에이전트 경제를 위한 검증 및 프라이버시 레이어

Primus Labs

March 12, 2026

Abstract

사용자를 대신하여 행동하고 거래하며 복잡한 작업을 조율할 수 있는 새로운 유형의 AI 에이전트가 등장하고 있습니다. 그 기회는 매우 큽니다. 자율 에이전트는 디지털 경제 전반에 걸쳐 높은 마찰의 워크플로우를 간단한 위임 작업으로 전환할 수 있습니다. 그러나 이를 가로막는 근본적인 격차가 존재합니다. 에이전트가 약속한 작업을 실제로 수행했는지 사용자가 검증할 수 있는 범용 메커니즘이 존재하지 않으며, 에이전트가 민감한 데이터를 노출하지 않고 연산할 수 있는 방법도 부족합니다. 이러한 격차가 해소되지 않는 한, 에이전트 경제는 저위험·저신뢰 수준의 상호작용을 넘어 확장되기 어렵습니다.

Primus 는 이 문제를 해결합니다. zkTLS 를 통해 에이전트가 외부 서비스나 데이터 소스와 상호작용할 때마다 실제로 어떤 일이 발생했는지에 대한 간결한 암호학적 증명을 생성하여, 자기 보고에 의존한 주장을 독립적으로 검증 가능한 증거로 대체합니다. 또한 완전동형암호(FHE)를 활용함으로써 에이전트는 평문을 노출하지 않고도 민감한 데이터를 처리할 수 있습니다. 이 두 기술의 결합은 에이전트 경제에 부재했던 신뢰 레이어, 즉 증명 가능한 행동과 보호된 데이터를 하나의 실행 스택으로 제공합니다.

Primus 를 통해 사용자는 고가치 작업을 안심하고 위임할 수 있으며, 플랫폼은 에이전트 권한을 보다 안전하게 확장할 수 있습니다. 또한 기존에는 자동화하기에 지나치게 민감했던 서비스를 중심으로 새로운 시장이 열립니다.

1 문제 제기

에이전트 경제가 빠르게 형성되고 있습니다. 자율 AI 에이전트는 이미 여행 예약, 거래 실행, 구독 관리, 조달 조건 협상, 수십 개의 서비스에 걸친 다단계 워크플로우 조정을 수행하고 있습니다. 에이전트의 역량이 성장함에 따라 그들이 통제하는 경제적 영역도 확대되고 있습니다. 에이전트 매개 거래는 향후 몇 년 내 디지털 상거래의 상당 부분을 차지할 것으로 예상됩니다. 그러나 이러한 변화의 기반이 되는 인프라는 소프트웨어가 돈, 데이터, 신원에 대해 실질적인 권한을 가지고 사람을 대신해 행동하는 환경을 전제로 설계되지 않았습니다. 두 가지 구조적 결함이 에이전트 경제가 잠재력에 도달하는 것을 가로막고 있습니다.

에이전트는 자신이 무엇을 했는지 증명할 수 없습니다. 사용자가 에이전트에게 작업을 위임하면, 에이전트는 API 를 호출하고 데이터를 읽고 주문을 실행한 뒤 그 결과를 보고합니다. 그러나 사용자는 에이전트가 제공하는 요약 정보만을 받게 됩니다. 에이전트가 주장한 서비스에 실제로 접근했는지, 보고한 응답을 실제로 수신했는지, 설명한 단계를 수행했는지에 대한 독립적이고 암호학적인 기록이 존재하지 않습니다. 이해관계가 낮은 환경에서는 이러한 격차가 용인될 수 있지만, 고가치 또는 규제된

맥락에서는 치명적인 한계가 됩니다. 포트폴리오를 재조정했다고 주장하는 자산 관리 에이전트나, 정상적인 KYC 확인을 보고하는 준수 에이전트는 단순히 신뢰받는 것을 넘어 검증 가능해야 합니다. 증명 가능한 실행이 없다면 모든 위임은 잠재적인 거래 상대방 위험을 수반하게 되며, 분쟁 해결 비용은 증가하고 플랫폼은 에이전트 권한을 보수적으로 제한할 수밖에 없습니다.

에이전트는 개인 데이터를 안전하게 다룰 수 없습니다. 유용한 에이전트 작업은 거의 항상 계좌 잔고, 건강 기록, 급여 데이터, 독점 가격, 고객 명단, 법적 문서와 같은 민감한 입력을 필요로 합니다. 오늘날 에이전트에 이러한 데이터에 대한 접근 권한을 부여하는 것은 곧 데이터를 평문으로 노출하는 것을 의미합니다. 에이전트 운영자, 호스팅 환경, 그리고 데이터 파이프라인의 모든 중개자는 해당 정보를 관찰, 기록 또는 유출할 수 있습니다. 사용자는 비익명화, 경쟁적 피해, 규제 책임에 노출되며, 조직은 에이전트가 보호된 데이터를 읽는 순간 데이터 보호 의무를 위반할 위험에 직면합니다. 이에 대한 합리적인 대응은 데이터를 제공하지 않는 것이지만, 데이터가 부족한 에이전트는 충분한 가치를 제공하지 못합니다. 그 결과 악순환이 발생합니다. 프라이버시 우려는 위임을 억제하고, 제한된 위임은 에이전트 서비스를 피상적으로 만들며, 이러한 서비스 수준은 신뢰 구축에 필요한 투자를 정당화하지 못합니다.

이 두 가지 결함은 서로를 더욱 악화시킵니다. 자신의 행동을 증명할 수 없고, 처리하는 데이터를 보호할 수 없는 에이전트는 이중으로 신뢰를 잃게 됩니다. 이러한 에이전트에 의존하는 시장은 얕게 형성되고, 수수료는 높게 유지되며, 실제 자금, 규제 데이터 또는 조직 간 협업이 포함된 고가치 활용 사례는 실현되기 어렵습니다. 아래에서는 그 결과를 네 가지 구체적인 위험 범주로 나누어 설명합니다.

- 기밀성 위험. 모든 에이전트 상호작용은 잠재적인 데이터 노출 사건이 될 수 있습니다. 에이전트는 자격 증명, 개인 식별 정보, 재무 데이터, 전략적 신호 등을 일상적으로 처리합니다. 침해된 에이전트, 안전하지 않은 통합, 과도한 권한이 부여된 API 호출로 인한 단 한 번의 유출도 신원 도용, 선행 매매, 규제 처벌, 영업 비밀 손실과 같은 돌이킬 수 없는 피해로 이어질 수 있습니다. 에이전트는 다수의 서비스에 걸쳐 기계적인 속도로 작동하기 때문에, 이러한 기밀성 실패의 파급 범위는 기존의 인간 중심 워크플로우보다 훨씬 큽니다.
- 책임성 격차. 에이전트가 사용자를 대신해 행동할 때 발생하는 분쟁은 사용자, 에이전트 운영자, 서비스 제공자 간의 진실 공방으로 이어집니다. 에이전트의 신원과 사용자의 위임을 암호학적으로 연결한 위변조 방지 실행 로그가 없다면, 이를 판단할 수 있는 중립적인 근거가 존재하지 않습니다. 이는 분쟁 해결 비용을 증가시키고, 고가치 위임을 저해하며, 플랫폼이 세밀한 증거 기반 신뢰 대신 단순한 권한 제한에 의존하도록 만듭니다.
- 규제 마찰. 다양한 관할 구역과 산업을 넘나드는 에이전트는 AML, KYC, GDPR, HIPAA, 산업별 라이선스, 그리고 새로운 AI 거버넌스 규정을 동시에 충족해야 합니다. 이러한 준수 과정에서는 일반적으로 검증자 또는 감사자와 데이터를 공유해야 하지만, 원시 데이터를 공유하는 것 자체가 프라이버시 규정을 위반할 수 있습니다. 이와 같은 역설은 플랫폼을 최소 기준 수준으로 제한하게 만듭니다. 즉, 준수가 입증되기 전까지 에이전트의 기능을 제한하게 되며, 결과적으로 대부분의 기능이 비활성화된 상태로 남게 됩니다.
- 파편화 및 락인(Lock-in). 에이전트가 필요로 하는 데이터는 은행, SaaS 플랫폼, 정부 등록부, 기업 시스템, 개인 기기 등 다양한 곳에 분산되어 있으며, 각각 서로 다른 인증 방식, 데이터 형식, 접근 제어 모델을 가지고 있습니다. 데이터의 진위를

