신뢰실행환경을 활용한 스마트 컨트랙트 실행 보호 기술 개발

팀 TrustForge

부산대학교 정보컴퓨터공학부

202155651 이준태

202155648 위재준

202055581 이준혁

지도교수 **권동현**

Contents

1 과제의 목표 **3**

1.1 과제 배경 3

1.2 과제 세부 목표 3

2 요구조건 및 제약 사항 분석에 대한 수정사항 **4**

2.1 유사 시스템 분석 결과에 대한 보완 4

2.2 문제점에 대한 정량적 분석 결과 반영 4

2.3 기능적/비기능적 요구사항의 구체화 5

3 설계 상세화 및 변경 내역 **6**

3.1 초기 설계 요약 6

3.2 상세화된 설계 요소 6

3.2.1 체인코드 실행 구조 6

3.2.2 성능 측정 흐름 6

3.2.3 Remote Attestation 프로토콜 구성 6

3.3 향후 설계 고도화 계획 7

4 갱신된 과제 추진 계획 **8**

4.1 개발 일정 8

5 구성원별 진척도 **9**

6 보고 시점까지의 과제 수행 내용 및 중간 결과 **10**

6.1 WATZ 환경 구성 및 실행 실험 10

6.2 성능 벤치마크 실험 10

6.3 Remote Attestation 기능 실험 14

6.4 Hyperledger Fabric 연동 구조 설계 14

6.5 중간 성과 요약 15

과제의 목표

1.1) 과제 배경

최근 블록체인 기술은 금융, 의료, 물류 등 다양한 분야에서 활용되고 있으며, 특히 Hyperledger Fabric과 같은 프라이빗 블록체인은 높은 확장성과 접근제어 기능으로 주목받고 있다. 그러나 Fabric의 체인코드는 일반 운영체제 환경에서 실행되어 보안상의 한계가 존재하며, 민감한 연산 결과의 위변조 위험 또한 존재한다.

1.2) 과제 세부 목표

본 과제는 WebAssembly 기반의 체인코드를 ARM 보드의 신뢰 실행 환경(TEE: Trusted Execution Environment)에서 실행하고, 실행 결과를 Hyperledger Fabric 블록체인과 연동하는 보안 실행 아키텍처를 구현하는 것을 목표로 한다. 이를 통해 체인코드 실행의 무결성과 신뢰성을 확보하고, 스마트 계약의 보안성을 향상시키고자 한다.

- TEE 환경에서 WebAssembly 체인코드 실행 구조 구현
- TEE 내부 실행 결과에 대한 서명 처리 및 전달
- Hyperledger Fabric과의 연동을 위한 Proxy 체인코드 구성
- 전체 시스템의 실험 및 보안성 검증

요구조건 및 제약 사항 분석에 대한 수정사항

본 과제는 초기 착수보고서를 통해 WebAssembly(Wasm) 기반 스마트 컨트랙트가 실행 보안 측면에서 취약하다는 점에 주목하고, 이를 ARM TrustZone 기반 신뢰 실행 환경(TEE)과 결합하여 보호하고자 하는 구조를 제안하였다. 특히, TEE 내부에서 Wasm 기반 체인코드를 안전하게 실행하고 그 결과를 Hyperledger Fabric 블록체인과 연동함으로써 체인코드의 무결성과 실행 신뢰성을 확보하는 것을 목표로 하였다. 이후 실질적인 환경 구성과 성능 실험을 진행하면서 다음과 같은 요구조건과 제약 사항에 대한 구체화, 수정, 보완이 필요하다는 점이 드러났다.

2.1) 유사 시스템 분석 결과에 대한 보완

초기에는 기존의 스마트 컨트랙트가 대부분 Ethereum Virtual Machine(EVM) 기반으로 실행되며 코드의 투명성과 불변성을 장점으로 갖는 대신, 공격자에게는 오히려 공격 벡터를 노출시킬 수 있다는 문제를 지적하였다. 이에 대한 대안으로 Wasm 기반 플랫폼(예: Polkadot)이 제시되었으며, 이는 이식성과 성능 면에서 장점이 있는 것으로 평가되었다.

