0$ to $n_t - 1$ **do** ▷ Generate key chain
4: $h \leftarrow \boldsymbol{H}(\sigma)$
5: $\sigma \leftarrow \text{trunc}(h, 128)$
6: $k_{n_t-i-1} \leftarrow \sigma$
7: **Return** s_k, K

TESLA 키 체인은 하루를 주기로 갱신되며, 4개의 프레임 동안 동일한 키를 사용하므로, 2분 마다 하나씩 사용한다. 따라서 하루동안 사용하는 키 체인의 길이는 720이 된다. TESLA 키 체인 갱신 시에도 사용자가 원활한 인증 서비스를 제공받을 수 있도록 새로운 키 체인의 루트 키와 그 서명이 미리 전송되어야 한다. 단, 본 논문에서 제시하는 방법은 예비 필드를 사용하기 때문에 가용 비트수가 제약적인 관계로, 키 체인 및 서명의 갱신을 알리는 플래그를 사용할 수 없으므로, 키 체인 및 서명 갱신의 74분 이전부터는 갱신되는 루트 키와 그에 대한 서명을 방송하도록 한다. 여기서 74분은 수신자가 완전한 서명을 취득하기 위해서 필요한 148 프레임 수신에 소요되는 시간이다.

3.2.1 ECDSA-TESLA transmitter

ECDSA-TESLA의 송신자는 먼저 키 체인 생성 후 루트 키에 대한 서명을 생성해야 한다. Algorithm 6은 ECDSA-TESLA 송신자의 키 체인 생성 과정을 보인다. 먼저, TESLA 키 체인 생성을 위하여 키 길이와 같은 128 비트의 난수 σ를 생성하여 s_k에 할당한다. 이후, line 2에서, 생성한 난수 σ를 입력으로 SHA-256 해시를 출력한다. 본 방법에서는 128 비트의 MAC 키를 사용하므로, SHA-256의 출력 값 중 최상위 128 비트를 절삭하여 키 체인에 저장한다. 생성한 128 비트의 MAC 키에 대한 해시를 생성하고 그것을 128 비트로 절삭하는 것을 반복하며 키 체인에 저장한다. trunc(m,n)는 메시지 m의 최상위 n 비트를 반환하는 함수로, Extract_bits(m, 0, n)과 동일한 동작을 한다. 이때, 저장되는 MAC 키는 키 체인에 역순으로 저장되어, 마지막에 생성된 키는 k_0에 저장된다. 알고리즘에서는 생성된 키 체인 K와 키 체인 생성에 사용된 시드 s_k를 반환한다.

Algorithm 7은 Algorithm 6에 따라 생성된 키 체인에 대한 검증을 가능하도록 하는 루트 키에 대한 서명 생성 과정을 보인다. 이 알고리즘에서는 개인키 d를 이용하여, 루트 키 k_0에 대한 ECDSA P-256 서명을 생성하여 반환한다.

Algorithm 7 Root key signature generation

Input: ECDSA private key d, root key k_0.
Output: root key signature (r_r, s_r).
1: $(r_r, s_r) \leftarrow Sign_d(k_0)$ with ECDSA P-256 Domain parameter D
2: **Return** (r_r, s_r)

Algorithm 8 MAC tag generation

Input: Frame list $V = (v_0, ..., v_3)$, key chain $K = (k_0, ..., k_{n-1})$, key number $n_k \in [0, n-1]$.
Output: MAC tag g.
1: $msg \leftarrow v_0||...||v_3$
2: $temp \leftarrow$ HMAC(msg, k_{n_k}) with SHA-256
3: $g \leftarrow$ trunc($temp$, 32)
4: **Return** g

Algorithm 9 Root key and its signature part generation

Input: Root key k_0, root key signature (r_r, s_r), counter c.
Output: kroot/sig part o_c.
1: **if** c is even **then**
2: $i \leftarrow c$ mod 38
3: $temp \leftarrow$ Extract_bits$(k_0, 7 * i, 7)$
4: $o_c \leftarrow temp \gg 1$
5: **else**
6: $i \leftarrow c$ mod 148
7: $temp \leftarrow$ Extract_bits$(r_r||s_r, 7 * i, 7)$
8: $o_c \leftarrow temp \gg 1$
9: **if** i is 0 **then**
10: $o_c \leftarrow o_c \mid$ 0x80
11: **Return** o_c

Algorithm 10 ECDSA-TESLA transmitter authentication message generation

Input: Kroot/sig part $O = (o_0, o_1, o_2, o_3)$, MAC tag g, MAC key k_m.
Output: Authentication messages $M_{auth} = (m_{auth\_0}, ..., m_{auth\_3})$.
1: **for** $i \leftarrow 0$ to 3 **do**
2: $m_{auth_i} \leftarrow i||$Extract_bits$(g, i \times 8, 8)||$Extract_bits$(k_m, i \times 32, 32)||o_i$
3: **Return** M_{auth}

Algorithm 8은 인증 대상인 4개의 프레임에서 전송된 메시지에 대한 MAC 태그 생성 과정을 기술한다. 인증 대상이 되는 메시지는 V로, 해당 메시지에 대한 MAC 태그를 생성한다. MAC 태그 생성 시에는 HMAC-SHA-256을 사용하며, 이때 사용하는 키는 Algorithm 6에 의해 생성된 키 체인의 키를 0번부터 n_k-1번까지 순서대로 사용한다. 인증 대상인 4개의 프레임을 연접한 msg와 키를 입력으로 HMAC-SHA-256을 수행하여 256 비트의 값을 얻는다. HMAC(msg,k_{n_k})은 키 k_{n_k}를 사용하여 메시지 msg에 대한 MAC 태그를 생성하는 것을 의미한다. 본 방법에서 인증에 사용하는 MAC 태그의 크기는 32 비트이므로, 생성된 HMAC-SHA-256의 값에서 상위 32 비트를 절삭하여 MAC 태그로 사용한다. 이에, 알고리즘에서는 생성된 32 비트의 MAC 태그를 반환한다.

