



2026 논문집



5.20(수) ~ 23(토)
라카이 샌드파인

주최 kips Korea Information Processing Society
한국정보처리학회

주관 kips Korea Information Processing Society
한국정보처리학회

후원 KCST 한국과학기술단체총연합회 2025-2026 강원 방문이 해 GWT 강원관광재단 강릉시 강릉관광개발공사

서울대학교 컴퓨터연구소
INSTITUTE OF COMPUTER TECHNOLOGY

CS Lab
Cognitive Systems Lab @KWINA
지능형 비즈니스 연구실

협찬 Libertron AI & FPGA 임베디드 시스템 설계 전문기업 COONTEC kt 대신정보통신(주) DESILO MarkAny* (주)범일정보

ORZ TECHNOLOGY

아이티센 엔텍

SK telecom

eroun&company

TGV (주)티씨브이

lim 세림TSG

디엑스 솔루션

BonC Innovators

비트컴퓨터

TRIZN

kpc 한국생산성본부

BITPLANET

SFACSPACE

SysOne Since 1982

GlobalTelecom

lab

올포랜드

UbiTec [주]유비텍

LIG 시스템

(주)그린텍아이엔씨

NEOBRIX

NCT Network Core Technology

WI(주)우리아이티

JWATS

이주태진티엔에스

rwink

LG U+

HUAWEI

GL associates

TTA 한국정보통신기술협회

무순

	064 네트워크 플로우 특성 기반 프라이버시-침입 탐지 성능 균형을 위한 선택적 보호 프레임워크 KIPS_C2026A0162	▶ 문가원*, 박수지, 이일구(성신여자대학교) • 213
	065 비동기 상호작용 환경에서의 IRL 기반 공격-방어 전략 공진화를 위한 Multi-Agent 강화학습 프레임워크 KIPS_C2026A0168	 손우영*, 권순홍, 이종혁(세종대학교) • 216
	066 컨테이너 환경에서 디코이 자원 기반 Reconnaissance 탐지 기법 KIPS_C2026A0169	 권순홍*, 손우영, 이종혁(세종대학교) • 220
	067 Shared Knowledge Base Poisoning Enables Cross-Application Zero-Click PII Exfiltration in Self-Hosted RAG KIPS_C2026A0174	 Chia Ni Lim*, Byoungyoung Lee(Seoul National University) • 224
	068 SMEs 환경에서 프롬프트 유출 방지를 위한 3단계 질의 필터링 프레임워크 KIPS_C2026A0175	 이창규*, 유현창(고려대학교) • 228
	069 위성체 내부 시스템 Taxonomy 구축 및 이상 징후 탐지를 위한 서브시스템 부호 식별 체계 KIPS_C2026A0179	 ▶ 정해린*, 김혜정, 김은서, 조승현, 박기웅(세종대학교) • 232
	070 On-Device AI 모델 소유권 증명을 위한 구조화된 다중 트리거 워터마킹 프레임워크 KIPS_C2026A0180	 ▶ 송민희*, 손우영, 박재형, 이종혁(세종대학교) • 235
	071 On-Device AI 환경에서의 불법 모델 탐지를 위한 유사도 검증 프레임워크 KIPS_C2026A0181	 ▶ 민윤홍*, 박재형, 이하늘, 이종혁(세종대학교) • 238
	072 리눅스 커널의 깊은 동시성 버그 탐지를 위한 스케줄링 연구 KIPS_C2026A0184	 김을규*, 이병영(서울대학교) • 241
	073 ReLU MLP 모델의 활성화 경계 인식 기반 역공학 방지 기법 KIPS_C2026A0194	 ▶ 이성현*, 박재형, 김지혜, 이종혁(세종대학교) • 244
	074 클라우드 네이티브 환경의 공공 클라우드 단일 보안검증 체계 설계 방안: 서버리스 아키텍처의 기술적 충돌 분석을 중심으로 KIPS_C2026A0196	 이예지*, 정광식(한국방송통신대학교) • 247
	075 공개된 MCP 취약점 교차 매핑 기반 공격 사례 분석 KIPS_C2026A0201	 남장우*, 레리사 아테마 질차, 김득훈, 곽진(아주대학교) • 251
	076 A2A Agent Card 위변조 탐지를 위한 JWS 기반 프로세스 KIPS_C2026A0204	 유진*, 송유래, 김득훈, 곽진(아주대학교) • 255
	077 자동화된 SAST와 지속 학습을 통합한 DistilBERT 기반 선제적 취약점 개선 프레임워크 KIPS_C2026A0212	 ▶ 김영현*, 김태현, 김영중(숭실대학교) • 259
	078 iOS AOP 펌웨어 런타임 패치 기법 연구 KIPS_C2026A0213	 정승민*, 류재철(충남대학교) • 262
	079 SOAR 오케스트레이션 기반 제로트러스트 성숙도 진단 자동화 프레임워크 KIPS_C2026A0214	 ▶ 서정우*, 서진우, 공나영, 송민희, 박기웅(세종대학교) • 265
	080 OOXML 기반 악성 MS Word 문서 공격 방식의 진화 및 탐지에 대한 연구 KIPS_C2026A0217	 천송현*, 최두섭, 임을규(한양대학교) • 268