하지만 Wasm은 실행 자체에 대한 보호 메커니즘이 부족하다는 단점이 존재하였고, TEE를 결합하는 방안이 실효성이 있을지에 대해서는 실험적 근거가 필요했다. 본 프로젝트에서는 WATZ(Wasm + TrustZone) 환경을 구성하여 이를 실험적으로 평가하였고, 실제로 Wasm AOT(Ahead-of-Time) 방식과 TrustZone을 결합하는 구조가 유사 시스템들에 비해 현실적인 대안이 될 수 있다는 점을 확인하였다. 특히 기존의 직접 작성한 C 기반 TA보다, Wasm 모듈을 샌드박스에서 실행하는 방식이 구현의 단순성, 이식성, 언어 다양성 측면에서 유리함을 확인하였다.

2.2) 문제점에 대한 정량적 분석 결과 반영

초기에는 TEE 도입 시 발생할 수 있는 성능 저하 문제에 대해 명확한 수치 기반 근거가 부족하였다. 이에 따라, 본 프로젝트에서는 WATZ 환경에서 SQLite 및 Genann 기반의 체인코드 워크로드를 AOT Wasm으로 실행하고, 그 결과를 직접 측정하였다. 실험 결과, TEE를 활용한 WATZ 환경은 일반 native 환경에 비해 평균적으로 약 1.6배에서 2.1배 사이의 성능 저하가 있었으며, 이는 보안성을 얻기 위한 대가로서 수용 가능한 수준임을 확인하였다.

또한 WAMR의 실행 모드를 비교해본 결과, 인터프리터 방식보다 AOT 방식이 약 28배 빠르다는 WATZ 논문과 동일한 결과를 실험적으로 재현하였으며, 이는 실제 적용 가능성을 보여주는 중요한 성과이다. 이를 통해 "TEE 도입 시 성능 저하"라는 추상적인 우려를 실제 수치 기반의 근거로 해소할 수 있게 되었다.

2.3) 기능적/비기능적 요구사항의 구체화

초기 요구사항 분석에서는 TEE 내에서 스마트 컨트랙트를 실행하고, 외부와 통신하며, 기존 블록체인 구조와 연동되어야 한다는 정도로 기술되어 있었다. 중간보고 시점에서는 해당 요구사항들이 보다 구체적인 기술 설계와 실험을 통해 명확화되었다.

- 기능적 요구사항 측면에서는, 체인코드 실행 환경으로서 OP-TEE 상에 구동되는 WATZ 런타임을 사용하고, 외부 Fabric peer와는 gRPC 통신을 통해 연결되도록 하였다. 체인코드는 Go 언어로 작성한 후 WASI 기반 Wasm으로 컴파일되어 WATZ 내에서 안전하게 실행되며, 그 결과를 WATZ 내부에서 측정된 해시 기반 증거(evidence)와 함께 외부로 전달하는 구조로 설계되었다.
- 비기능적 요구사항 중 "이식성" 측면에서는 Raspberry Pi 또는 QEMU 가상환경이 아닌, NXP MCIMX8M-EVK 보드에서 WATZ를 실제 구동하여 실험을 수행하였다. 해당 보드는 ARM Cortex-A53 기반의 TrustZone을 지원하며, WATZ의 secure world 실행을 위한 하드웨어 요건을 만족하였다. 성능 실험은 AOT 방식으로 실행된 SQLite 및 Genann 벤치마크를 대상으로 수행되었으며, 논문에서 보고된 수치와 유사한 실행 시간이 확인되어 성능 재현 가능성이 검증되었다.
- "보안성" 측면에서는, WATZ 내부에 이미 Remote Attestation 기능이 포함되어 있으며, 프로토콜 흐름(msg0 msg3)과 evidence 생성 기능도 구현되어 있다. 다만, 아직 Verifier와의 실제 네트워크 통신 테스트는 수행하지 않았으며, 향후 RA 전용 서버와의 연동 실험을 통해 전체 인증 흐름을 검증할 계획이다.

설계 상세화 및 변경 내역

3.1) 초기 설계 요약

초기 착수보고서에서는 체인코드를 WebAssembly로 컴파일한 후 ARM TrustZone 기반 TEE 환경(OP-TEE) 내에서 실행하고, 이를 프록시 체인코드를 통해 Hyperledger Fabric과 연동하는 구조를 제시하였다. 체인코드는 안전한 환경 내에서 실행되어 결과의 무결성과 기밀성이 보장되며, 외부에서는 이를 일반 체인코드처럼 사용할 수 있도록 abstraction을 제공하는 것이 목표였다.