하루 동안 루트 키와 서명은 갱신되지 않으므로, 루트 키와 서명은 전송 형태에 맞추어 1 비트의 플래그와 7 비트의 루트 키 혹은 서명의 조각을 미리 생성할 수 있다. 플래그와 결합한 루트 키 또는 서명 조각 생성은 Algorithm 9를 따른다.

MN이 짝수인 경우에는 루트 키의 조각을, 홀수인 경우에는 루트 키에 대한 서명 조각을 전송하기 때문에, 현재 몇 번째의 메시지를 전송하는지에 따라 전송해야 하는 값이 바뀐다. 따라서 현재 몇 번째 인증 메시지를 전송하는지 나타내는 카운터를 입력받아서 그것이 짝수인 경우에는 루트 키 조각을, 홀수인 경우에는 서명 조각을 생성한다. 이때, 루트 키의 전송 주기는 38 프레임이므로, 38 프레임마다 해당하는 순서의 7 비트의 루트 키 조각을 전송할 수 있도록, line 2와 같이 카운터를 38로 나눈 나머지를 i에 저장한다. 이후, 루트 키에서 i번째의 7 비트를 임시 변수 $temp$에 저장한 뒤, 1 비트 right shift 연산으로 플래그를 설정할 비트를 확보한다. 조각의 순서를 나타내는 i가 0인 경우 루트 키의 첫 번째 조각임을 뜻하기 때문에, 최상위 비트를 1로 설정하여 반환한다. 루트 키에 대한 서명 조각 생성은 카운터가 홀수인 경우 진행되며, 서명 조각 생성 방식은 루트 키 조각 생성 방식과 동일하다. 단, 서명의 전송 주기는 148이므로, 카운터를 148로 나눈 나머지를 변수 i에 저장하여 사용한다.

Algorithm 10은 ECDSA-TESLA 송신자의 인증 메시지 생성 과정을 기술한다. 하나의 MAC 태그와 MAC 키는 총 네 개의 프레임으로 나누어 전송되므로, MAC 태그와 MAC 키를 하나씩 입력 받아 네 개로 나눈다. 각각의 메시지는 순서에 맞추어 8 비트의 MAC 태그와 32 비트의 MAC 키 값을 포함한다. 이때 전송되

는 MAC 키는, TESLA 프로토콜에 따라, 바로 이전의 4개 프레임에서 전송된 MAC 태그 생성에 사용된 MAC 키를 전송한다. 각각의 인증 메시지는 루트 키 또는 서명 조각을 가지므로, Algorithm 9에 따라 사전에 생성된 루트 키 또는 서명 조각을 순서에 맞추어 입력하여 인증 메시지를 생성한다.

3.2.2 ECDSA-TESLA receiver

ECDSA-TESLA 방법의 수신자는 총 12개의 프레임을 수신한 뒤 최초 4개의 프레임에서 전송된 메시지에 대한 인증을, 인증 대상 메시지 이후 전송되는 4개의 프레임에서 전송되는 MAC 태그와 마지막 4개의 프레임에서 전송되는 MAC 키를 이용하여 수행한다. 수신자가 MAC 키를 사용하기 위해서는 MAC 키에 대한 인증이 선행되어야 하기 때문에, 수신자는 사전에 인증에 사용할 키 체인의 루트 키와 그것에 대한 전자서명을 취득해야 한다. 루트 키는 전송 주기인 38 프레임의 값 중 짝수 번째인 19개 프레임들의 값을 조합하여 사용한다. 마찬가지로, 전자서명은 전송 주기인 148 프레임 중 전자서명에 해당하는 홀수 번째 74개 프레임들의 값을 조합하여 사용한다. 루트 키와 전자서명은 플래그를 통해 시작 지점을 알리므로, 루트 키와 서명 연접 시에는 플래그가 1인 부분을 가장 앞으로 오도록 하여, 수신자가 인증에 참여한 순간부터 최단시간에 루트 키와 그에 대한 전자서명을 얻을 수 있도록 한다.

루트 키와 루트 키에 대한 서명의 획득 과정은 각각 Algorithm 11과 Algorithm 12에 기술되어 있다. 인증 메시지에서 시작을 알리는 플래그와 그 뒤 7 비트(루트 키 또는 서명의 일부)를 포함하여 각 프레임별 총 8 비트를 저장해 두고, 모든 부분들이 수신되면 이들을 연접하여 루트 키 또는 서명을 복구한다. 루트 키에 대한 값은 $Y=(y_0,\cdots,y_{18})$ (각 y_i는 8 비트)에 저장하며, 서명에 대한 값은 $P=(p_0,\cdots,p_{73})$ (각 p_i는 8 비트)에 저장한다.

루트 키를 연접하는 알고리즘(Algorithm 11)의 입력은 루트 키 조각, 루트 키의 시작을 알리는 플래그가 있는 루트 키 조각의 인덱스 번호이다. 인덱스 번호에 해당하는 루트 키 조각을 첫 번째로 오도록 하여 모든 조각을 루트 키를 조합할 순서에 맞추어 임시 변수에 저장한다. 이후, 플래그를 나타내는 최상위 1 비트를 제거하고 모든 루트 키 값을 연접하기 위하여 line 3에서 7을 수행한다. 이 부분은 임시 변수 t_{i-1}에 t_i의 최상위 비트부터 적정 수의 비트들을 가져와서 루트 키 값의 일부를 8 비트 단위로 저장하도록 한다. 모든 임시 변수가 8 비트의 값을 가지면 그것들을 연접하여 하나의 루트 키를 만들어 그것을 반환한다. 이때, 7 비트의 루트 키 조각을 가지고 있는 각각의 임시 변수에 바로 다음의 비트를 가져와 8 비트로 만들었으므로, t_7, t_{15} 에 있던 7 비트의 값은 각각 t_6, t_{14} 에 옮겨져 저장된다. 또한 t_{18}에 있던 값은 t_{17}에 옮겨져 저장된다. 따라서 line 8에서는 루트 키 값을 갖지 않는 t_6, t_{14}, t_{18} 은 제외하고 연접하여 128 비트 루트 키를 생성한다.