A2A Agent Card 위변조 탐지를 위한 JWS 기반 프로세스

유진¹, 송유래², 김득훈³, 콕진⁴

¹아주대학교 사이버보안학과 석사과정, 정보보호응용및보증연구실

²아주대학교 사이버보안학과 석박통합과정, 정보보호응용및보증연구실

³아주대학교 소프트웨어융합연구소 박사후연구원

⁴아주대학교 사이버보안학과 교수

yujin.isaa@gmail.com, yrsong.isaa@gmail.com, dhkim.isaa@gmail.com, security@ajou.ac.kr

JWS-Based Process for Tamper Detection in A2A Agent Card

Jin Yu¹, Yu-Rae Song², Deuk-Hun Kim³, Jin Kwak⁴

^{1,2}ISAA Lab., Dept. of Cyber Security, Ajou University

³Inst. for Computing and Informatics Research, Ajou University

⁴Dept. of Cyber Security, Ajou University

요 약

A2A(Agent-to-Agent) 프로토콜은 에이전트 메타데이터를 Agent Card라는 JSON(JavaScript Object Notation) 문서로 배포한다. Agent Card에는 통신 엔드포인트와 인증 방식이 포함되어 있어 위변조 시 보안 위협으로 이어질 수 있으나, A2A 프로토콜 명세는 서명을 선택 사항으로 규정하며 위협 시나리오와 적용 절차를 정의하지 않아 실질적인 적용이 어렵다. 본 논문은 이러한 위협과 명세의 부재를 분석하고, TLS(Transport Layer Security)의 종단 간 무결성 보장에서의 한계점을 보완하는 JWS(JSON Web Signature) 기반 서명 방안을 제안한다.

1. 서론

다양한 작업을 수행하는 AI(Artificial Intelligence) 에이전트가 확산되며, 각 에이전트가 역할을 분담하고 협력하는 멀티에이전트 환경이 산업 전반에 도입되고 있다[1]. 그러나 서로 다른 프레임워크와 벤더로 개발된 에이전트들은 고유한 통신 방식을 가지고 있어, 이기종 에이전트 간 협업을 위해서는 통신 규격을 통일하기 위한 별도의 연동 작업이 요구된다. 이를 해결하기 위해 Google은 2025년 기존 웹 표준 기반의 A2A(Agent-to-Agent) 프로토콜을 공개하였다[2].

A2A 생태계에서 에이전트는 자신의 기능과 통신 방식을 Agent Card로 공개하며, 클라이언트는 이를 신뢰의 근거로 협업 상대를 결정한다. Agent Card에는 통신 엔드포인트 URL(Uniform Resource Locator)과 인증 방식이 포함되어 있어, 위변조 시 공격자가 통신 대상 및 인증 방식을 조작하는 보안 위협이 발생할 수 있다. 그러나 A2A 명세는 Agent Card 서명을 선택 사항으로 규정하며, 위협 분석과 적용 절차가 정의되어 있지 않다[2].

본 논문의 구성은 다음과 같다. 2장에서는 Agent Card 구조, TLS(Transport Layer Security)의 종단 간 무결성 한계, JWS(JSON Web Signature), JCS(JSON Canonicalization Scheme), JWKS(JSON Web Key Set) 표준을 정리한다. 3장에서는 위협을 분석하고 JWS+JCS 기반 서명 방안을 제안한다. 4장에서는 위변조 탐지 효과와 서명·검증 오버헤드를 실험으로 검증하며, 5장에서 결론을 맺는다.

2. 관련 연구

2.1 A2A 프로토콜과 Agent Card

A2A 프로토콜은 에이전트 간 태스크 위임, 상태 스트리밍, 푸시 알림 등을 정의하며, Agent Card는 에이전트 URI, 지원 스킴, 인증 방식 등의 메타데이터를 담은 JSON(JavaScript Object Notation) 문서이다[2].