검증하고 이기종 소스 간 신호를 통합하는 보편적이고 프라이버시 보존적인 방법이 없다면, 에이전트는 폐쇄적인 생태계에 갇히게 됩니다. 시스템 간 조율과 같은 고부가가치 시나리오는 여전히 실현되기 어렵습니다.

이러한 문제들은 단순한 기술적 과제가 아니라, 현재 에이전트 경제의 확장을 제한하는 구조적 장벽입니다. 이를 극복하기 위해서는 모든 에이전트 행동을 독립적으로 검증 가능하게 하고, 연산 전 과정에서 민감한 데이터를 암호화 상태로 유지하며, 기존의 규제 및 신원 체계와 호환되는 레이어가 필요합니다. 본 논문의 나머지 부분에서는 이러한 요구를 충족하는 Primus 를 소개합니다.

1.1 데이터 접근의 딜레마

대출 심사, 신원 확인, 재무 감사, 공급망 최적화와 같은 핵심적인 에이전트 작업은 단일 시스템 외부에 존재하는 개인 데이터에 의존합니다. 신용 기록은 신용 평가 기관에, 잔고 정보는 은행에, 신원 문서는 정부 등록부에, 비즈니스 기록은 기업 ERP 시스템 내부에 존재합니다. 프라이버시나 데이터 진위성을 훼손하지 않으면서 이러한 데이터를 에이전트 워크플로우에 연결하는 것은 여전히 해결되지 않은 핵심 과제입니다.

- 기본값으로서의 노출. 현재의 통합 방식에서는 에이전트가 동작하기 전에 데이터가 소스에서 평문 형태로 외부로 전달됩니다. 소스 API 에서 에이전트 실행 환경, 그리고 다운스트림 소비자에 이르기까지 모든 단계가 잠재적인 유출 지점이 됩니다. 프라이버시 보존 아키텍처는 이러한 기본 전제를 뒤집어야 합니다. 데이터는 소스에서 암호화되거나 약속(commitment)된 상태로 유지되어야 하며, 에이전트는 이를 직접 관찰하지 않고도 연산할 수 있어야 합니다.
- 신뢰 없는 진위성. 위조되거나 오래된 데이터에 기반해 의사결정을 수행하는 에이전트는, 실제 데이터를 유출하는 경우만큼이나 심각한 피해를 초래할 수 있습니다. 그러나 대부분의 API 는 해당 응답이 진짜이며 최신 상태이고 변경되지 않았다는 것을 증명하는 암호학적 근거를 제공하지 않습니다. 에이전트와 그 거래 상대방은 에이전트의 자기 보고가 아니라, 데이터 출처 자체에 대한 위변조 방지 가능한 검증을 필요로 합니다.
- 대규모 파편화. 유용한 에이전트 의사결정은 종종 5 개, 10 개, 혹은 그 이상의 이기종 데이터 소스로부터 수집된 신호를 결합해야 합니다. 각 소스는 서로 다른 스키마, 인증 방식, 속도 제한, 서비스 약관을 가지고 있습니다. 표준화되고 프라이버시를 보존하는 커넥터가 없다면, 모든 새로운 통합은 맞춤형 엔지니어링 작업이 되며, 에이전트가 가장 큰 가치를 창출하는 소스 간 조합은 여전히 높은 비용 구조에 머물게 됩니다.
- 동의 및 생명주기 제어. 사용자와 데이터 소유자는 에이전트가 어떤 데이터에 대해, 얼마나 오랫동안, 어떤 조건 하에 접근할 수 있는지에 대해 세밀한 통제권을 유지해야 합니다. 권한 철회는 즉각적으로 이루어져야 하며, 감사 추적은 검증 가능해야 합니다. 또한 데이터 최소화 원칙은 단순한 정책 수준을 넘어 인프라 차원에서 강제될 수 있어야 합니다.

이러한 제약 조건들은 실용적인 신뢰 및 프라이버시 레이어가 충족해야 할 요구 사항을 정의합니다. 즉, 선택적 공개, 증명 가능한 데이터 진위성, 기밀 연산, 구성 가능한 소스 간 접근, 그리고 강제 가능한 동의 거버넌스입니다. Primus 는 이러한 기능을 모두 구현하며, 다음 섹션에서 설명하듯 에이전트 경제를 위한 일관된 실행 스택으로 이를 통합합니다.

2 해결책

Primus 는 두 가지 첨단 암호 기술, 즉 데이터 및 API 호출에 대해 검증 가능하면서도 프라이버시를 보존하는 접근을 제공하는 영지식 전송 계층 보안(zkTLS)과, 기밀 연산 및 저장을 위한 완전동형암호(FHE)를 통합함으로써 에이전트 경제를 위한 포괄적인 신뢰·검증·프라이버시 계층을 제공합니다. 이 두 기술을 통해 에이전트는 자신이 수행한 작업을 증명할 수 있으며, 자기 보고식 주장에 의존하지 않고 데이터와 서비스에 접근하고, 민감한 입력을 노출하지 않은 상태에서 연산을 수행할 수 있습니다. 본 섹션에서는 이러한 기술을 공식적으로 설명하고, 관련 연구를 검토하며, Primus 의 혁신과 아키텍처 프레임워크를 상세히 소개합니다.

2.1 영지식 전송 계층 보안 (zkTLS)

전송 계층 보안(TLS) [1,2]은 인터넷에서 클라이언트와 서버 간 통신을 보호하기 위한 대표적인 프로토콜입니다. TLS 는 클라이언트(C)와 서버(S) 간 세션에 대해 기밀성과 무결성을 제공합니다. 그러나 TLS 는 애플리케이션 계층 페이로드에 대한 제 3 자 검증을 생성하도록 설계되어 있지 않습니다. 즉, 진위성은 세션의 양 끝점 사이에서만 보장됩니다.

클라이언트가 이후 TLS 로 보호된 데이터를 제 3 자 애플리케이션에 제시하는 경우, 악의적인 클라이언트는 외형상 유효해 보이는 트랜스크립트를 위조할 수 있습니다. 이는 TLS 가 애플리케이션 수준의 데이터 사용을 외부 검증과 결합하지 않기 때문입니다. 이러한 간극은 클라이언트와 서버가 모두 프로토콜을 성실히 따른다는 암묵적인 가정에서 비롯되며, 클라이언트를 신뢰할 수 없는 환경에서는 더 이상 성립하지 않습니다.