3.2) 상세화된 설계 요소

3.2.1 체인코드 실행 구조

- 체인코드는 Go 언어로 작성된 후 WASI 기반 WebAssembly(.wasm)로 컴파일되었다.
- 해당 `.wasm` 파일은 WATZ 런타임이 구동되는 TrustZone secure world 내에서 실행된다.
- 체인코드 실행 시 외부에서 바이트코드를 동적으로 전달하고 secure memory에 적재한다.
- AOT 컴파일된 코드는 WAMR을 통해 sandbox 환경에서 실행된다.

3.2.2 성능 측정 흐름

- 체인코드 실행 성능은 SQLite 및 Genann 워크로드를 이용하여 평가되었다.
- 각각의 워크로드에 대해 AOT 기반 실행 속도 측정을 완료하였다.
- baseline(native) 실행 대비 성능 저하는 평균 1.6∼2.1배 수준으로 확인되었다.
- WATZ 내부의 해시 측정 및 evidence 생성 여부는 성능에 큰 영향을 주지 않는 것으로 보인다.

3.2.3 Remote Attestation 프로토콜 구성

- WATZ 내부에는 이미 msg0∼msg3 기반의 원격 증명 프로토콜이 포함되어 있다.
- 해당 흐름은 프로토콜 설계 및 코드 분석을 통해 파악되었다.

- 외부 Verifier와의 연결 테스트는 향후 진행할 예정이다.

3.3) 향후 설계 고도화 계획

현재는 체인코드와 WATZ 런타임이 연동되어 .wasm 파일을 secure world에서 안전하게 실행할 수 있으며, SQLite 및 Genann을 통한 성능 측정도 완료된 상태이다. 그러나 전체 end-to-end 아키텍처 관점에서 보면, 실행 결과를 Hyperledger Fabric 네트워크에 전달하고 기록하는 과정은 아직 구현되지 않았다.

이를 위해 향후에는 다음과 같은 방향으로 설계를 고도화할 예정이다:

- Remote Attestation 연동: WATZ 내부에서 생성된 evidence를 외부 Verifier 서버와의 네트워크 통신을 통해 실제로 검증하고, RA 결과를 기반으로 민감 데이터를 안전하게 교환하는 구조를 완성할 예정이다.
- 체인코드 결과 기록 흐름 구현: 체인코드 실행 결과를 secure world로부터 수신한 후, 이를 프록시 체인코드를 거쳐 Hyperledger Fabric ledger에 기록하는 연동 흐름을 설계하고 구현한다. 이때, 결과의 무결성이 RA evidence와 함께 검증되었음을 보장할 수 있도록 한다.
- 연동 오픈소스 기반 확장 계획: 또한, 향후 개발 과정에서는 TZ4Fabric의 오픈소스 구현인 open-source-fabric-optee-chaincode 프로젝트를 기반으로 Fabric 네트워크의 구조와 체인코드 프록시(wrapper), gRPC 기반 chaincode_proxy 모듈을 참고 및 연동할 계획이다. 해당 구현체는 Hyperledger Fabric의 체인코드 실행 요청을 normal world에서 처리하고 secure world로 전달하는 구조를 잘 갖추고 있으며, 이를 WATZ 기반 실행 환경에 맞춰 변형하여 활용할 예정이다.

갱신된 과제 추진 계획

4.1) 개발 일정

월	계획
5월: 논문 분석 및 기초 설계	- 관련 논문 및 기술 문서 분석 - TrustZone, Wasm, OP-TEE, RA 개념 학습 - 착수보고서 및 지도확인서 제출
6월: 실험 환경 구축 및 구조 이해	- MCIMX8M-EVK 보드를 활용한 OP-TEE 환경 구축 - WATZ 빌드 및 AOT 실행 실습 - SQLite, Genann 벤치마크 준비
7월: 성능 측정 및 RA 기능 실험	- AOT 기반 성능 실험 및 실행 시간 측정 - WATZ 내 RA 프로토콜(msg0∼msg3) 분석 - evidence 생성 및 해시 측정 테스트 - 중간보고서 및 중간평가표 제출
8월: 하이퍼레저 패브릭 연동 구현	- open-source-fabric-optee-chaincode 구조 분석 - chaincode_proxy ↔ WATZ 연동 흐름 설계 - 체인코드 결과 전달 및 wrapper 통합 구현
9월: RA 통합 고도화 및 전체 실험	- Verifier 연동 실험 및 보안 채널 테스트 - end-to-end 연동 시나리오 구현 - 최종보고서 및 최종평가표 작성
10월: 최종 발표 준비 및 마무리	- 발표자료 제작 및 시연 영상 정리 - 졸업 과제 발표 심사 진행