A2A 프로토콜 명세 8.4절에서는 Agent Card에 JWS 서명을 첨부하는 구조를 기술하지만, 구현 의무를 부여하지 않는다. 공식 Python SDK(Software Development Kit) 역시 서명 기능을

선택적 옵션으로 제공하지만, 구체적인 위협 시나리오와 적용 절차는 명세와 SDK에서 정의되어 있지 않다[3].

2.2 TLS의 종단 간 무결성 한계

TLS의 Reverse Proxy, CDN(Content Delivery Network), API(Application Programming Interface) 게이트웨이가 배포 경로에 포함될 경우 TLS를 종료하고 내부 연결을 다시 수립하며, 각 구간은 독립적인 TLS 세션으로 처리된다[4]. 이는 클라이언트와 서버 간의 전송 경로를 암호화하지만, 종단 간(end-to-end) 무결성은 보장하지 않는다.

Durumeric 등은 실제 배포 환경에서 TLS 인터셉션이 광범위하게 발생하며, 인터셉션 구간에서 전달 내용의 무결성이 보장되지 않음을 실증하였다[5]. TLS 인터셉션 구간에서 Agent Card 내용이 수정되어도 클라이언트는 탐지할 수 없다. Agent Card처럼 한 번 배포되면 캐싱되어 재사용되는 문서는 TLS 인터셉션에 특히 취약하다[2].

2.3 JWS, JCS, JWKS

JWS(RFC 7515)는 JSON 데이터에 비대칭 서명을 적용하는 표준으로, 헤더와 페이로드를 Base64URL로 인코딩한 뒤 서명한다[6]. COSE(RFC 8152) 등의 대안은 CBOR 기반으로 JSON 환경에 적합하지 않다.

JCS(RFC 8785)는 JSON 직렬화 순서를 정규화하여 동일한 논리적 내용이 항상 동일한 바이트 시퀀스를 생성하도록 보장하며, 서명 전 적용 시 필드 순서 변경에 의한 서명 무효화를 방지한다[7].

JWKS(RFC 7517)는 공개키 집합을 JSON 형식으로 게시하는 표준으로, X.509 인증서 기반 대비 별도 PKI(Public Key Infrastructure) 인프라 없이 적용할 수 있다[8].

본 논문은 세 표준을 기반으로 Agent Card 서명 생성과 검증 흐름을 구성한다.

3. 위협 분석 및 대응

공격자는 전송 경로 상의 중간 노드에서 Agent Card를 도청·수정할 수 있으나, 에이전트 서버의 개인키에는 접근하지 못한다고 가정한다.

3.1 T1: 엔드포인트 하이재킹

공격자는 url 필드를 자신이 통제하는 서버 주소로 교체한다.

```
{ // Original
  "supported_interfaces": [{"url": "https://agent.example.com/a2a"}]
// After tampering
  "supported_interfaces": [{"url": "https://attacker.com/a2a"}] }
```

클라이언트는 변조된 URL을 신뢰하고 공격자 서버에 요청을 보낸다. 공격자는 요청을 원래 서버로 중계하면서 통신 내용을 열람하거나 조작할 수 있다.

3.2 T2: 인증 다운그레이드

공격자는 security_requirements 필드를 제거하거나, OAuth2 방식의 authorization_url을 피싱 서버 주소로 교체한다.

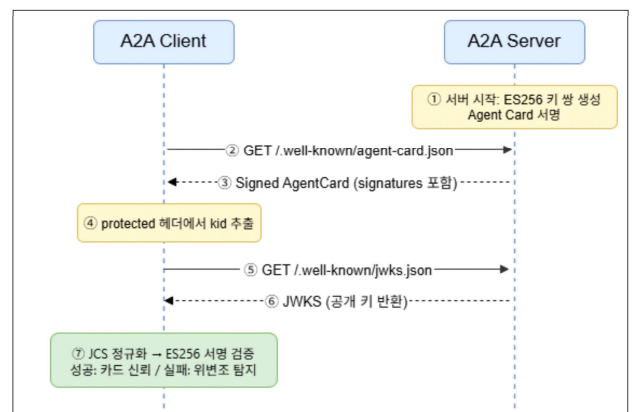
```
{ // Original
  "security_requirements": [{"oauth2": ["read", "write"]} ]
// After tampering -- authentication requirements removed
  "security_requirements": [ ] }
```

인증 요구사항이 제거되면 클라이언트는 자격 증명 없이 요청을 보내고, 공격자는 인증 없는 통신을 가로채거나 자격 증명을 탈취할 수 있다.