일반적인 웹 기반 완화 방법은 사용자가 사이트 간 데이터 공유를 중개하도록 요구합니다(예: OAuth 기반 흐름 또는 직접적인 사이트 간 인증). 이러한 접근 방식은 실무적으로 사용되고 있으나, 본 문서의 설정에서는 두 가지 한계를 가집니다. 첫째, 종종 필요한 수준을 초과하는 정보 공개를 요구합니다(예: 특정 조건을 증명하기 위해 전체 신용 점수를 공개). 둘째, 추가적인 웹 인프라 또는 프로토콜 변경이 필요하여 배포 복잡성을 증가시킵니다. 이러한 제약으로 인해 해당 방식은 Web3 환경에서 검증 가능하면서도 프라이버시를 보존하는 Web2 데이터 활용에 적합하지 않습니다.

웹 증명(web proofs)으로도 알려진 zkTLS 는, TLS 로 보호된 웹 데이터에 대해 기밀성을 유지하면서 외부 검증자에게 제시 가능한 암호학적 증명을 생성하는 프로토콜입니다. 이 구조에서 TLS 서버로부터 데이터를 수신한 클라이언트는, 관찰된 평문이 해당 서버에서 생성되었으며 기록된 TLS 트랜스크립트와 일치함을 증명할 수 있습니다. 이 과정에서 관련 없는 비밀 정보는 노출되지 않습니다.

실제 배포에서는 증명 생성을 보조하거나 세션을 관찰하는 인증자(또는 공증인)가 도입됩니다. 인증자는 증명 과정에서 의도적으로 공개된 정보만을 알 수 있으며, 서버 측에서는 별도의 프로토콜 변경이 필요하지 않습니다. DECO [3]는 기록된 TLS 트랜스크립트로부터 증명을 생성하는 다양한 접근 방식을 제시하며 zkTLS 연구의 출발점을 마련했습니다. 초기 구조는 상당한 오버헤드를 수반했으나, 이후 연구를 통해 이를 개선하기 위한 핵심 기법들이 지속적으로 발전해 왔습니다.

2.1.1 TLS 개요

zkTLS 를 이해하기 위해 필요한 최소한의 TLS 개념을 간략히 정리합니다. TLS 는 클라이언트와 서버 사이에서 동작하며, (1) 키 재료를 설정하는 암호화 핸드셰이크(Handshake) 단계와, (2) AEAD 암호를 사용하여 애플리케이션 데이터를 보호하는 레코드(Record) 계층으로 구성됩니다.

핸드셰이크 단계 핸드셰이크는 인증된 공유 비밀을 생성하는 과정입니다. 비공식적으로 표현하면, 키 교환을 통해 프리마스터 시크릿(pms)이 생성됩니다. 트래픽 암호화 및 인증에 사용되는 세션 비밀 정보는 프로토콜의 키 유도 함수(KDF)를 통해 pms 와 공개된 핸드셰이크 트랜스크립트(난수, 논스, 트랜스크립트 해시)로부터 유도됩니다. TLS 1.3 에서는 이 KDF 가 HKDF(명시적인 "추출(extract)" 및 "확장(expand)" 단계와 트랜스크립트 바인딩을 포함)로 구현됩니다. TLS 1.2 에서는 RFC 5246 에 정의된 PRF 를 사용합니다. 이를 추상적으로 표현하면 다음과 같이 나타낼 수 있습니다.

$$\text{secrets} \leftarrow \text{KDF}(\text{pms}, \text{transcript}),$$

여기서 트랜스크립트(transcript)는 유도된 키에 결합되는 공개 핸드셰이크 재료를 나타냅니다. 이러한 결합은 TLS 가 세션 트랜스크립트를 인증하는 이유입니다. 키는 pms 와 트랜스크립트의 결정론적 함수이기 때문입니다.

레코드 단계에서 트래픽 키가 유도되면, 레코드 계층은 AES-GCM 과 같은 AEAD 를 사용하여 애플리케이션 데이터를 보호합니다. K 를 애플리케이션 트래픽 키라 하고, $\mathrm{m}_0,\ldots,\mathrm{m}_\ell$ 을 평문 블록이라 할 때, AES-GCM 은 인증 태그 τ 와 함께 암호문 블록 c_i 를 생성합니다.

$$\mathrm{c}_i = \mathrm{AES\_CTR}\left(\mathrm{K}, \mathrm{ctr}_i\right) \oplus \mathrm{m}_i,\ i = 0, \ldots, \ell,$$

그리고

$$\tau = \mathrm{GCM\_Tag}\left(\mathrm{K}, \mathrm{c}_{0:\ell}, \mathrm{AD}\right),$$

여기서 AD 는 관련 데이터(예: 레코드 헤더)를 나타내고, ctr_i 는 레코드별 논스/IV 를 의미합니다. 올바른 복호화를 위해서는 평문을 복구하기 전에 인증 태그 τ 의 검증이 필요합니다. 많은 zkTLS 구조는 AES-GCM 의 비확정적(non-committing) 특성을 고려해야 하므로, 키 유도 일관성을 증명하거나 증명을 구성할 때 공개 평문 구조(패딩, 상태 라인 등)를 활용합니다. AEAD 에 대한 자세한 내용은 RFC 5288 을 참조하십시오.

2.1.2 zkTLS 모드

zkTLS 프로토콜은 MPC 모드와 프록시(Proxy) 모드라는 두 가지 배포 방식을 사용합니다. 두 모드 모두 TLS 로 보호된 데이터에 대해 검증 가능한 인증을 제공하지만, 프라이버시, 성능, 네트워크 신뢰 측면에서 서로 다른 절충을 가집니다. 아래에서는 각 모드에 대한 간략한 개요와 함께 설계 선택을 돕기 위한 장단점을 요약합니다.

MPC 모드 MPC 모드에서는 클라이언트와 인증자가 2 자 간 연산(2PC)을 통해 TLS 핸드셰이크와 레코드 계층의 일부를 공동으로 시뮬레이션합니다. 목표는 클라이언트의 비밀을 노출하지 않으면서 인증을 생성하는 것입니다. 클라이언트는 자격 증명, 쿠키,

개인 키와 같은 비밀 정보를 보호하며, 인증자는 관찰된 암호문이 주장된 평문과 일치한다는 암호학적 확신을 얻습니다. 개요는 그림 1 및 아래 설명을 참조하십시오.

1. 핸드셰이크에서의 2PC: 클라이언트와 인증자는 프리마스터 시크릿과 공개 핸드셰이크 트랜스크립트에 대해 KDF 를 평가하는 2PC 프로토콜을 실행합니다. 어느 쪽도 전체 유도 키를 알 수 없으며, 대신 세션 비밀의 비밀 공유(또는 마스킹된 값)를 획득합니다.
2. TLS 요청에서의 2PC: 요청 평문이나 키가 노출되는 것을 방지하기 위해, 클라이언트와 인증자는 2PC 환경에서 나가는 요청에 대한 대칭 암호화 및 인증 태그 계산(AES-GCM)을 공동 수행합니다. 이를 통해 클라이언트는 키를 공유한 상태로 데이터 소스로 전송할 암호문을 생성할 수 있습니다.
3. 서버 응답 릴레이: 클라이언트는 데이터 소스로부터 응답 암호문을 수신한 후, 키를 재구성하지 않은 상태에서 이를(또는 트랜스크립트) 검증을 위해 인증자에게 전달합니다.
4. 키 재구성 및 증명: 인증자가 응답 암호문을 검증한 뒤 자신의 키 공유를 클라이언트에 전달합니다. 클라이언트는 전체 세션 키를 재구성하고, 기록된 암호문이 주장된 평문으로 복호화됨과 해당 AES 키가 핸드셰이크 2PC 에서 사용된 KDF 출력과 일치함을 보이는 대화형 증명(예: QuickSilver iZK)을 생성합니다.
5. 인증서 발행: 증명 검증이 완료되면, 인증자는 주장된 평문과 TLS 트랜스크립트를 결합한 서명된 인증서를 발행하여 외부 사용(예: 온체인 제출)에 활용할 수 있도록 합니다.