구성원별 진척도

이름	진척도
위재준	Hyperledger Fabric 체인코드 구조, 호출 흐름, 파일 구성 및 빌드 체계 분석 완료 WaTZ 이식 전, 기존의 proxy를 통해 실행하는 구조 사전 테스트 수행 중
이준혁	WaTZ 런타임 포팅 및 OP-TEE 기반 RA 기능 테스트 수행 완료 i.MX 보드 기반의 TEE 실행 환경을 구성, SSH 접속, 네트워크 설정, 커널 정보 확인, 실행 환경 안정성 확보 완료
이준태	OP-TEE 기반 Trusted Application(TA) 호출 로직 제거 chaincode_proxy 내부 호출 흐름 변경, Wasm 기반 실행 환경과 연결될 수 있도록 gRPC 통신 구조 수정 중

보고 시점까지의 과제 수행 내용 및 중간 결과

본 과제는 WATZ (WebAssembly Trusted Zone) 기반의 신뢰 실행 환경에서 체인코드를 구동하고, 이를 Hyperledger Fabric과 연동하여 보안성과 실행 무결성을 동시에 확보하는 것을 목표로 한다. 중간보고 시점까지 다음과 같은 성과를 도출하였다.

6.1) WATZ 환경 구성 및 실행 실험

- MCIMX8M-EVK 보드에 OP-TEE 기반의 WATZ 실행 환경을 성공적으로 구성하였다.
- 체인코드는 Go 언어로 작성된 후 WASI 기반 WebAssembly (.wasm)로 컴파일되어, AOT 방식으로 secure world 내에서 실행할 예정이다.
- WAMR 런타임을 통해 바이트코드를 secure memory에 적재하고, sandbox 형태로 안전하게 실행할 예정이다.

6.2) 성능 벤치마크 실험

본 실험은 WATZ 논문에 제시된 Figure 3 - 8의 내용을 기반으로 성능을 재현한 것으로, 실제 측정 데이터를 기반으로 다음과 같이 분석하였다.

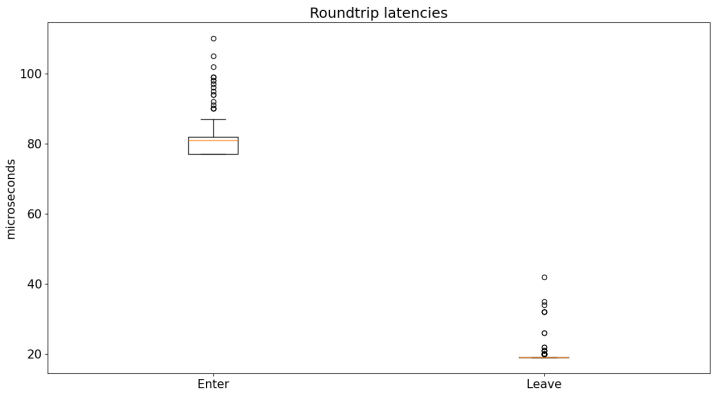

Figure 1: Time retrieval 및 TrustZone 전환 지연 측정 결과

Time retrieval 및 TrustZone world transition latency: TEE 진입/이탈 시의 roundtrip latency를 boxplot으로 시각화하였다. 실험 결과, **Enter 평균은 약 80µs**, **Leave 평균은 약 18µs** 수준으로 나타났으며, 이는 논문 보고치인 86µs / 20µs와 근접한 수치이다. 측정값 분산은 enter 동작에서 상대적으로 컸으며, outlier가 다수 발생하였다. 이는 보드 환경의 I/O 변동성과 관련 있을 수 있다.

