서버리스 환경에서의 비밀키 유출 방지를 위한 경량 난독화 기법에 관한 연구

안승민, 배채은, 권다은, 김시환, 김예찬, 김형지, 조재문, 김예진, 김동찬*
국민대학교

{bryan0126, co5eun, ekdms3809, sihwan031008, tomking0820, aloa2012, henryjm, alice1225kim, dckim*}@kookmin.ac.kr

A Study on Lightweight Obfuscation for Preventing Key Leakage in Serverless Environment

Ahn Seung-Min, Bae Chae-Eun, Kwon Da-Eun, Kim Si-Hwan,
Kim Yae-Chan, Kim Hyung-Ji, Jo Jae-Moon, Kim Ye-Jin, Kim Dong-Chan*
Kookmin University

요 약

서버리스 환경에서 공격자가 자격 증명(Credential)을 탈취하면 배포 패키지 내 비밀키가 정적 분석으로 노출될 수 있다. 또한 구현에 포함된 고정 상수가 노출될 경우, 상수 테이블 0 치환 변조로 비밀키가 노출될 수 있는 위험이 존재한다. 본 논문은 이를 완화하기 위해 SSS 기반 key split과 constant encoding을 결합한 알고리즘 수준 보호를 적용하고, Tigress로 제어 흐름 난독화를 추가한다. 보안성 평가는 정적 분석 관점에서 비밀키 및 고정 상수의 직접 노출이 완화됨을 확인하였다. 성능 평가는 AWS Lambda 환경에서 수행하였으며, 전체 평균 실행 시간 차이는 약 +424.4ms로 측정되었다.

Ⅰ. 서 론

서버리스 컴퓨팅 환경에서는 애플리케이션 로직이 함수 단위로 패키징되어 클라우드에 배포된다. 이때 공격자가 자격 증명(Credential)을 탈취하면 배포된 함수 코드를 확보하여 내부에 포함된 비밀키 등 민감 정보를 정적 분석으로 추출할 수 있다[1].

코드 보호 기법 중 하나인 난독화는 프로그램의 의미를 유지한 채 분석 난이도를 높이는 방법이다. Tigress는 C 언어 기반 난독화 도구로, 제어 흐름 평탄화(Flatten)와 불투명 조건문 삽입(AddOpaque) 등 제어 흐름 변환을 제공한다. 그러나 해당 도구의 변환은 적용 지점과 결과가 무작위적으로 결정될 수 있어, S-Box 및 T-Table과 같은 고정 상수가 난독화되었음을 보장하기 어렵다. 그 결과 정적 분석 과정에서 암호 알고리즘 구현이 상대적으로 쉽게 식별되거나, 코드 내 하드코딩된 비밀키가 노출될 가능성이 남는다.

본 논문은 서버리스 환경에서의 비밀키 및 고정 상수 노출 위험을 완화하기 위해 SSS 기반 key split과 constant encoding을 결합한 알고리즘 수준 난독화 기법을 제안하고, Tigress를 추가 적용한다. 이후 보안성 및 성능 평가를 통해 제안 기법의 효과와 오버헤드를 분석한다.

Ⅱ. 문제 정의

i. 대상 시스템

본 논문에서 고려하는 대상 시스템은 T-Table 기반 AES 구현이다. AES 내부 연산 중 SubBytes에서만 S-Box가 사용되며, 이는 AES에서 유일한 비선형 변환으로 알려져 있다[2]. 따라서 AES의 안전성은 SubBytes의 비선형성에 크게 의존한다[2].

T-Table 기반 구현은 SubBytes와 MixColumns 연산을 사전 계산된 테이블(T0~T3)로 통합하여 성능을 최적화하며, 각 라운드에서 테이블 조회 결과와 라운드 키의 XOR 연산을 통해 라운드 변환을 수행한다.[3] 또한 키 확장 과정에서 S-Box와 Rcon이 사용된다[2]. 이러한 고정 상수 테이블은 바이너리에 포함될 수 있어 정적 분석의 주요 단서가 된다.

ii. 위협 모델 및 공격 시나리오

본 연구의 위협 모델은 공격자가 자격 증명 탈취를 통해 서버리스 함수의 배포 패키지와 암호화된 데이터(DB, 스토리지 등)를 획득한 상황을 가정한다[1]. 공격자는 런타임 메모리에는 접근할 수 없으며, 획득한 코드에 대해 정적 분석만 수행할 수 있다[1].

공격 시나리오는 공격자가 배포 패키지를 획득한 뒤, 바이너리에 포함된 AES의 고정 상수(S-Box, T-Table, Rcon)를 임의의 값으로 치환하여 동작을 변조하는 상황을 포함한다. 특히 고정 상수 테이블을 0으로 치환할 경우 AES 내부 연산의 비선형성이 소거되어 연산 구조가 단순화되며, 그 결과 암·복호 과정에서 키의 영향이 직접적으로 드러날 수 있다. 예를 들어 T-Table 또는 S-Box를 0으로 치환하면 키 스케줄의 성질이 약화되어 키 복구가 용이해질 수 있다. 또한 모든 테이블을 0으로 치환하는 경우 암호문이 마스터 키와 동일해지는 등 심각한 취약점이 발생할 수 있다. 이러한 가능성은 AES에서 S-Box 비선형성이 핵심 역할을 수행한다는 점과 함께, 구현 코드에 포함된 고정 상수의 기밀성과 무결성 보호가 필요함을 시사한다[2].