3.3 보안 요구사항 및 제안 프로세스

두 가지 위협에 대응하기 위해 에이전트 서버의 서명 시점 바인딩과 클라이언트의 의무 검증 두 가지 요구사항을 도출하고, 이를 충족하는 절차를 (그림 1)과 같이 제안한다.

공격자가 어느 필드를 수정하면 JCS 정규화 결과가 달라져 원본 서명과 불일치하므로 T1, T2 공격은 클라이언트의 서명 검증 단계에서 탐지된다. 서명 재생성에는 서버의 개인키가 필요하므로, 개인키를 탈취하지 않는 한 재서명이 불가능하다.



(그림 1) Agent Card JWS 서명 및 검증 흐름

3.3.1 서버 측: Agent Card 서명 절차

Step 1. JCS 정규화

Agent Card JSON에 JCS를 적용하여 키를 사전 순으로 정렬하고 최소 직렬화 형식으로 변환하여 정규화한다.

Step 2. JWS 헤더 구성

서명 알고리즘과 키 식별자(Key ID, kid)를 헤더에 기재한다. 서명 알고리즘인 ES256은 RS256 대비 서명 크기가 작고 연산이 빠르면서 비슷한 보안 수준을 제공하여 소형 JSON 문서의 빈번한 서명·검증에 적합하다. kid는 JWS Protected Header에 포함되는 파라미터로, 공개키 집합 내에서 키를 식별하는 문자열이며 클라이언트가 검증에 사용할 공개키를 조회하는 데 사용된다.

Step 3. 서명 생성 및 첨부

정규화된 페이로드에 개인키로 JWS 서명을 생성하고, 결과 토큰을 Agent Card의 signatures 필드에 삽입하여 배포한다.

Step 4. 공개키 게시

서명 검증에 사용할 공개키를 /.well-known/jwks.json에 JWKS 형식으로 게시한다.

3.3.2 클라이언트 측: Agent Card 검증 절차

Step 1. 서명 필드 존재 확인

수신한 Agent Card에 signatures 필드가 없으면 미서명 문서로 간주하고 사용을 거부한다. 이 단계가 공격자의 서명 제거 시도를 차단한다.

Step 2. 공개키 조회

JWS 헤더의 kid 값을 기반으로 /.well-known/jwks.json에서 공개키를 조회한다. JWKS 엔드포인트는 동일 출처에 위치한다고 가정하며 TLS 신뢰를 기반으로 조회한다.

Step 3. JCS 정규화 및 서명 검증

signatures 필드를 제외한 Agent Card 본문을 JCS로 정규화한 뒤, 공개키로 서명을 검증한다. 검증 실패 시 위변조된 문서로 판단하고 폐기한다.

Step 4. 검증 통과 후 사용

서명 검증이 성공한 경우에만 Agent Card에 기재된 엔드포인트 URL과 인증 방식을 신뢰하고 사용한다.

4. 실험 및 평가

4.1 실험 환경

제안 방안의 실험 가능성을 <표 1>의 환경에서 프로토타입으로 검증하였다.

<표 1> 실험 환경

항목	내용
언어	Python 3.12
서명 라이브러리	a2a-sdk (PyJWT 2.11.0)
서명 알고리즘	ES256 (ECDSA P-256 / SHA-256)
정규화 방식	JCS (RFC 8785), SDK 내장 함수

4.2 실험 설계

실험은 세 축으로 구성하였다. 첫째, T1·T2 위협 시나리오에서 제안 방안의 방어 효과를 확인하였다. 둘째, 성능 벤치마크는 통계적 안정성을 확보하기 위해 N=1,000회 반복으로 서명·검증 지연시간을 측정하였다. 셋째, 카드 복잡도별 오버헤드를 측정하여 제안 방안의 범용 적용 가능성을 검증하였다.

4.3 실험 결과

4.3.1 시나리오별 방어 효과

위변조 탐지 실험은 2종의 변조 시나리오(T1, T2)에 대해 각각 N=1,000회 수행하였다. <표 2>와 같이 모든 시나리오에서 거부율 100%를 달성하였고, 정상 카드 검증에서는 오검출이 발생하지 않았다.