서명을 연접하는 알고리즘(Algorithm 12)은 루트 키 연접과 유사한 방식으로 진행된다. 입력 받은 인덱스의 서명 조각이 가장 앞으로 오도록 하여 서명 조각의 순서를 정렬한 뒤, 각각이 8 비트의 서명 값을 가지도록 그 다음의 변수에서 필요한 비트를 가져와 채운다. 이후, 8의 배수가 되는 순서에 있는 임시 변수는 값을 가지지 않기 때문에 그것들을 제외하고 연접하여 서명을 만든다. 서명 조각을 연접하여 얻은 결과는 서명의 r_r과 s_r가 연접된 형태로, 그것의 상위 256 비트는 r_r의 값을, 하위 256 비트는 s_r의 값을 가진다.

Algorithm 11 ECDSA-TESLA receiver root key concatenation

Input: Root key parts $Y = (y_0, ..., y_{18})$, root key start index x_r.
Output: Root key k_{root}.

1: **for** $i \leftarrow 0$ to 18 **do**
2: $\quad t_i \leftarrow y_{x_r+i \bmod 19}$
3: **for** $i \leftarrow 0$ to 1 **do**
4: $\quad$ **for** $j \leftarrow 0$ to 6 **do**
5: $\quad\quad t_{j+i\times 8} \leftarrow (t_{j+i\times 8} \ll (j+1)) | (t_{j+1+i\times 8} \gg (6-j))$
6: **for** $j \leftarrow 0$ to 1 **do**
7: $\quad t_{j+2\times 8} \leftarrow (t_{j+2\times 8} \ll (j+1)) | (t_{j+1+2\times 8} \gg (6-j))$
8: $k_{root} \leftarrow t_0 || ... || t_6 || t_8 || ... || t_{13} || t_{15} || t_{16} || t_{17}$
9: **Return** k_{root}

Algorithm 12 ECDSA-TESLA receiver signature concatenation

Input: Signature parts $P = (p_0, ..., p_{73})$, signature start index x_s.
Output: Root key signature (r_r, s_r).

1: **for** $i \leftarrow 0$ to 73 **do**
2: $\quad t_i \leftarrow y_{x_s+i \bmod 74}$
3: **for** $i \leftarrow 0$ to 8 **do**
4: $\quad$ **for** $j \leftarrow 0$ to 6 **do**
5: $\quad\quad t_{j+i\times 7} \leftarrow t_{j+i\times 8} \ll (j+1) | (t_{j+1+i\times 8} \gg (6-j))$
6: $t_{72} \leftarrow t_{72} \ll 1 | (t_{73} \gg 6)$
7: **for** $i \leftarrow 0$ to 72 **do**
8: $\quad$ **if** $(i+1) \bmod 8 \neq 0$ **then**
9: $\quad\quad c \leftarrow c || t_i$
10: $(r_r, s_r) \leftarrow c$
11: **Return** (r_r, s_r)

Algorithm 13은 ECDSA-TESLA 방법의 수신자가 인증을 수행하는 과정을 보인다. Algorithm 13의 입력은 연속된 12개의 프레임과 루트 키, 루트 키에 대한 서명과 서명 검증에 사용될 공개 키이다. 12개의 프레임 중 최초 4개의 프레임에는 인증 대상 메시지가, 그 다음의 4개의 프레임에는 MAC 태그가, 마지막 4개의 프레임에는 MAC 키가 포함된다. Line 1과 2에서는 루트 키에 대한 서명을 검증하고, 검증 실패 시 루트 키를 신뢰할 수 없으므로, 알고리즘을 종료하고, 검증 성공 시 다음 단계를 진행한다. 루트 키 검증 이후에는 line 3에서 10과 같이 MAC 키를 확인한다. 마지막 4개의 프레임(8에서 11번 프레임)에 32 비트씩 저장된 MAC 키 조각을 연접하여 128 비트의 MAC 키를 만들고, 반복적으로 해시를 수행하여 그것의 유효성을 검증한다. MAC 키는 TESLA 프로토콜에 따라 생성되었기 때문에, 본 방법에서 사용되는 키 체인의 길이는 720으로, MAC 키에 대해 최대 720번의 해시를 수행하면 루트 키에 도달한다. 따라서 앞서 생성한 MAC 키에 대한 해시를 반복적으로 생성하면서 루트 키와 비교하고, 루트 키와 일치하는 경우 MAC 키 검증에 성공한다. MAC 키의 해시를 720번 동안 수행하며 생성된 모든 값이 루트 키와 일치하지 않는 경우, MAC 키 인증에 실패하여 MAC 키에 대한 신뢰를 확보할 수 없기에 알고리즘을 종료한다. MAC 키 인증에 성공한 경우, line 11와 12을 수행한다. MAC 태그 조각이 저장된 4에서 7번째 프레임에 8 비트씩 저장된 MAC 태그 조각을 추출하고 연접하여 32 비트의 MAC

Algorithm 13 ECDSA-TESLA receiver MAC tag verification

Input: Frame list $V = (v_0, ..., v_{11})$, root key k_{root}, root key signature (r_r, s_r), public key Q.
Output: Acceptance or rejection of MAC tag.

1: $Verify_Q(k_{root}, r_r, s_r)$ with ECDSA P-256 Domain parameter D
2: **If** verification fails, **return** ("Reject root key signature"); **otherwise**, continue;
3: **for** $i \leftarrow 8$ to 11 **do**
4: $key \leftarrow key$ || Extract_bits$(v_i, 1695 \times 29 + 1645 + 10, 32)$
5: $digest \leftarrow key$
6: **for** $i \leftarrow 0$ to 719 **do**
7: **if** $digest$ is k_{root} **then** break;
8: $digest \leftarrow \boldsymbol{H}(key)$
9: $digest \leftarrow$ trunc$(digest, 128)$
10: **If** $i = 720$ **return** ("invalid key"); **otherwise**, continue;
11: **for** $i \leftarrow 4$ to 7 **do**
12: $g \leftarrow g$||Extract_bits$(v_i, 1695 \times 29 + 1645 + 2, 8)$
13: $m_{input} \leftarrow v_0||...||v_3$
14: $g' \leftarrow$ HMAC(m_{input}, key) with SHA-256
15: $g' \leftarrow$ trunc$(g', 32)$
16: **If** g' is g **return** ("Accept the MAC tag"); **else**, **return** ("Reject the MAC tag")

태그를 만든다. 이후, line 13과 같이 인증 대상 메시지가 저장된 0에서 3번의 프레임의 메시지를 연접하고, 연접한 메시지와 앞서 생성한 MAC 키로 HMAC-SHA-256을 사용하여 수신자가 MAC 태그를 생성한다. 수신자가 생성한 MAC 태그 g'와 line 12에서 얻은 연접한 MAC 태그 g의 값을 비교하고, 그 값이 같은 경우에는 인증 성공, 그렇지 않은 경우에는 인증 실패를 의미한다.