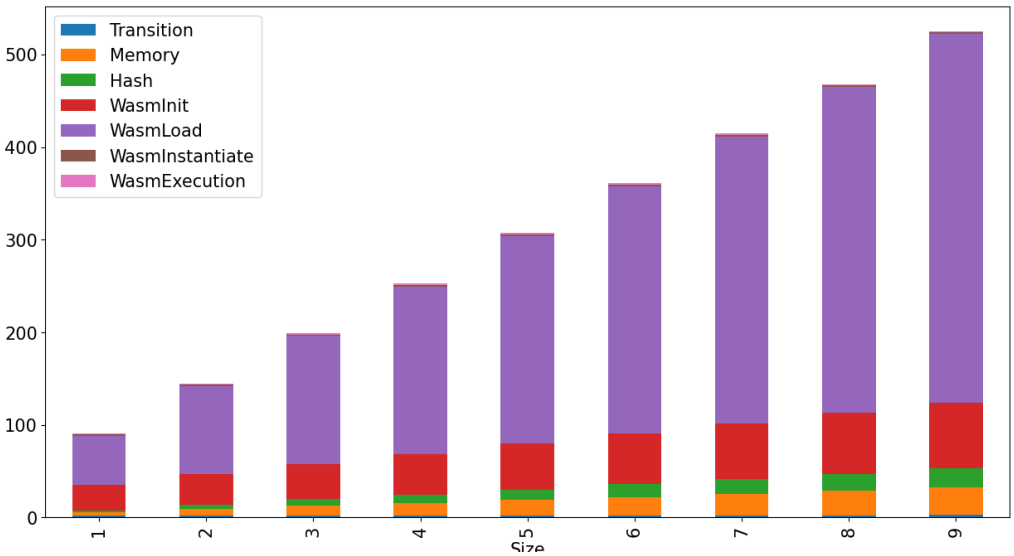

Figure 2: Wasm 프로그램 Startup 단계별 시간 분석

WATZ Startup Breakdown: 애플리케이션 크기(1MB - 9MB)에 따른 startup 단계별 시간(Transition, Memory, Hash, Init, Load, Instantiate, Execution)을 스택 막대그래프로 표현하였다. 실험 결과, **WasmLoad가 전체 시간의 대부분을 차지하며**, 파일 크기에 정비례 증가하였다 (예: 1MB에서 약 55ms, 9MB에서는 약 400ms 이상). 해싱, 인스턴스 생성 등의 오버헤드는 상대적으로 작았으며, 전체 startup 지연은 **최대 약 530ms** 수준으로 확인되었다.