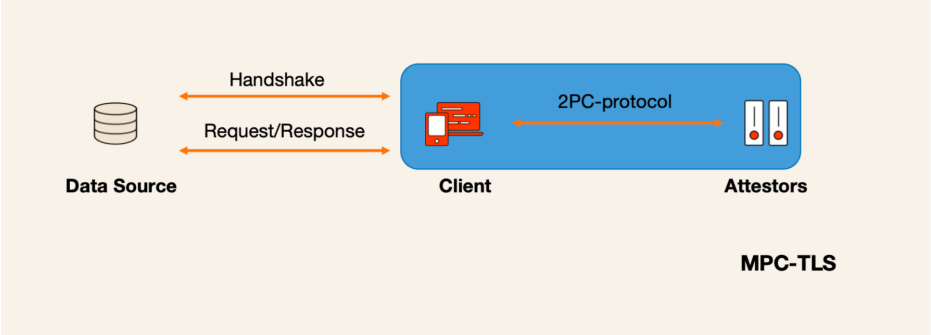

Figure 1: MPC 모드 상위 수준 흐름

주요 포인트: 다음 항목들은 MPC 모드의 메커니즘, 프라이버시 보장, 성능 비용, 네트워크 견고성 및 운영 완화 조치를 요약합니다.

- 메커니즘: 클라이언트와 인증자는 2PC 를 공동으로 실행하여 KDF 와 요청 및 응답을 형성하고 검증하는 데 필요한 대칭 기본 요소(AES-GCM)를 평가합니다. 참여자들은 일반적으로 프로토콜 실행 중 세션 키를 재구성하는 대신, 비밀 공유 또는 마스킹된 값을 유지합니다.
- 프라이버시 보장: 클라이언트의 장기 비밀 정보(자격 증명, 쿠키, 개인 키)는 비공개로 유지됩니다. 인증자는 2PC 출력과 최종 인증서를 통해 공개된 최소한의 정보만을 알 수 있습니다.

- 성능 비용: KDF 및 AEAD 를 위한 2PC 실행은 상당한 CPU, 메모리 및 대역폭 오버헤드를 유발합니다. 준정직 가블드 서킷, OT 확장, 가블-덴-프루브 파이프라인 등의 최적화를 통해 비용을 줄일 수 있으나, 기본 TLS 대비 여전히 추가 비용이 발생합니다.
- 네트워크 견고성: MPC 모드는 인증자의 네트워크 위치에 대한 의존도를 낮춥니다. 인증자가 인라인 프록시로 동작할 필요가 없으므로, 네트워크 수준 조작에 따른 공격 표면이 감소합니다.

프록시 모드 프록시 모드에서는 인증자가 네트워크 경로(인라인 또는 신뢰할 수 있는 기록기)에 위치하여 실제 TLS 핸드셰이크와 암호문을 기록하고, 이후 해당 기록에 대한 진술을 검증할 수 있습니다. 개요는 그림 2 를 참조하십시오. 이 모드는 MPC 모드에 비해 훨씬 낮은 런타임 암호화 비용을 제공하는 대신, 네트워크 신뢰에 대한 요구가 높아집니다.

1. 세션 캡처: 클라이언트가 데이터 소스와 TLS 세션을 설정하는 동안, 인증자는 핸드셰이크 및 이후 레코드 암호문을 프록시하거나 수동으로 기록하고, 트랜스크립트 해시를 계산한 뒤 서명된 타임스탬프를 부여합니다.
2. 클라이언트 증명 구성: 클라이언트는 응답 암호문을 수신한 후, 보관된 트랜스크립트를 기반으로 주장된 평문 속성(KDF→AES 일관성 또는 공개 패딩 관련 진술)을 입증하는 영지식 증명을 구성합니다.
3. 검증 및 인증: 클라이언트는 생성한 증명을 인증자에게 전달합니다. 인증자는 보관된 트랜스크립트를 기준으로 이를 검증하고, 성공 시 온체인 또는 오프체인 검증자가 사용할 수 있는 서명된 인증서를 발행합니다.
4. 공개(선택 사항): 투명성 강화를 위해 인증자 또는 클라이언트는 인증서 자체 또는 그에 대한 간결한 약속(예: 머클 루트, 트랜스크립트 해시)을 공공 장부에 게시할 수 있습니다.

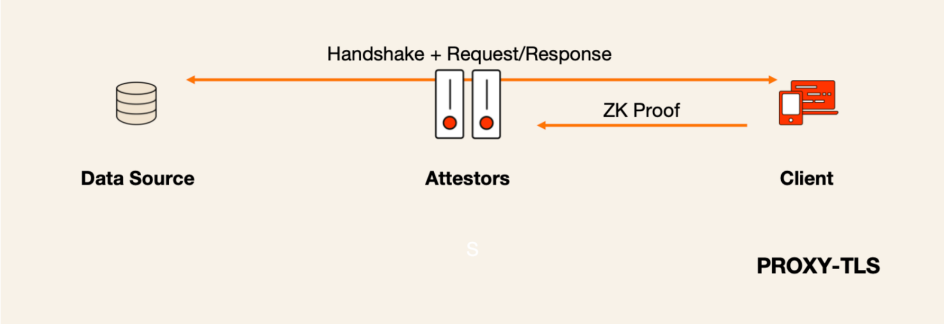

Figure 2: 프록시 모드 상위 수준 흐름

클라이언트는 다음 두 가지 상호 보완적인 증명 전략 중 하나를 사용하여 주장된 평문의 정확성을 입증할 수 있습니다.

- 옵션 1: KDF 및 AES 일관성. 클라이언트는 기록된 암호문을 생성하는 데 사용된 AES 애플리케이션 키가 프리마스터 시크릿 pms 와 공개 핸드셰이크 트랜스크립트에 적용된 프로토콜 KDF 의 결과임을, 그리고 해당 키로 기록된 암호문을 복호화하면 주장된 평문이 도출됨을 증명합니다.

- 옵션 2: 공개 패딩 및 AES. 응답에 알려진 고정된 공개 구조(예: 표준 HTTP 상태 줄 또는 기타 예측 가능한 헤더)가 포함된 경우, 클라이언트는 선택된 암호문 블록이 해당 공개 평문에 대응하며, 나머지 블록은 단일 AES 키 하에서 비밀 페이로드를 암호화함을 증명합니다.

두 접근 방식 모두 암호문을 하나의 입증된 키에 결합함으로써 AES-GCM 의 비확정적 동작을 완화합니다. 옵션 1 은 KDF 를 통해 키를 핸드셰이크에 암호학적으로 결합함으로써 더 강력한 보장을 제공합니다. 옵션 2 는 공개 구조가 충분한 응답에 대해 더 효율적이지만, 해당 구조가 없는 경우에는 추가적인 주의가 필요합니다. 어떤 경우에도 증명은 레코드 블록 전반에 걸쳐 키 일관성을 보장해야 하며, 필요 시 해당 키가 핸드셰이크 과정에서 도출된 KDF 출력과 일치함을 보장해야 합니다.

이 모드는 다음과 같은 속성과 절충점을 가집니다.