4. PERFORMANCE ANALYSIS

이번 장에서는 제안하는 두 가지 방법의 성능을 평가하고, 이를 분석한다. 성능의 평가를 위해 Fernández-Hernández et al. (2014)에 의해 제안된 Time Between Authentication (TBA)과 Time To First Authenticated Fix (TTFAF)를 사용한다. TBA는 하나의 위성이 전송하는 연속된 두 인증 메시지 사이의 시간 간격을 의미하며, 첫 번째 인증 메시지 전송 시작부터 다음 인증 메시지 전송 시작 시간의 차를 이용하여 구할 수 있다. $\overline{TBA}$는 TBA의 평균을 의미하며, Eq. (2)와 같이 계산할 수 있다. Eq. (2)의 *AER*은 Authentication Error Rate로, Eq. (3)과 같이 계산할 수 있으며, 여기서 *BER*은 Bit Error Rate로 비트 전송 오류율을, *NNA*는 인증 대상 메시지의 비트 수와 해당 메시지에 대한 인증 비트 수의 합을 의미한다.

$$\overline{TBA} = \frac{TBA}{1 - AER} \tag{2}$$

$$AER = 1 - (1 - BER)^{NNA} \tag{3}$$

TTFAF는 수신자가 신뢰 가능한 하나의 인증 메시지를 얻는 데 걸리는 시간을 의미한다. TTFAF의 평균인 $\overline{TTFAF}$는 Eq. (4)와 같이 계산한다.

$$\overline{TTFAF} = \frac{TBA}{2} + \overline{TBA} \tag{4}$$

ECDSA-only 방법은 인증 메시지를 12 프레임마다 하나씩 전송한다. 따라서 TBA는 12 프레임을 수신하는 시간이 되며, 한 프레임의 전송에 30초가 소요되므로, 비트 오류율을 0으로 가정한 경우, ECDSA-only 방법의 $\overline{TBA}$는 360초, $\overline{TTFAF}$는 540초이다.

Table 4. ECDSA-only $\overline{TBA}$ and $\overline{TTFAF}$ according to *BER*.

BER	*AER*	$\overline{TBA}$	$\overline{TTFAF}$
0	0	360	540
10^{-8}	0.0061	362.209	542.209
10^{-7}	0.0592	382.653	562.653
10^{-6}	0.4570	662.983	842.983

*BER*이 10^{-8}, 10^{-7}, 10^{-6}인 경우의 *AER*과 $\overline{TBA}$, $\overline{TTFAF}$는 Table 4와 같다. Fig. 1에 따르면 하나의 데이터 파트는 1695 비트이며, 하나의 프레임은 30개의 데이터 파트를 가진다. 따라서, 12 프레임의 데이터를 한 번에 인증하는 ECDSA-only 방법의 인증 대상 메시지의 비트 수는 610200이며, 인증 비트 수는 전자서명의 크기인 512이므로, *NNA*는 610712이다. *BER*이 10^{-8}, 10^{-7}, 10^{-6}일 때 *AER*은 각각 0.0061, 0.0592, 0.4570이다. *AER*을 이용하여 계산한 $\overline{TBA}$는 362.209, 382.653, 662.983이며, 이에 따른 $\overline{TTFAF}$는 542.209, 562.653, 842.982이다. *BER*이 0인 경우에는 $\overline{TBA}$와 $\overline{TTFAF}$가 각각 360, 540으로, *BER*이 증가하는 경우에는 $\overline{TBA}$와 $\overline{TTFAF}$가 모두 증가하며, *BER*이 10^{-8}로 오류율이 낮은 경우에는 오류율이 0인 $\overline{TBA}$와 $\overline{TTFAF}$에 근접한 것을 알 수 있다.

ECDSA-TESLA 방법은 4 프레임마다 하나의 인증 메시지를 전송하므로, 비트 오류율을 고려하지 않았을 때의 $\overline{TBA}$는 120초, $\overline{TTFAF}$는 180초이다. *BER*이 10^{-8}, 10^{-7}, 10^{-6}인 경우 ECDSA-TESLA 방법의 *AER*과 $\overline{TBA}$, $\overline{TTFAF}$는 Table 5와 같다. ECDSA-TESLA는 한 번의 인증에 4개의 프레임 동안 전송된 메시지를 사용하므로, 인증 대상 메시지의 비트 수는 203400 비트이며, 인증 비트 수는 MAC 태그와 MAC 키의 비트 수인 160 비트이다. 따라서 *AER* 계산에 사용되는 *NNA*는 203560이다. Eq. (3)에 따라 계산한 *AER*은 *BER*이 10^{-8}, 10^{-7}, 10^{-6}일 때 각각 0.0020, 0.0202, 0.1842이다. *AER*을 이용하여 계산한 $\overline{TBA}$는 120.240, 122.474, 147.095이며, 이에 따른 $\overline{TTFAF}$는 180.240, 182.474, 207.095이다. *BER*이 0인 경우에는 $\overline{TBA}$와 $\overline{TTFAF}$가 각각 120, 180으로, *BER*이 증가하는 경우에는 $\overline{TBA}$와 $\overline{TTFAF}$가 모두 증가하며, *BER*이 10^{-7}과 10^{-8}인 경우에는 오류율이 0인 $\overline{TBA}$와 $\overline{TTFAF}$에 근접한 것을 알 수 있다.

Table 5. ECDSA-TESLA $\overline{TBA}$ and $\overline{TTFAF}$ according to *BER*.