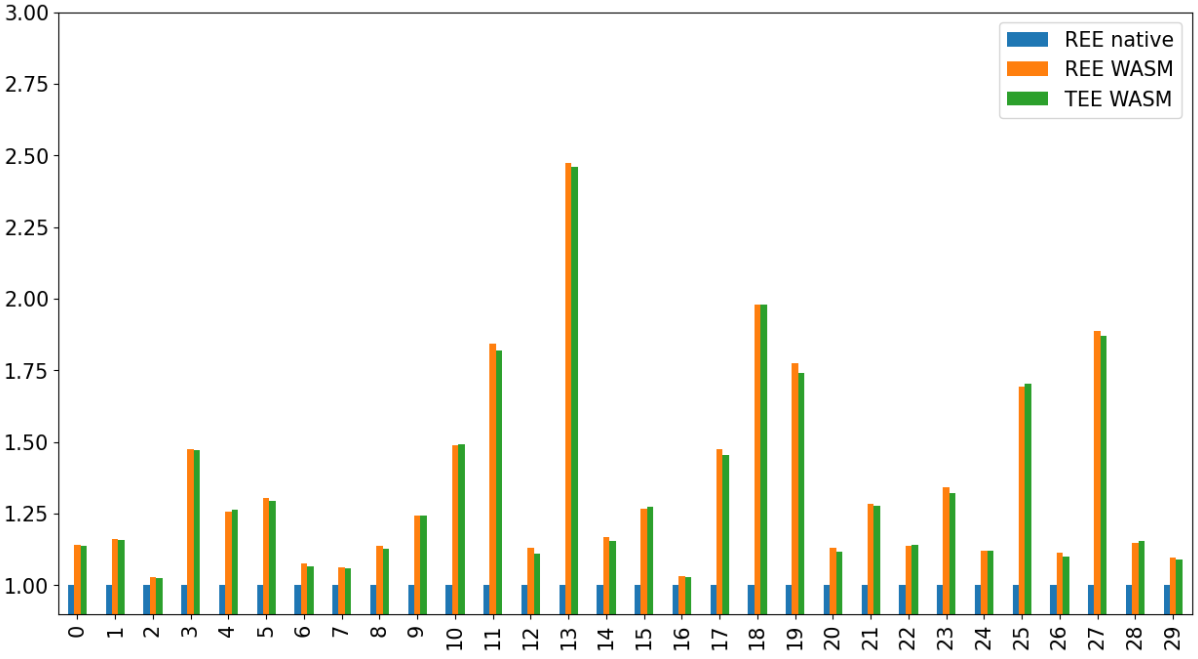

Figure 3: PolyBench/C 워크로드 실행 성능 비교

PolyBench/C Benchmark: 30개 워크로드에 대해 REE Native (=1) 기준으로 REE WASM, TEE WASM 성능을 정규화하여 비교하였다. 실험 결과, 대부분의 워크로드에서 **TEE WASM은 평균 약 1.2∼1.8배** 정도의 실행 시간 오버헤드를 보였으며, 가장 큰 경우는 약 **2.47배** (index 13)로 나타났다. 이는 전반적으로 논문 수치(1.3∼2.4배)와 유사하며, 수치 계산 중심 워크로드에 대해 WATZ의 AOT 실행 성능이 실용적임을 보여준다.

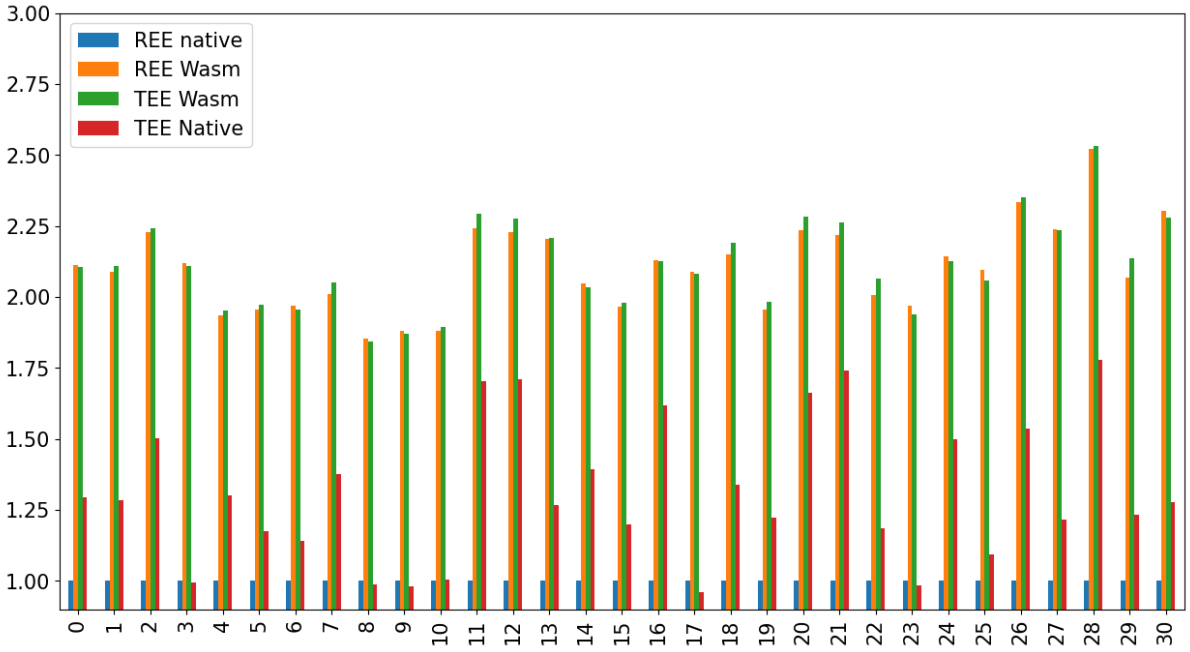

Figure 4: SQLite Speedtest1 성능 비교

SQLite Macro-Benchmark: 30개 이상 쿼리에 대해 REE Native 기준으로 REE Wasm, TEE Wasm, TEE Native까지 비교한 정규화된 실행 시간을 막대그래프로 시각화하였다. 실험 결과, **TEE WASM은 평균 약 2.1∼2.4배의 성능 저하**를 보였으며, 특정 쿼리에서는 **최대 2.6배 수준**까지 관찰되었다. 이는 WASI 호출, secure memory 접근 및 RA hash 계산에 따른 비용으로 판단된다.