- 메커니즘: 인증자는 트래픽을 전달하거나 관찰하고, 핸드셰이크 트랜스크립트, 관련 레코드 암호문 및 메타데이터를 저장하며, 주장된 평문을 보관된 암호문과 연결하는 클라이언트 제출 영지식 증명을 검증합니다.
- 성능: 보관된 트랜스크립트에 대한 증명 생성은 일반적으로 핸드셰이크마다 전체 2PC 를 실행하는 것보다 훨씬 효율적입니다. 증명 비용은 선택한 진술(KDF→AES vs 공개 패딩)과 사용된 증명 시스템에 따라 달라집니다.
- 신뢰 가정 및 완화: 프록시 모드는 인증자가 실제 데이터 소스에 대한 신뢰 가능한 경로를 확보하고 있음을 전제로 합니다. 인증자 오동작에 대한 완화 조치로는 TLS 세션에 대한 채널 바인딩, 인증서 투명성 검증, OCSP/CT 증거, 보관물에 대한 서명된 타임스탬프, 트랜스크립트 해시에 대한 선택적 제 3 자 스냅샷 등이 포함됩니다.

2.1.3 Primus zkTLS 프로토콜

Primus 는 MPC 모드와 프록시 모드 모두에 적용 가능한 garble-then-prove [4] 기반의 고도로 최적화된 zkTLS 파이프라인을 제안합니다. Primus 는 QuickSilver [5]의 최신 VOLE 기법을 활용하여 전처리 비용을 상각하고, 온라인 대칭 연산 및 통신 비용을 크게 줄입니다. 이를 통해 Primus 는 증명 런타임, 피크 메모리 사용량, 그리고 증명자-검증자 간 통신 측면에서 실질적인 성능 향상을 달성합니다.

- 가블-덴-프루브(Garble-then-prove) 워크플로우: 완전한 악의적 2PC 프로토콜을 그대로 적용하는 대신, 에뮬레이션 과정에서 TLS 고유의 의미론을 활용하여 적대적 요구 사항을 완화합니다. 부패한 검증자는 에뮬레이션 중 세션 키를 알 수 없으며, 부패한 증명자는 세션 종료 이후에도 제한적이고 무해한 정보만을 알 수 있습니다. Primus 는 증명자의 모든 일탈이 검증 단계에서 탐지될 수 있도록 설계되었습니다. 에뮬레이션 이후 증명자는 필요한 공유를 공개하고, 영지식 증명(대화형 증명, 예: QuickSilver)을 통해 가블드 실행이 약속된 입력과 공개 트랜스크립트에 부합하는 출력을 생성했음을 입증합니다. 이 설계는 zkTLS 의 무결성을 유지하면서도 고비용의 악의적 2PC 메커니즘을 제거합니다.
- TLS 전용 최적화: 공격자의 이점을 증가시키지 않는 범위 내에서 공개 가능한 중간 값을 선별하여 회로 크기와 비용을 줄입니다. 이를 통해 많은 경우 핸드셰이크 회로 크기를 2 배 이상 감소시킬 수 있습니다. 또한 메인 불리언 회로에서 GMAC/AES 태그 계산을 제거합니다. 오류 허용 의무 선형 평가(OLEe)를 활용하여 GMAC 에 필요한 무작위 거듭제곱의 가법적 공유를

생성함으로써, 보안에 영향을 주지 않는 수준의 미미한 정보 누출만 허용하면서 GMAC 비용을 수십 배 절감합니다.

- OLE 를 통한 GMAC 최적화: GMAC 전체를 불리언 회로 내에서 구현하는 대신, Primus 는 의무 선형 평가(OLE)를 사용하여 GMAC/AES-GCM 에 필요한 선형 항의 가법적 공유를 생성합니다. 태그 계산을 선형 연산으로 처리함으로써, 비용이 큰 와이어 단위 불리언 연산을 상대적으로 저렴한 필드 연산으로 대체하여 GMAC 비용을 100 배 이상 절감할 수 있습니다.
- 약속 변환(Commitment conversion): 가블-덴-프루브 실행을 zkSNARK 기반 검증과 연결하기 위해, Primus 는 실행 중 생성된 정보 이론적 MAC(IT-MAC)을 SNARK 친화적인 가법적 약속으로 변환합니다. 먼저 불리언 IT-MAC 을 큰 필드 상의 산술 IT-MAC 으로 매핑한 뒤, 이를 SNARK 친화적인 가법적 약속으로 변환합니다. 이 두 단계는 모두 효율적이며, 악의적 환경에서도 오버헤드를 반정직(semi-honest) 수준에 가깝게 유지합니다. 이 파이프라인은 가블-덴-프루브와 필요 시 수행되는 SNARK 검증을 결합한 하이브리드 증명을 가능하게 합니다.
- 실증적 개선: 프로토콜을 종단 간 구현하고 x86, ARM, 브라우저(WASM) 환경에서 평가했습니다. 해당 구현은 DECO 대비 한 차원 높은 성능을 보이며(보고된 실험 기준 통신량 약 14 배 개선, 런타임 약 7.5 배~15 배 단축), 실제 API 페이로드에 대해 낮고 예측 가능한 증명자 지연 시간(온라인 시간은 일반적으로 초 단위)을 달성합니다. 또한 응답 크기에 따라 안정적으로 확장됩니다. 이러한 결과는 브라우저 중심 배포에서 garble-then-prove 와 QuickSilver 를 채택하고, 고위험 주장 [4,6]에 대해 선택적 키 일관성 검증을 유지한 설계 선택을 뒷받침합니다.

2.1.4 벤치마크

우리는 Primus, TLSNotary, Reclaim, Pluto 등 여러 오픈소스 zkTLS 구현을 벤치마킹합니다 [7,8,9,10]. 벤치마크는 우리의 오픈 프레임워크(참조 [11])를 사용하여 x86, ARM 및 브라우저(WASM) 플랫폼에서 수집되었습니다. 이 프레임워크는 구성 가능한 네트워크 매개변수와 워크로드 크기를 지원합니다.

대표적인 웹 액세스(쿠키를 전송하는 브라우저 클라이언트)를 시뮬레이션하기 위해 요청 크기를 1KB 또는 2KB 로 설정하고, 응답 크기는 16B 에서 2KB 까지 다양하게 구성했습니다. 실제 성능은 네트워크 및 플랫폼 환경에 따라 달라질 수 있습니다.

아래 보고된 실행 환경은 다음과 같습니다: 대역폭 200Mbps, 지연 시간 10ms, 요청 크기 2KB, Ubuntu 22.04, 8 vCPUs @ 3.10 GHz, 32GB RAM. 일부 대표 구현에 대한 결과는 다음과 같습니다.

MPC 모드: TLS 1.2용 MPC 모드를 구현한 Primus와 TLSNotary를 비교합니다.

응답 크기 (바이트)	업로드 크기 (KB)	다운로드 크기 (KB)	네이티브 (x86) 런타임 (초)	네이티브 (arm) 런타임 (초)	브라우저 런타임 (초)	메모리 (MB)
16	34,888	1,141	2.56	2.64	5.05	146
256	34,901	1,141	2.57	2.62	5.09	146
1024	34,930	1,141	2.57	2.64	5.04	146
2048	34,971	1,141	2.58	2.65	5.11	146

Table 1: Primus (MPC 모드) 벤치마크 결과.