BER	*AER*	$\overline{TBA}$	$\overline{TTFAF}$
0	0	120	180
10^{-8}	0.0020	120.240	180.240
10^{-7}	0.0202	122.474	182.474
10^{-6}	0.1842	147.095	207.095

Table 6. ECDSA-TESLA $\overline{TBS}$ and $\overline{TTFAF}$ according to *BER*.

BER	*AER*	$\overline{TBS}$	$\overline{TTFAF}$
0	0	4440	6660
10^{-8}	0.0020	4448.898	6668.898
10^{-7}	0.0202	4531.537	6751.537
10^{-6}	0.1847	5445.848	7665.848

Table 7. $\overline{TBA}$ or $\overline{TBS}$ according to *BER*.

	0	10^{-8}	10^{-7}	10^{-6}
Chimera (Jeon et al. 2022)	180	180.0150	180.1504	181.5093
OSNMA (Jeon et al. 2022)	10	10.0001	10.0008	10.0076
OSNMA (cold) (Jeon et al. 2022)	240	240.0036	240.0360	240.3603
QZSS LNAV (Jeon et al. 2023)	240	240.0034	240.0339	240.3391
QZSS CNAV (Jeon et al. 2023)	240	240.0034	240.0339	240.3391
QZSS CNAV2 (Jeon et al. 2023)	240	240.0027	240.0267	240.2670
Proposed method (ECDSA-only)	360	362.209	382.653	662.983
Proposed method (ECDSA-TESLA)	120	120.24	122.474	147.095
Proposed method (ECDSA-TESLA (cold))	4440	4448.898	5445.848	5445.848

Table 8. $\overline{TTFAF}$ according to *BER*.

	0	10^{-8}	10^{-7}	10^{-6}
Chimera (Jeon et al. 2022)	270	270.0150	270.1504	271.5093
OSNMA (Jeon et al. 2022)	15	15.0001	15.0008	15.0076
OSNMA (cold) (Jeon et al. 2022)	360	360.0036	360.0360	360.3603
QZSS LNAV (Jeon et al. 2023)	360	360.0034	360.0339	360.3391
QZSS CNAV (Jeon et al. 2023)	360	360.0034	360.0339	360.3391
QZSS CNAV2 (Jeon et al. 2023)	360	360.0027	360.0267	360.2670
Proposed method (ECDSA-only)	540	542.209	562.653	842.983
Proposed method (ECDSA-TESLA)	180	180.24	182.474	207.095
Proposed method (ECDSA-TESLA (cold))	6660	6668.898	6751.537	7665.848

ECDSA-TESLA 방법의 경우, 수신자가 루트 키에 대한 전자서명을 모두 취득하여 루트 키에 대한 검증을 한 후 MAC과 MAC 키를 이용하여 메시지 인증을 수행할 수 있다. 수신자가 전자서명을 기다려야 하는 cold start 상황의 경우에는 인증을 위한 시간이 추가로 필요하게 된다. 이때, 전자서명 검증을 위한 시간에 대한 지표는 TBA가 아닌 Time Between Signatures (TBS)를 사용한다 (Gaybullaev et al. 2021). $\overline{TBS}$와 $\overline{TBS}$의 $\overline{TTFAF}$는 Eqs. (5, 6)을 통해 계산된다.

$$\overline{TBS} = \frac{TBS}{1 - AER} \tag{5}$$

$$\overline{TTFAF} = \frac{TBS}{2} + \overline{TBS} \tag{6}$$

ECDSA-TESLA 방법에서 전자서명은 148 프레임마다 전송되며, 하나의 전자서명을 얻는 데에 4440초가 필요하다. 비트 에러를 고려하지 않은 경우의 *TBS*는 4440이며, *TTFAF*는 6660이다. 비트 에러율에 따른 *AER*과 $\overline{TBS}$, $\overline{TTFAF}$는 Table 6과 같다.

ECDSA-TESLA cold start 상황에서는 인증 대상이 되는 메시지의 길이는 4개의 프레임동안 전송되는 203400 비트의 메시지와 전자서명을 통해 검증해야 하는 루트 키의 비트 수인 128 비트의 합인 203528이다. 인증 메시지의 비트 수는 MAC 태그와 MAC 키의 160 비트와 전자서명의 512 비트의 합인 672 비트이다. 따라서 ECDSA-TESLA 방법에서 cold start의 *NNA* 값은 204200이다. 이를 이용하여 구한 *BER*에 따른 *AER*은 10^{-8}일 때 0.0020, 10^{-7}일 때 0.0202, 10^{-6}일 때 0.1847이다. *AER*에 따른 $\overline{TBA}$와 $\overline{TTFAF}$는 *AER*이 0.0020일 때 각각 4448.898과 6668.898, *AER*이 0.2020일 때 4531.537과 6751.537, *AER*이 0.1847일 때 5445.848과 7665.848이다. 서명의 경우, *NNA*의 값이 크기 때문에 비트 오류율이 증가하는 경우, $\overline{TBA}$와 $\overline{TTFAF}$의 변화율이 높은 것을 확인할 수 있다.

Table 7은 *BER*에 따른 $\overline{TBA}$ 또는 $\overline{TBS}$를 GPS의 Chimera, Galileo의 OSNMA, QZSS의 QZNMA의 성능과 비교한다. Table 8은 *BER*에 따른 $\overline{TTFAF}$를 Chimera, Galileo의 OSNMA, QZSS의 QZNMA의 성능과 비교한다. Chimera, OSNMA, OSNMA (cold)의 *BER*에 따른 $\overline{TBA}$와 $\overline{TTFAF}$는 Jeon et al. (2022)에서 제시된 값을 사용하였으며, QZSS LNAV, CNAV, CANV2의 $\overline{TBA}$와 $\overline{TTFAF}$는 Jeon et al. (2023)에서 제시된 성능을 참조하였다.