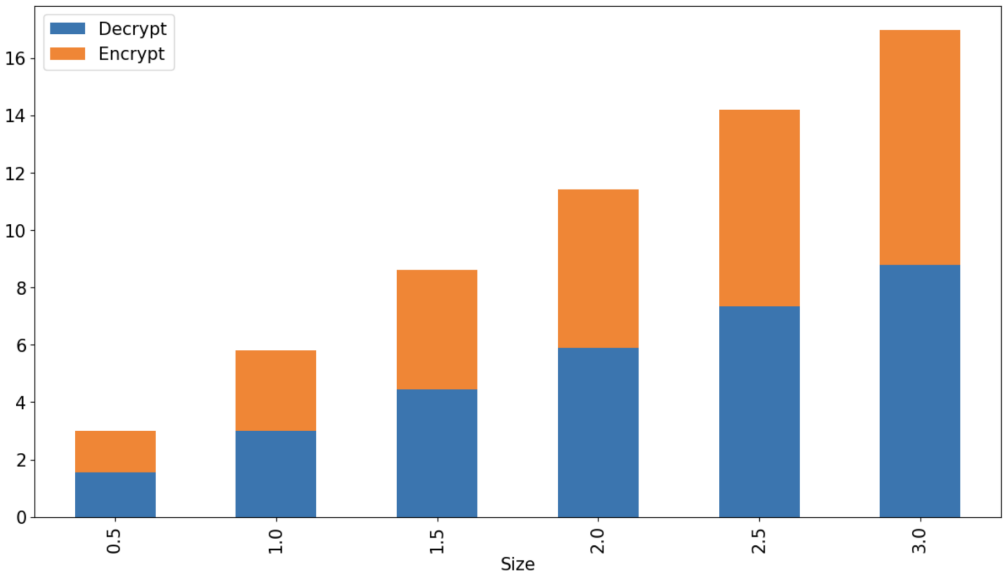

Figure 5: msg3 단계에서 AES-GCM 암복호화 시간

Remote Attestation Micro-Benchmark (AES-GCM): RA 프로토콜의 마지막 단계에서 전송되는 기밀 데이터(Secret Blob)에 대한 AES-GCM 기반 암복호화 성능을 측정하였다. 데이터 크기는 0.5MB에서 3MB까지 증가시키며, 암호화(Encrypt)와 복호화(Decrypt) 각각의 실행 시간을 비교하였다. 실험 결과, **암복호화 시간은 데이터 크기에 비례하여 선형적으로 증가하며**, 3MB 데이터 처리 시 총 17ms 내외로 나타났다. 이는 논문에서 보고된 결과와 거의 동일한 값이다. 따라서, 기밀 데이터 전송 시 암복호화 연산은 전체 RA 비용에서 차지하는 비중이 매우 작아, 실용적 성능을 보장함을 확인할 수 있다.