<표 2> 시나리오별 방어 효과

공격	기존	제안	거부율
T1	탐지 불가	서명 불일치 → 거부	100%
T2	탐지 불가	서명 불일치 → 거부	100%

4.3.2 서명·검증 지연시간

<표 3>과 같이 서명 생성과 검증 모두 평균 0.2ms 미만으로 측정되었다.

<표 3> ES256 서명·검증 지연시간 (N=1,000)

연산	평균	P95	P99
서명 생성	0.08ms	0.12ms	0.18ms
서명 검증	0.11ms	0.16ms	0.22ms

P95와 P99는 각각 전체 측정값 중 하위 95%, 99%가 해당 값 이하임을 나타내는 백분위수로, 이상값을 포함한 지연시간 분포를 파악하는 데 사용된다.

4.3.3 카드 복잡도별 오버헤드

에이전트가 제공하는 기능 단위인 스킴 수가 증가할수록 카드 JSON 크기가 커진다. <표 4>와 같이 스킴 수에 비례하여 지연시간이 증가하나, 스킴 100개 조건에서도 평균 0.5ms 미만으로 유지되었다.

<표 4> 카드 복잡도별 서명·검증 지연시간 (N=1,000)

스킴 수	서명 평균	검증 평균
1개	0.07ms	0.10ms
10개	0.11ms	0.14ms
50개	0.28ms	0.29ms
100개	0.48ms	0.47ms

5. 결론

본 논문은 A2A 프로토콜에서 Agent Card 위변조로 인한 보안 위협을 분석하고, JWS+JCS 기반의 탐지 방안을 제안하였다. 실험을 통해 제안 방안이 전송 경로 상의 공격 시도를 오검출 없이 차단하며, 서명·검증 오버헤드가 낮음을 확인하였다. 또한 스킴 수가 증가하더라도 지연시간 증가 폭이 작아, 단순한 에이전트부터 복잡한 에이전트까지 제안 방안을 범용적으로 적용할 수 있다.

제안 방안은 별도 인프라 없이 기존 JWKS 엔드포인트를 재사용할 수 있으며, 개인키 탈취 및 초기 신뢰 형성 시점의 MITM(Man-In-The-Middle)을 제외한 전송 경로 상 공격에 대해 탐지 수단을 제공한다.

다만, JWKS는 동일 출처 서버에 직접 조회된다고 가정하므로 JWKS 자체가 변조되는 경우는 보호 범위에 포함되지 않는다. 또한 에이전트 서버의 개인키가 탈취되거나 서버가 침해된 경우에도 방어가 어렵다.

그리고 초기 신뢰 형성 시점에서 공격자가 Agent Card와 JWKS를 동시에 조작하고 자신의 키로 재서명하는 경우, 탐지하는 데 한계가 존재한다.

향후에는 키 관리 및 순환 정책을 구체화하고, 초기 신뢰 형성 시점의 한계를 보완하기 위한 신뢰 체계를 탐색하며 네트워크 지연, 운영 변수 등을 고려하여 실제 A2A 에이전트 간 통신 환경에서의 적용 가능성을 검증할 예정이다.

사사문구

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2026-25486953).

참고문헌

- [1] Gartner, “Gartner Predicts 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026”, Press Release, Stamford, CT, Gartner Inc., Aug. 2025.
- [2] Google, “Agent2Agent Protocol Specification”, Linux Foundation, 2025.
<https://a2a-protocol.org/latest/specification/>
- [3] Google, “a2a-python SDK”, GitHub, 2025.
<https://github.com/a2aproject/a2a-python>
- [4] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, RFC 8446, IETF(Internet Engineering Task Force), Fremont, CA, 2018.
<https://datatracker.ietf.org/doc/html/rfc8446>
- [5] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, “The Security Impact of HTTPS Interception”, in Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, Feb. 2017, pp. 1-6.
- [6] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Signature (JWS)”, RFC 7515, IETF(Internet Engineering Task Force), Fremont, CA, 2015.
<https://datatracker.ietf.org/doc/html/rfc7515>
- [7] A. Rundgren, B. Jordan, and S. Erdtman, “JSON Canonicalization Scheme (JCS)”, RFC 8785, IETF(Internet Engineering Task Force), Fremont, CA, 2020.
<https://datatracker.ietf.org/doc/html/rfc8785>
- [8] M. Jones, “JSON Web Key (JWK)”, RFC 7517, IETF(Internet Engineering Task Force), Fremont, CA, 2015.
<https://datatracker.ietf.org/doc/html/rfc7517>