Tables 7과 8을 통해 본 논문에서 제안한 방법인 ECDSA-only와 ECDSA-TESLA는 *BER*이 10^{-6}일 때 $\overline{TBA}$와 $\overline{TTFAF}$가 급격하게 변하는 것을 확인할 수 있다. Chimera, OSNMA, OSNMA (cold), QZSS LNAV, CNAV, CNAV2의 *NNA*는 각각 8350, 756, 1500, 1412, 1412, 1112인 반면, 제안 방법의 ECDSA-only와 ECDSA-TESLA, ECDSA-TESLA (cold)는 *NNA*가 각각 610712, 203560, 204200이다. 본 논문에서 제안하는 방법은 인증을 위한 가용 비트수가 제약적인 상황을 가정하므로, 여러 프레임에 걸쳐 인증 메시지를 전송하기 때문에 많은 양의 메시지를 한 번에 인증한다. 따라서 비트 오류율에 따른 $\overline{TBA}$와 $\overline{TTFAF}$가 다른 위성 항법 시스템의 인증 프로토콜에 비해 급격한 변화를 보인다.

전자서명을 이용한 인증 프로토콜은 Chimera, OSNMA (cold), QZSS LNAV, QZSS CNAV, QZSS CNAV2와 ECDSA-only가 있다. Chimera는 ECDSA P-224를 사용하며, 이외의 프로토콜

은 ECDSA P-256을 사용한다. 본 논문에서 제안하는 방법 중 ECDSA P-256을 사용하는 ECDSA-only 방법은 비트 오류율이 없는 경우와 10^{-8}, 10^{-7}일 때, 동일한 커브의 전자서명을 사용하는 프로토콜의 약 1.5배의 $\overline{TBA}$와 $\overline{TTFAF}$를 보인다.

MAC 태그를 사용하는 ECDSA-TESLA 방법의 $\overline{TBA}$와 $\overline{TTFAF}$는 동일하게 MAC 태그를 사용하는 OSNMA의 $\overline{TBA}$와 $\overline{TTFAF}$의 약 12배의 값을 가진다.

분석 결과에 따르면, 전자서명을 사용하는 OSNMA (cold), QZSS의 TBA와 TTFAF에 비해 TESLA MAC 태그를 사용하는 OSNMA의 TBA와 TTFAF는 24배로 감소하는 반면, 제안 방법에서는 ECDSA-only의 TBA와 TTFAF에 비해 ECDSA-TESLA 방법의 TBA와 TTFAF는 단지 3배만 감소한다. TESLA 기반 방법의 장점은 MAC 태그를 사용하는 경우 인증에 사용하는 비트 수가 적어져 빠른 인증이 가능하다는 것이지만, 본 논문에서 가정하는 것과 같이 인증에 사용하는 비트 수가 극히 제한적인 상황에서는 TESLA 키 체인과 MAC 태그를 사용하더라도 전자서명만 사용하는 경우와 인증 시간의 차이가 극명히 드러나지 않음을 알 수 있다.

위와 같은 사실은 수신자 측의 메시지 수신 시작으로부터 인증 종료까지의 시간을 분석함으로써 더 분명하게 확인할 수 있다. ECDSA-only 방법의 경우, Fig. 6과 같이 1번에서 12번 프레임의 메시지는 13번에서 24번 프레임의 인증 메시지를 이용하여 인증을 수행하기 때문에 1번 프레임은 이후의 23개의 프레임을 수신해야 인증이 가능하며, 12번 프레임은 이후의 12개의 프레임을 수신해야 인증할 수 있다. 따라서 ECDSA-only 방법에서 메시지를 취득한 시점부터 해당 메시지를 인증하는데 걸리는 시간은 최소 6분에서 최대 12분이다. 한편, ECDSA-TESLA 방법은 Fig. 8과 같이, 1번에서 4번 프레임에서 전송된 메시지는 5번에서 8번 프레임의 MAC tag와 9번에서 12번 프레임의 MAC 키를 이용하여 인증을 수행한다. 따라서, 1번 프레임은 이후의 11개의 프레임을 수신해야 인증이 가능하며, 4번 프레임은 이후의 8개의 프레임을 수신해야 인증이 가능하다. 따라서 ECDSA-TESLA의 메시지 취득 시점부터 해당 메시지를 인증하는 데에 걸리는 시간은 최소 4분에서 최대 6분이다. 요약하면, ECDSA-only 방법의 메시지 취득부터 메시지 인증까지의 시간은 6분에서 12분, ECDSA-TESLA는 4분에서 6분으로, ECDSA-TESLA 방법에 의한 인증 시간 감소가 1.5배에서 2배에 불과함을 알 수 있다.

마지막으로, 본 논문에서 제안하는 ECDSA-only 방법과 ECDSA-TESLA 방법의 주요 연산의 성능 측정 결과를 설명한다. Table 9는 제안하는 방법들의 구현 환경이며, 공개 라이브러리인 mbedtls 기반으로, Intel(R) Core(TM) i7-13700F @ 2.10 GHz CPU, Ubuntu 22.06 x64 OS, 32GB RAM이 장착된 데스크탑 PC 환경에서 구현 및 성능을 측정하였다. 성능 측정의 대상이 되는 주요 연산은 ECDSA-only 방법에서는 송신자의 서명 생성과 전송 형태의 인증 메시지 생성, 수신자의 서명 조각 연접과 서명 검증이며, ECDSA-TESLA 방법에서는 송신자의 키 체인 생성, 루트 키 서명 생성, MAC 생성과 수신자의 서명 조각 연접, MAC 연접, MAC 키 조각 연접, MAC 검증, MAC 키 검증, 서명 검증이며, 각 연산을 1000번 실행하여 그 평균을 연산 시간으로 사용하였다.

ECDSA-only 방법과 ECDSA-TESLA 방법의 주요 연산 시간은 Tables 10, 11과 같다. ECDSA-only 방법의 서명 생성은 2.2393 ms, 서명 검증은 4.111 ms가 소요되며, ECDSA-TESLA 방법의 서명 생성은 1.955 ms, 서명 검증은 3.699 ms로, 서명 생성 및 검증에는 ms 단위의 시간이 필요하다. Table 10에 따르면, ECDSA-only 방법의 서명 생성 및 검증을 제외한 연산들은 1 ms 이하의 시간이 소모되어, 연산 시간이 무시할 수 있는 정도의 수준임을 알 수 있다.