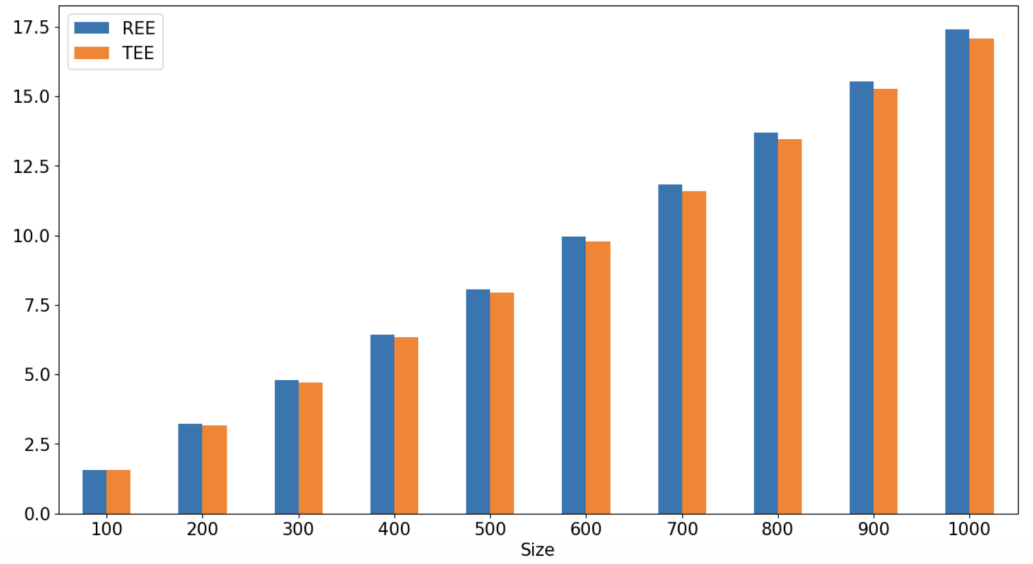

Figure 6: Genann 학습 워크로드 성능 비교 (WAMR vs WaTZ)

Remote Attestation Macro-Benchmark (Genann): 경량 신경망 라이브러리인 Genann을 사용하여, RA 이후 데이터셋을 안전 채널을 통해 수신하고 학습을 수행하는 end-to-end 시나리오를 평가하였다. 실험 구성은 논문과 동일하게 Iris 데이터셋을 기반으로, 입력 크기를 100KB에서 1MB까지 증가시키며 학습 시간을 측정하였다. 결과는 **WAMR(REE)와 WaTZ(TEE) 간의 성능 차이가 거의 없으며**, 오히려 WaTZ에서 약간의 성능 향상(평균 1.4%)이 관찰되었다. 이는 논문에서 보고된 결과와 동일한 경향을 보이며, RA 초기 단계(msg0, msg1, msg2)에서 발생하는 오버헤드는 학습 단계에 미미한 영향을 주는 것으로 확인되었다. 따라서, **TEE 환경에서도 머신러닝 워크로드 실행이 실용적임**을 확인할 수 있다.

6.3) Remote Attestation 기능 실험

- WATZ 내부에 구현된 msg0∼msg3 기반의 RA 프로토콜 구조를 분석하고, 실제 실험을 통해 evidence 생성, 서명, 키 교환 흐름을 확인하였다.
- secure world 내에서는 실행 전 코드 해시를 생성하고, 이를 서명하여 외부 Verifier에 전달할 수 있도록 준비되었다.
- 현재까지는 내부 측정과 서명은 성공적으로 동작하고 있으며, Verifier 연동 및 증명 검증은 9월 중 테스트 예정이다.

6.4) Hyperledger Fabric 연동 구조 설계

- 기존 open-source-fabric-optee-chaincode 구조(wrapper → proxy → TA)를 분석하고, 이를 기반으로 WATZ 환경에 맞는 연동 구조를 설계하였다.
- 체인코드 실행 요청은 wrapper에서 proxy를 거쳐 WATZ로 전달되며, 실행 결과는 다시 wrapper를 통해 Fabric ledger에 저장될 예정이다.
- 현재 chaincode_proxy ↔ WATZ 연동 인터페이스 정의와 구현이 진행 중이며, 8월 말까지 연동 실험을 목표로 하고 있다.

6.5) 중간 성과 요약

항목	**현재까지의 성과**
보드 환경 구성	MCIMX8M-EVK 기반 OP-TEE + WATZ 환경 구축 완료
성능 실험	SQLite, Genann, PolyBench 성능 측정 완료 (AOT 기준)
보안 기능	RA 프로토콜(msg0∼msg3) 분석 및 evidence 생성 성공
연동 설계	Fabric wrapper/proxy ↔ WATZ 연동 구조 설계 및 구현 중

Table 1: 보고 시점까지의 중간 결과 요약

References

[1] Michele Rossi, Christian Platzer, and Frank Kargl, *WaTZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone*, NDSS, 2023.

[2] Linaro, *OP-TEE: Open Portable Trusted Execution Environment*, `https://www.op-tee.org/`

[3] Andreas Haas et al., *Bringing the Web up to Speed with WebAssembly*, ACM PLDI, 2017.

[4] Hyperledger Foundation, *Hyperledger Fabric Documentation*, `https://hyperledger-fabric.readthedocs.io/`