Ⅲ. 난독화 기법 설계

본 장에서는 제안 기법의 설계를 설명한다. 제안 기법은 비밀키에 key split을 적용한 뒤, 분할된 share와 AES 고정 상수(S-Box, T-Table, Rcon)에 constant encoding을 적용하고, 마지막으로 Tigress를 적용하여 제어 흐름 수준의 난독화를 추가한다.

i. Tigress 기반 코드 난독화

Tigress는 C 언어 기반 난독화 도구이며, 본 연구에서는 Tigress v4.0.11을 사용하여 AES 관련 함수에 난독화를 적용하였다[4]. 제어 흐름 평탄화(Flatten)는 dispatch=switch, splitBasicBlocks=true 옵션으로 적용하였고, 불투명 조건문 삽입(AddOpaque)은 count=300으로 설정하였다. 또한 산술 연산 인코딩(EncodeArithmetic)을 추가 적용하였다.

다만 Tigress의 변환은 적용 지점과 결과가 무작위적으로 결정될 수 있으므로, S-Box, T-Table, Rcon과 같은 고정 상수가 항상 난독화된다고 보장하기 어렵다. 그 결과 고정 상수의 원형 노출 및 상수 테이블 0 치환 변조에 대한 위험이 잔존할 수 있다. 따라서 Tigress 적용 이전에 알고리즘 수준의 보호를 선적용하는 것이 필요하며, 본 연구에서는 비밀키에는 key split을, 상수 테이블 및 share에는 constant encoding을 적용한 뒤 Tigress를 적용한다.

ii. Key Split

Key Split은 SSS(Shamir's Secret Sharing)를 적용하여 비밀키를 분할하는 기법이다[5]. SSS는 (k,n) 임계값 구조를 지원하며, 비밀 값 s를

상수항으로 갖는 (k−1)차 다항식 $f(x) = s + a_1 x + \cdots + a_{k-1} x^{k-1}$을 정의한다. 이때 상수항을 제외한 계수 $a_1, \cdots, a_{k-1}$은 무작위로 선택되며, 서로 다른 n개의 입력값에 대한 $f(x)$의 출력값이 각 share가 된다. 복원 시에는 k개 이상의 share를 이용해 라그랑주 보간법으로 원본 비밀값 s를 계산한다.

일반적인 SSS 기반 키 관리에서는 share를 서로 다른 저장소에 분산 저장하는 방식을 고려할 수 있다. 그러나 본 연구가 가정하는 서버리스 환경에서는 자격 증명 탈취 시 공격자가 관련 저장소 전반에 접근할 수 있으므로, 저장소 분산만으로는 안전성을 확보하기 어렵다. 이에 본 연구에서는 모든 share를 동일 바이너리 내에 저장하되, 정적 분석에서 share 식별이 어렵도록 각 share에 constant encoding을 적용하는 방식을 채택하였다.

본 연구에서는 (3,5) 임계값 구조를 사용하여 비밀키를 5개의 share로 분할하였다. 분할된 share는 인코딩된 형태로 저장되며, 실행 시점에 디코딩 및 복원된 후 암호 연산에 사용된다.

iii. Constant Encoding

constant encoding은 상수 값을 변형된 형태로 저장한 뒤, 실행 시 역변환을 통해 원본 값을 복원하는 기법이다. 본 연구에서는 share 값을 바이트 단위로 처리하되, 각 바이트를 상·하위 nibble로 분할하여 별도 배열에 저장하고, 선형 변환과 XOR 마스크를 결합하여 인코딩한다. 각 nibble $n \in \{0, \cdots, 15\}$에 대해 인코딩/디코딩은 다음과 같이 정의된다.

$$\mathrm{enc}(n) = ((n \times MLT) + \mathrm{ADD}) \oplus \mathrm{XOR\_MASK}$$

$$n = (((\mathrm{enc}(n) \oplus \mathrm{XOR\_MASK}) - \mathrm{ADD}) \times \mathrm{INV\_MULT}) \bmod 2^4$$

여기서 INV_MULT는 MULT의 모듈러 역원이며, gcd(MULT,16)=1을 만족하는 MULT를 선택한다.

또한 AES 테이블 상수에 대해서는 인덱스에 따라 변화하는 마스크를 결합하는 position-dependent XOR 방식을 적용한다. 비트폭이 w인 테이블 T와 인코딩 테이블 E는 다음을 만족한다.

$$E[i] = T[i] \oplus K \oplus \mathrm{ROL\_w}(((i \times M) \bmod 2^w), r)$$

$$T[i] = E[i] \oplus K \oplus \mathrm{ROL\_w}(((i \times M) \bmod 2^w), r)$$

여기서 K는 XOR 키, M은 인덱스 i에 곱해져 위치별 마스크를 생성하는 계수, r은 좌회전 비트 수를 의미한다. S-Box는 w=8, T-Table 및 Rcon은 w=32를 사용한다.