Table 11에 따르면, ECDSA-TESLA 송신자의 TESLA 키 체인 생성은 하루 동안 사용하기 위한 720개의 키를 생성하였으며, 0.222 ms가 소요된다. MAC 태그 생성의 경우, 4개의 프레임에서 전송되는 인증 대상 메시지를 연접하여 하나의 MAC 태그를 생성하였으며, 0.106 ms가 소요된다. ECDSA-TESLA 수신자의 루트 키, 서명, MAC 키, MAC 태그를 각각 연접하는 연산은 1 ms 이하의 시간이 소모된다. 하나의 MAC 태그의 유효성을 검증하는 데에는 0.109 ms의 시간이 소요되며, MAC 키 검증에는 최대 0.207 ms가 소요된다. MAC 키 검증의 경우, TESLA 키 체인에서 몇 번째의 키를 검증하는지에 따라 해시 연산 수행 횟수가 달라지므로, 본 실험에서는 최대 횟수의 해시를 수행하는 720번째의 키 검증의 성능을 측정하였다. 최대 횟수인 720번의 해시 연산을 수행하도록 하였음에도, MAC 키 검증에는 0.207 ms의 시간이 소요되며, 이는 매 인증 시 4.111 ms가 소요되는 ECDSA-only 방법의 서명 검증과 비교하여 짧은 수행 시간을 보이는 것을 확인할 수 있다.

Table 9. Implementation environment.

	ECDSA-only	ECDSA-TESLA
Open library	Mbedtls	
CPU	Intel(R) Core (TM) i7-13700F @ 2.10 GHz	
OS	Ubuntu 22.06 x64	
MEMORY	32 GB	

Table 10. Performance evaluation of ECDSA-only method.

	Operation	Time
Transmitter	ECDSA signature generation	2.293 ms
	authentication message generation	11.992 μs
Receiver	ECDSA signature concatenation	1.598 μs
	ECDSA signature verification	4.111 ms

Table 11. Performance evaluation of ECDSA-TESLA method.

	Operation	Time
Transmitter	TESLA key chain generation	0.222 ms
	Root key signature generation	1.955 ms
	MAC generation	0.106 ms
Receiver	Root key concatenation	0.354 μs
	ECDSA signature concatenation	0.512 μs
	MAC concatenation	0.312 μs
	MAC key concatenation	0.231 μs
	MAC verification	0.109 ms
	MAC key verification (maximum)	0.207 ms
	ECDSA signature verification	3.699 ms

5. CONCLUSION

본 논문은 가용 비트 수가 제약적인 상황에서 실시간 보정 메

시지에 적용 가능한 두 가지 신호 인증 방법을 제안하고 성능을 분석하였다. ECDSA-only 방법은 ECDSA P-256으로 생성한 전자서명을 이용하여 메시지를 인증하며, ECDSA-TESLA 방법은 전자서명으로 검증한 TESLA 키 체인을 사용한 MAC 태그를 통해 메시지를 인증한다.

ECDSA-only 방법의 경우, 전자서명의 크기가 크기 때문에 하나의 서명을 여러 프레임에 나누어 전송하여, TESLA에 비해 긴 시간이 필요하다. 반면에, ECDSA-TESLA 방법의 경우, 전자서명에 비해 짧은 길이의 MAC 태그와 MAC 키를 이용하기 때문에 인증 메시지 전송에 비교적 짧은 시간이 필요하다. 다만, ECDSA-only 방법은 사전에 별도의 경로로 취득한 공개키를 사용하여 서명을 직접 확인함으로써 인증 서비스를 이용할 수 있는 반면, ECDSA-TESLA 방법은 키 체인을 이용하여 인증을 진행하기 전에 그 키 체인과 연관된 루트키에 대한 서명을 확인하여 루트키를 검증하는 추가 작업이 필요하다.

실시간 보정 메시지의 경우 메시지를 사전에 획득하여 인증 데이터를 생성하기 어렵기 때문에 인증에 소요되는 시간이 길어지며, 전자서명만을 사용한 경우보다 TESLA 키 체인을 사용하는 경우 비교적 짧은 TBA와 TTFAF를 갖는 것을 확인할 수 있었다. 다만, 비트 수가 제약적인 상황에서는 TESLA 키 체인과 MAC 태그를 사용하여 인증하는 시간과 전자서명만을 이용하여 인증하는 시간 사이의 차이가 극명히 드러나지 않음을 확인할 수 있었다. 또한 다른 위성 항법 시스템의 인증 프로토콜과 비교 분석한 결과 ECDSA-TESLA 방법을 사용하는 경우는 전자서명만을 사용한 프로토콜보다 짧은 TBA와 TTFAF을 보이나, ECDSA-only 방법과 ECDSA-TESLA 방법 모두 다른 프로토콜에 비해 인증 대상 메시지가 길어서 비트 오류에 비교적 영향을 많이 받는다는 점을 확인할 수 있었다. 본 논문에서 제안하는 두 가지 방법 모두 구현하여 연산 시간을 측정한 결과 모든 연산은 ms 혹은 μs 단위의 시간이 소요되었으며, 이는 모두 인증 시간에 비하여 매우 짧은 시간으로, 그 오버헤드를 무시할 수 있다.

본 논문에서는 인증을 위한 가용 비트로 프레임 당 50 비트 이하의 예비 영역을 사용하는 상황을 가정하여 인증 메시지 구조를 설계하였다. 향후 다른 QZSS의 신호 인증에서 사용하는 것과 같이 인증을 위한 비트 영역이 명시적으로 할당되어 가용 비트 수가 증가하는 경우, 본 논문에서 제안한 방법보다 빠른 시간 내에 인증이 가능할 것으로 기대된다. 또한 ECDSA의 서명 중 r은 생성 시 인증 메시지가 필요 없기 때문에, 메시지 취득 전 r을 미리 생성하는 것이 가능하다. 따라서, 인증 대상의 실시간 보정 메시지 취득 과정에서 r을 미리 생성하여 방송하는 경우 더욱 빠른 인증이 가능할 것으로 기대된다.