응답 크기 (바이트)	업로드 크기 (KB)	다운로드 크기 (KB)	네이티브 (x86) 런타임 (초)	네이티브 (arm) 런타임 (초)	브라우저 런타임 (초)	메모리 (MB)
16	61,089	62,144	20	32	47	147
256	61,121	65,309	20	33	48	148
1024	61,222	75,437	21	35	49	148
2048	61,356	88,940	23	39	53	148

Table 2: TLSNotary (MPC 모드) 벤치마크 결과.

1. Primus 와 TLSNotary 는 모두 TLS 1.2 용 MPC 모드 프로토콜을 구현했으며, 두 구현 모두 다양한 플랫폼에서 메모리 효율적입니다. 또한 두 구현 모두 브라우저 런타임이 네이티브 실행 대비 약 2 배 느립니다.
2. 이러한 조건에서 Primus 는 다양한 플랫폼에서 총 런타임 기준 TLSNotary 보다 약 10 배 빠릅니다. Primus 는 응답 크기와 무관하게 안정적인 런타임을 유지하는 반면, TLSNotary 는 응답 크기가 증가함에 따라 다운로드 크기와 런타임이 함께 증가합니다(증가 폭은 상대적으로 작음).
3. 이러한 벤치마크 결과의 주요 원인은 Primus 가 QuickSilver, 최적화된 회로, 효율적인 가블드 서킷(GC) 구현을 사용하기 때문으로 보입니다. 특히 TLSNotary 팀 역시 Primus 팀의 지원을 받아 QuickSilver 를 통합하고 있습니다. 대략적인 추정에 따르면 QuickSilver 를 적용할 경우 다운로드 크기는 크게 감소하고, 계산 오버헤드는 약 3GC(가블드 서킷 3 회 실행)에서 2GC 수준으로 줄어듭니다.

프록시 모드: 프록시 모드 벤치마크에서는 요청 데이터를 증명하지 않도록 요청 크기를 0 으로 설정했습니다. 이는 응답 크기에 따른 비용 비교에 초점을 맞추기 위함입니다. Primus(프록시 모드, TLS 1.2)를 Reclaim(프록시 모드, TLS 1.2/1.3) 및 Pluto(프록시 모드, TLS 1.3)와 비교합니다.

응답 크기 (바이트)	업로드 크기 (KB)	다운로드 크기 (KB)	네이티브 (x86) 런타임 (초)	네이티브 (arm) 런타임 (초)	브라우저 런타임 (초)	메모리 (MB)
16	375	788	0.45	0.48	2.31	71
256	383	788	0.46	0.47	2.34	75
1024	416	788	0.47	0.49	2.48	91
2048	461	788	0.48	0.52	2.66	111

Table 3: Primus (프록시 모드, TLS 1.2) 벤치마크 결과.

1. Primus 는 TLS 1.2 용 프록시 모드에서 옵션 1 을 구현한 반면, Pluto 는 ChaCha20-Poly1305 암호 스위트를 사용하여 TLS 1.3 용 옵션 1 을 채택했습니다. 이는 주로 ChaCha20 이 그들의 IVC 시스템과 더 높은 호환성을 가지기 때문입니다. 반면 Reclaim 은 TLS 1.2 와 TLS 1.3 모두에서 옵션 2 를 구현합니다.
2. Reclaim 과 Pluto 의 총 통신 크기는 거의 동일하며, zkSNARK 를 사용한다는 점을 고려할 때 예상대로 Primus 대비 최대 18 배 작습니다.

응답 크기 (바이트)	업로드 크기 (1.2/1.3 KB)	다운로드 크기 (1.2/1.3 KB)	네이티브 (x86) 런타임 (1.2/1.3 초)	네이티브 (arm) 런타임 (1.2/1.3 초)	브라우저 런타임 (1.2/1.3 초)	메모리 (MB)
16	32/23	31/22	4.09/3.62	8.79/8.16	13.88/13.05	356
256	34/24	32/23	5.09/4.76	11.06/10.64	18.76/21.43	313
1024	38/29	38/29	8.53/8.43	18.71/19.10	36.63/49.21	298
2048	44/35	44/36	12.92/13.38	28.61/29.83	60.22/86.21	306

Table 4: Reclaim (프록시 모드, TLS 1.2/1.3) 벤치마크 결과.

응답 크기 (바이트)	업로드 크기 (KB)	다운로드 크기 (KB)	네이티브 (x86) 런타임 (초)	네이티브 (arm) 런타임 (초)	브라우저 런타임 (초)	메모리 (MB)
16	62	14	41	56	193	1395
256	62	14	41	57	206	1395
1024	63	15	58	89	293	1391
2048	65	16	76	119	390	1412

Table 5: Pluto (프록시 모드, TLS 1.3) 벤치마크 결과.

3. Primus 의 총 런타임은 다양한 플랫폼에서 Reclaim 보다 약 5 배에서 30 배 빠릅니다. 또한 Primus 는 메모리를 약 3 배에서 5 배 적게 사용하며, 두 구현 모두 경량이고 대부분의 애플리케이션에 충분합니다. 반면 Pluto 는 Primus 와 Reclaim 보다 현저히 느리며, Primus 대비 90 배 이상 느리고 메모리 사용량도 상당히 높습니다. 이는 계산 집약적인 IVC 를 사용하기 때문입니다.
4. 이러한 벤치마크 결과의 주요 원인은 Primus 가 QuickSilver 를 사용하는 반면, Reclaim 은 Groth16 에, Pluto 는 Nova 에 의존하기 때문으로 보입니다. 두 방식 모두 계산 집약적입니다. 이는 Primus 의 런타임이 응답 크기와 무관하게 비교적 안정적인 반면, Reclaim 과 Pluto 의 런타임은 응답 크기에 매우 민감한 이유를 설명합니다. 일부 저장소는 지속적으로 업데이트되므로 성능 지표는 시간에 따라 변할 수 있습니다.

해당 결과는 내부 벤치마크를 기반으로 하며, 실제 구현 방식 및 운영 환경에 따라 달라질 수 있습니다.

2.2 완전 동형 암호 (FHE)

완전 동형 암호(FHE)는 복호화 없이 암호화된 데이터에 대해 연산을 수행할 수 있도록 하는 암호화 기술입니다. FHE 스키마에서는 데이터가 암호문 형태로 암호화되며, 이러한 암호문에 대해 직접 수학적 연산을 수행할 수 있습니다. 이러한 연산의 결과는 복호화 시 원래 평문 데이터에 대해 동일한 연산을 수행한 결과와 일치합니다. 이 속성은 연산 과정 전반에 걸쳐 민감한 데이터의 기밀성을 유지할 수 있게 하므로, 데이터 프라이버시가 중요한 시나리오에서 특히 유용합니다.

2.2.1 FHE 스키마 개요

FHE 연구는 다양한 연산 패턴과 배포 제약 조건에 맞춰 최적화된 여러 구조 계열을 발전시켜 왔습니다. 아래에서는 대표적인 스키마와 주요 절충점을 간략히 정리합니다. 기초 개념과 실제 구현에 대한 참고 문헌도 함께 제시합니다.

BGV 및 BFV (패킹된 정수 산술) BGV 및 BFV 계열은 SIMD 패킹과 효율적인 정확한 정수 산술을 지원하는 격자 기반 스키마입니다 [12,13]. 이러한 스키마는 다수의 독립 슬롯에 대해 벡터화된 연산으로 표현 가능한 워크로드에서 높은 처리량을 제공합니다. 주요 장점은 정확한 산술 의미론과 배칭에 대한 안정적인 매개변수 선택입니다. 한계로는 회로 깊이에 따른 노이즈 축적, 비용이 높은 회전 및 재선형화 연산, 그리고 부트스트래핑 비용으로 인해 임의 깊이 평가가 어렵다는 점이 있습니다. 따라서 BGV/BFV 는 불리언 로직보다 배칭 가능한 정수 산술이 지배적인 워크로드에 적합합니다.