이와 같이 인코딩된 테이블은 원본과 상이한 값으로 저장되므로 고정 상수의 원형 노출을 완화한다. 또한 공격자가 테이블을 0으로 치환하는 등 임의로 변조하더라도 디코딩 과정에서 필요한 관계가 유지되지 않아 정상적인 암·복호 동작이 성립하기 어렵고, 결과적으로 상수 테이블 변조 공격의 효과를 제한한다.

Ⅳ. 보안성 및 성능 평가

i. 보안성 평가

제안 기법이 적용된 바이너리에 대해 정적 분석 기반 보안성을 평가하였다. 분석에는 정적 분석 도구(IDA Free 9.2, Ghidra 11.0)를 사용하였다. 난독화 적용 전에는 S-Box 및 T-Table 등 고정 상수가 원형에 가까운 형태로 바이너리에 포함되었고, 비밀키 또한 전역 배열 형태로 노출되었다. 난독화 적용 후에는 AES 상수와 share에 constant encoding을 적용하여 고정 상수의 원형 노출을 감소시켰다. 또한 비밀키는 SSS (3,5) 구조로 분할하고 각 share를 nibble 단위로 인코딩된 형태로 저장하여, 정적 분석에서 비밀키가 직접적으로 드러나지 않도록 구성하였다.

ii. 성능 평가

AWS Lambda(메모리 2,048MB, Python 3.11, ap-northeast-2 리전) 환경에서 난독화 적용에 따른 성능을 측정하였다. 파일 크기는 100KB, 500KB, 1MB, 4MB로 설정하였으며, 각 조건에서 워밍업을 2회 수행한 후 1,000회 반복 측정하였다. 측정 값은 함수 실행의 처리 시간으로 정의하였고, 네트워크 지연은 제외하였다. 표 1은 파일 크기별 암·복호 처리 시간과 차이를 제시한다. 표 2는 바이너리 크기, 최대 메모리 사용량, 평균 실행 시간의 변화를 비교한다.

[표 1] 파일 크기별 성능 비교 (N=1,000)

파일 크기	비난독화 버전(ms)	난독화 버전(ms)	차이(ms)
업로드			
100KB	51.3	90.9	+39.6
500KB	86.5	247.4	+160.9
1MB	112.0	416.7	+304.6
4MB	204.6	1394.9	+1190.2
다운로드			
100KB	146.6	176.4	+29.9
500KB	162.8	314.4	+151.7
1MB	190.8	492.2	+301.5
4MB	362.6	1579.9	+1217.2

[표 2] 바이너리 및 실행 환경 비교

항목	비난독화 버전	난독화 버전	차이
바이너리 크기	34.1KB	201.7KB	+167.6KB
최대 메모리 사용량	130MB	135MB	+5MB
평균 실행 시간	203.8ms	1390.3ms	+1186.5ms

실험 결과 업로드 연산의 평균 처리 시간 차이는 +423.8ms, 다운로드 연산의 평균 처리 시간 차이는 +425.1ms였으며, 전체 평균 차이는 +424.4ms로 측정되었다. 바이너리 크기는 +167.6KB 증가하였고, 평균 실행 시간은 +1186.5ms 증가하였다. 최대 메모리 사용량은 +5MB 증가하여 메모리 변화는 제한적인 수준이었다. 성능 저하는 Tigress 적용에 따른 제어 흐름 변환 및 불투명 조건문 삽입으로 인한 연산 증가에 기인하며, 제안 기법은 정적 분석 저항성 향상과 실행 오버헤드 간의 트레이드오프를 갖는다.

Ⅴ. 결 론

본 논문은 서버리스 환경에서의 비밀키 및 고정 상수 노출 위험을 완화하기 위해 SSS 기반 key split과 constant encoding을 결합하고, Tigress를 추가 적용하는 난독화 기법을 제안하였다. 평가 결과, 비밀키와 고정 상수의 직접 노출이 완화되었으며 상수 테이블 변조 공격에 대한 저항성이 향상되었다. 반면 실행 시간과 바이너리 크기 증가가 관찰되었고, 이는 난독화 과정에서 추가 연산이 발생하는 특성에 기인한다.

ACKNOWLEDGMENT

이 논문은 국민대학교 2025학년도 동계방학 학부생 연구참여 프로그램(UROP)의 지원과 2025년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No. RS-2024-00397105, KCMVP 보안수준 3 암호모듈 제작을 위한 핵심기술 개발).

참 고 문 헌

[1] E. Marin, D. Perino, and R. Di Pietro, "Serverless Computing: A Security Perspective" Journal of Cloud Computing, vol. 11, no. 69, pp. 1-26, 2022.

[2] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, Nov. 2001.

[3] J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard," Springer-Verlag, 2002.

[4] C. Collberg, "The Tigress C Diversifier/Obfuscator"

[5] A. Shamir, "How to Share a Secret," Communications of the ACM, vol. 22, no. 11, pp. 612-613, Nov. 1979.