AUTHOR CONTRIBUTIONS

Conceptualization, Y.Jeon. and M.-K.Lee.; methodology, Y.Jeon. and M.-K.Lee.; validation, Y.Jeon. and H.-Y.Kwon; formal analysis, Y.Jeon.; investigation, Y.Jeon.; resources, J.H.Noh; data curation, Y.Jeon. and H.-Y.Kwon; writing—original draft preparation, Y.Jeon; writing—review and editing, H.-Y.Kwon., J.H.Noh. and M.-K.Lee.; visualization, Y.Jeon; supervision, M.-K.Lee.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

REFERENCES

Anderson, J. M., Carroll, K. L., DeVilbiss, N. P., Gillis, J. T., Hinks, J. C., et al. 2017, Chips-message robust authentication (chimera) for GPS civilian signals, In Proceedings of the 30th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2017), Portland, Oregon, 25-29 September 2017, pp.2388-2416. https://doi.org/10.33012/2017.15206

Barker, E. 2020, NIST Special Publication 800-57 Part 1 Revision 5, Recommendation for Key Management: Part 1 – General. NIST, Tech. Rep, pp.1-171. https://doi.org/10.6028/NIST.SP.800-57pt1r5

Cabinet Office 2022, Quasi-Zenith Satellite System Interface Specification Centimeter Level Augmentation Service (IS-QZSS-L6-005), Cabinet Office. https://qzss.go.jp/en/technical/ps-is-qzss/is_qzss_l6_005_agree.html

Cabinet Office 2023, Quasi-Zenith Satellite System Interface Specification Signal Authentication Service (IS-QZSS-SAS-001), Cabinet Office. https://qzss.go.jp/en/technical/ps-is-qzss/is_qzss_sas_agree.html

Dang, Q. 2012, NIST Special Publication 800-107 Revision 1, Recommendation for Applications Using Approved Hash Algorithms. NIST, Tech. Rep, pp.1-25. https://doi.org/10.6028/NIST.SP.800-107r1

European GNSS Agency 2023, Galileo Open Service Navigation Message Authentication (OSNMA) – Signal-in-Space Interface Control Document (SIS ICD) – Issue 1.1 https://www.gsc-europa.eu/sites/default/files/sites/all/files/Galileo_OSNMA_SIS_ICD_v1.1.pdf

Fernández-Hernández, I., Rijmen, V., Seco-Granados, G., Simón, J., Rodríguez, I., et al. 2014, Design Drivers, Solutions and Robustness Assessment of Navigation Message Authentication for the Galileo Open Service, Proceedings of the 27th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2014), 8-12 Sep 2014, Tampa, Florida, USA, pp.2810-2827. https://www.ion.org/publications/abstract.cfm?articleID=12532

Gaybullaev, T., Jeon, D.-Y., & Lee, M.-K. 2021, Poster: Performance comparison of GNSS navigation message

authentication protocols, The 5th International Symposium on Mobile Internet Security (Mobisec 2021), 7-9 Oct 2021, Jeju, Korea.

Hankerson, D., Menezes, A., & Vanstone, S. 2006, Guide to Elliptic Curve Cryptography (New York: Springer)

Jeon, D.-Y., Gaybullaev, T., Noh, J. H., Joo, J.-M., Lee, S. J., et al. 2022, Performance analysis of authentication protocols of GPS, Galileo and BeiDou, Journal of Positioning, Navigation, and Timing, 11, 1-9. https://doi.org/10.11003/JPNT.2022.11.1.1

Jeon, Y., Noh, J. H., Kwon, H.-Y., & Lee, M.-K. 2023, Performance analysis of QZSS navigation message authentication protocol, International Conference on Next Generation Computing, Da Nang, Vietnam, 20-23 December 2023, pp.203-206. https://www.earticle.net/Article/A448150

National Institute of Standards and Technology 2008, The Keyed-Hash Message Authentication Code (HMAC), NIST FIPS 198-1, pp.1-13. https://doi.org/10.6028/NIST.FIPS.198-1

National Institute of Standards and Technology 2015, Secure Hash Standard (SHS), NIST FIPS 180-4, pp.1-36. https://doi.org/10.6028/NIST.FIPS.180-4

Perrig, A., Canetti, R., Tygar, J. D., & Song, D. 2000, Efficient authentication and signing of multicast streams over lossy channels, In Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000, Berkeley, CA, USA, 14-17 May 2000, pp.56-73. https://doi.org/10.1109/SECPRI.2000.848446

Stallings, W. 2023, Cryptography and Network Security: Principles and Practice, 8th Edition (Harlow: Pearson Education Limited)

Wu, Z., Liu, R., & Cao, H. 2019, Ecdsa-based message authentication scheme for BeiDou-ii navigation satellite system, IEEE Transactions on Aerospace and Electronic Systems, 55, 1666-1682. https://doi.org/10.1109/TAES.2018.2874151

Wu, Z., Zhang, Y., Liu, L., & Yue, M. 2020, Tesla-based authentication for BeiDou civil navigation message, China Communications, 17, 194-218. https://doi.org/10.23919/JCC.2020.11.016

Youjin Jeon received the B.A. in English Language and Literature and the B.S. in Information and Communication Engineering from Inha University, Incheon, Korea in 2023. She is currently pursuing an M.S. in the Department of Electrical and Computer Engineering at the same university. Her research interests include information security, biometrics, authentication, and AI security.

Hee-Yong Kwon received the B.S., M.S., and Ph.D. degrees in Computer Engineering from Inha University, in 2015, 2017, and 2024, respectively. He is currently working as a Postdoctoral Researcher at Inha University, Korea. His research interests include communication system security, cryptographic algorithms, and information security.

Jae Hee Noh recently works as a senior researcher at Korea Aerospace Research Institute (KARI). She received B.S., M.S. and Ph.D. degrees from Chungnam National University, Department of Electronic Engineering in 2017, 2019 and 2022. Her research interests include designing the satellite navigation signal, GNSS receiver, anti-spoofing techniques and navigation message authentication.

Mun-Kyu Lee received the B.S. and M.S. degrees in Computer Engineering from Seoul National University in 1996 and 1998, respectively, and the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University in 2003. From 2003 to 2005, he was a senior engineer at Electronics and Telecommunications Research Institute, Korea. He is currently a professor in the Department of Computer Engineering at Inha University, Korea. His research interests are in authentication, privacy-enhancing technology, applied cryptography, blockchain security, and AI security.