CKKS (근사 산술) CKKS 는 실수 및 복소수에 대한 근사 산술을 제공하여 선형 대수 및 머신러닝 워크로드에 적합한 컴팩트한 인코딩을 지원합니다 [14]. CKKS 암호문은

벡터화된 수치 연산에 대해 공간 효율적이며, 내적 및 행렬 연산을 자연스럽게 동형적으로 평가할 수 있습니다. 다만 회로 깊이 및 재조정(rescaling)에 따라 근사 오차가 증가하며, 결과 정확도를 유지하기 위해 정밀도 관리와 (경우에 따라) 부트스트래핑이 필요합니다. 따라서 CKKS 는 근사 계산이 허용되는 분석 및 추론 작업에 적합하지만, 결정론적 검증이 필요한 온체인 환경에는 제한적입니다.

GSW / TFHE (게이트 부트스트래핑 및 비트 단위 평가) GSW 는 동형 게이트 평가를 위한 행렬 기반 암호문 구조를 도입했으며, 이후 FHEW 와 TFHE 는 이를 발전시켜 낮은 지연의 빠른 부트스트래핑과 효율적인 게이트 단위 평가를 가능하게 했습니다 [15,16,17]. 이러한 구조는 불리언 회로에 특히 적합하며, 게이트 수준에서 노이즈를 효과적으로 제거하여 임의 깊이 회로에서도 안정적인 성능을 제공합니다. 단점은 수치 연산에서 비트 단위 처리가 필요해 저장 및 통신 비용이 증가한다는 점입니다.

블록체인 배포를 위한 설계 고려 사항 블록체인 및 스마트 계약 환경에서는 검증 가능성과 결정론, 낮고 예측 가능한 지연 시간, 제한된 온체인 저장 및 가스 비용, 그리고 애플리케이션 로직과 동형 연산 간의 직접적인 매핑이 요구됩니다. 이러한 제약 조건 하에서 TFHE 기반 접근은 매력적인 선택입니다. 이는 결정론적 정확성, 즉시 부트스트래핑을 통한 예측 가능한 지연 시간, 그리고 복잡한 노이즈 관리 없이 불리언 로직을 직접 동형 평가로 매핑할 수 있기 때문입니다. 반면 BGV/BFV 는 오프체인에서 대규모 정수 연산을 수행하고 요약 결과만 온체인에 게시하는 경우에 적합합니다. CKKS 는 암호화된 머신러닝 추론과 같은 근사 수치 연산에 적합합니다.

따라서 본 연구의 목표 활용 사례를 고려할 때, Primus 는 온체인 기본 연산을 위한 실용적인 기본 전략으로 TFHE 중심의 FHE 접근을 채택합니다.

2.2.2 FHE 연산의 검증 가능성

표준 FHE 는 암호화된 입력에 적용된 연산이 입력 기밀성을 유지하면서 결정론적으로 정확한 암호문 결과를 생성함을 보장합니다. 그러나 일반적인 FHE 구조는 특정 암호문 결과가 실제로 특정 입력에 대한 특정 프로그램 실행의 결과임을 공개적이고 효율적으로 검증할 수 있는 증거를 제공하지 않습니다.

허가 없는 환경에서는 이러한 한계가 특히 중요합니다. 온체인 검증자와 외부 감사자는 비용이 높은 동형 연산을 재실행하기 어렵고, 신뢰할 수 없는 실행자에게 검증을 위임하는 것은 분산 원장의 무결성을 훼손할 수 있기 때문입니다. 따라서 블록체인 배포를 위해서는 다음과 같은 요구 사항을 충족하는 검증 가능한 FHE 구조가 필요합니다.

1. **기본 연산 성능 영향 최소화**: 검증 메커니즘은 게이트당 또는 암호문당 지연 시간을 크게 증가시키거나 처리량을 저하시키는 수준의 FHE 기본 연산 변경을 피해야 합니다.
2. **간결하고 공개적인 증명**: 평가와 함께 생성되는 증명은 컴팩트하며 공개적으로 검증 가능해야 합니다. 검증은 연산 재실행이나 비밀 정보에 의존하지 않고 제한된 검증자(스마트 계약 포함)가 수행할 수 있어야 합니다.
3. **실용적인 증명 시간**: 증명 생성에 필요한 시간과 계산 자원은 실제 서비스 지연 시간 및 경제적 제약과 호환되어야 합니다. 과도한 증명 비용이나 높은 지연 시간은 이론적 보안과 무관하게 설계를 비실용적으로 만듭니다.

널리 연구된 접근 방식은 동형 평가를 외부의 간결한 증명 시스템(succinct argument system)과 결합하는 것입니다. 증명자는 동형 평가 트랜스크립트(또는 평가 및 일관성 검증을 모델링한 산술 회로)의 정확성을 입증하는 SNARK 를 생성합니다. 이 방식은 검증 비용이 낮고, 컴팩트하며 공개적으로 검증 가능한 증명을 제공하므로 온체인 검증에 적합합니다.

그러나 FHE 를 범용 SNARK 와 결합하는 것은 실용적인 규모에서 매우 어렵습니다. FHE 연산을 SNARK 증명으로 변환하고, 그에 따라 생성되는 대규모 산술 회로를 증명하는 과정은 과도한 CPU, 메모리, 시간 비용을 요구합니다. 실증적 결과에 따르면, 의미 있는 FHE 워크로드에 대한 증명 시간은 일반적으로 범용 하드웨어에서 수 시간에서 수 일에 이르며, 이는 지연 시간에 민감한 블록체인 애플리케이션과 양립하기 어렵습니다.

따라서 실용적인 검증 가능한 FHE 를 위해서는 범용 SNARK 백엔드에 의존하기보다, FHE 연산에 특화된 SNARK 시스템이 필요합니다.

2.2.3 HasteBoots: Primus 의 검증 가능한 FHE

위탁된 프라이버시 보존 연산에서 FHE 는 신뢰할 수 없는 실행자에게 위임된 데이터의 기밀성을 보장하지만, 위임된 연산이 올바르게 수행되었음을 증명하는 공개적이고 간결한 증거를 자체적으로 생성하지는 않습니다. 따라서 클라이언트와 외부 검증자는 지정된 프로그램이 지정된 암호문에 대해 실행되었으며, 게시된 결과가 해당 실행에 대응함을 보여주는 컴팩트하고 효율적으로 검증 가능한 증명을 필요로 합니다.

범용 SNARK 또는 FHE 구현을 zkVM 으로 컴파일하는 기존 접근 방식은 현실적인 FHE 워크로드에 대해 실용적이지 않습니다. 고수준 FHE 코드를 일반 회로로 컴파일하면 증명 크기와 제약 조건 수가 크게 증가합니다. 특히 핵심 FHE 커널(예: 부트스트래핑)을 SNARK 로 직접 검증하는 경우, 여전히 수 분에서 수 시간에 이르는 증명 시간이 요구됩니다.

이러한 비효율성은 두 가지 구조적 불일치에서 비롯됩니다. (a) FHE 평가는 일반적인 산술화(arithmetization)로는 비효율적으로 표현되는 특수한 대수 연산(NTT, 모듈러스 스위칭 및 키 스위칭, 어큐뮬레이터 업데이트, 링 다항식 변환)에 의존하며, (b) FHE 내부 상태는 비트, 림(limbs), 암호문 구성 요소로 이루어져 있어 단순한 비트 단위 증명 인코딩이 회로 크기와 증명자 리소스 측면에서 큰 곱셈적 팽창을 유발합니다.

HasteBoots 의 기여 HasteBoots [18]의 주요 기여는 다음과 같습니다. HasteBoots 는 부트스트래핑을 포함한 FHE 평가의 정확성을 인증하도록 설계된 맞춤형 간결한 증명 시스템으로, 컴팩트하고 공개적으로 검증 가능한 증명을 생성하면서 증명자 비용을 최소화하는 것을 목표로 합니다.

- PIOP/PCS 증명 아키텍처: 다항식 대화형 오라클 증명(PIOP) 모델로 정식화되고, 다항식 약속 스키마(PCS)로 인스턴스화된 구조를 제시합니다. 이 아키텍처는 부트스트래핑을 포함한 FHE 평가를 모듈식 하위 프로토콜로 분해하며, 그 정확성은 간결한 다항식 오프닝을 통해 검증됩니다.
- FHE 인스턴스화 및 매개변수 선택: 효율적인 게이트 부트스트래핑 의미론을 고려하여 FHEW/TFHE 를 타겟 FHE 기본 요소로 선택하는 근거를 제시하고, 보안성과 실용적인 증명 비용 간 균형을 이루는 매개변수 영역을 정의합니다.

- 특수 PIOP 및 알고리즘 가젯: 핵심 FHE 연산을 위한 맞춤형 PIOP 및 산술 가젯을 제안합니다. 여기에는 비트 반전을 포함한 최적화된 정수론적 변환(NTT) 증명자, 배칭된 다항식 변환을 희소 선형 쿼리로 압축하는 배칭 NTT 감소, 그리고 노이즈 축적에 대한 건전성을 유지하면서 검증을 저비용 범위 및 일관성 확인으로 축소하는 모듈러스 스위칭 증명이 포함됩니다.
- PCS 패킹 및 상각 기술: 선형 코드 기반 약속을 활용하는 PCS 인스턴스화를 도입하고, 다수의 소규모 다항식에 대해 오프닝 비용을 상각하는 패킹 전략과 결합하여 추가적인 필드 가정 없이 증명자 처리량을 개선합니다.
- 점근적 및 실증적 복잡도: 선택된 매개변수화 및 프로토콜 분해 하에서 HasteBoots 는 FHE 연산 규모에 선형적으로 비례하는 증명자 복잡도를 달성합니다. 참조 구현 결과, Apple M4 환경에서 부트스트래핑을 포함한 FHE NAND 연산에 대한 증명을 약 5 초 내에 생성할 수 있으며, 이는 범용 SNARK 기반 접근 대비 유의미한 성능 개선을 보여줍니다.

2.2.4 HasteBoots 벤치마크

Apple M4 환경에서 HasteBoots 의 성능을 평가하여 표 6 에 제시된 각 FHE 연산에 대한 증명 시간, 검증 시간, 그리고 증명 크기를 측정했습니다. 결과에 따르면 부트스트래핑 절차는 NAND 회로에서 가장 많은 시간을 차지하는 단계입니다.

특히 부트스트래핑 과정에서는 어큐뮬레이터 업데이트가 복잡한 워크플로우로 인해 전체 시간의 약 95%를 차지하며 비용의 대부분을 지배합니다. 전반적으로 HasteBoots 는 약 5.13 초의 증명 시간, 219ms 의 검증 시간, 그리고 약 107MB 의 증명 크기를 달성했습니다.

FHE 연산		**증명 시간**	**검증 시간**	**증명 크기**
LWE 덧셈		17 ms	6 ms	11 MB
부트스트래핑	Mono-NTT	151 ms	36 ms	35 MB
	어큐뮬레이터 업데이트	5.02 s	219 ms	107 MB
	모듈러스 스위칭	22 ms	7 ms	12 MB
NAND		5.13 s	219 ms	107 MB

Table 6: Apple M4에서의 HasteBoots 성능.

증명 시간 vs 증명 크기 우리의 SNARG 구조에서 성능 절충을 보다 명확히 이해하기 위해, 표 7 에서는 PCS 와 PIOP 구성 요소의 기여도를 분리하여 분석합니다. PCS 비용은 다항식 약속 생성 시간과 대규모 약속 다항식에 대한 평가 증명 생성 시간으로 구성됩니다.

Brakedown PCS 를 사용할 경우, 증명자는 전체 증명 시간의 약 25%를 PCS 에 소비하는 반면, PCS 는 검증 시간과 증명 크기의 약 95%를 차지합니다. 증명 시간과 증명 크기 간의 절충을 분석하기 위해 Brakedown PCS 를 Jolt-b 구현의 BaseFold 로 대체합니다. 표 7 에서 확인할 수 있듯이, 이러한 교체는 증명 시간을 증가시키는 대신 검증 시간과 증명 크기를 모두 크게 감소시킵니다.

그럼에도 불구하고 이러한 절충을 적용하더라도, 전체 증명 시간은 [19]에서 보고된 최첨단 결과(약 30 분)보다 여전히 빠릅니다.

		증명 시간	검증 시간	증명 크기
NAND w/ Brakedown	PCS	1.39 s	212.5 ms	104 MB
	PIOP	3.75 s	6.9 ms	3 MB
	SNARG	5.13 s	219.4 ms	107 MB
NAND w/ BaseFold	PCS	63.74 s	4 ms	8 MB
	PIOP	3.75 s	6.9 ms	3 MB
	SNARG	67.49 s	10.9 ms	11 MB

Table 7: Brakedown PCS와 BaseFold PCS 사용 시 비교.

증명 시간	RISC0 [20]	SP1 [21]	[19]	HasteBoots
M3 Pro (8 코어)	–	–	40 분	**7 초**
C61.meta (128 코어)	–	–	21 분	**5 초**
Hpc7a.96xlarge (192 코어)	4600 분	1500 분	18 분	**4 초**
M4 Pro	–	–	–	**5 초**

Table 8: 다양한 장치에서의 단일 부트스트래핑 절차에 대한 증명 시간.

이전 연구와의 비교 표 8 에 요약된 바와 같이 HasteBoots 의 성능을 FHE 부트스트래핑 증명에 관한 기존 연구와 비교했습니다. RISC0 [20] 및 SP1 [21]과 같은 zkVM 기반 접근 방식은 복잡한 FHE 프로그램을 컴파일할 때 발생하는 성능 손실로 인해 각각 약 3 일과 1 일의 증명 시간이 필요합니다.

Thibault 와 Walter [19]는 TFHE 부트스트래핑 절차를 위한 SNARK 친화적인 산술 회로를 설계하고 Plonk 로 이를 증명하여 증명 시간을 약 30 분 수준으로 단축했습니다. 그러나 범용 SNARK 인 Plonk 는 FHE 전용 연산에 적용될 경우 여전히 일부 비효율성을 보입니다.

반면 HasteBoots 는 전체 산술 회로를 증명하는 방식을 피하고, FHE 연산의 입력과 출력 간 관계를 직접 증명함으로써 10 초 미만의 증명 생성이 가능합니다. 구체적으로 [19]는 O(N log N) 크기의 명시적 NTT 회로를 증명하여 준선형(quasi-linear) 증명자 시간을 갖는 반면, 본 접근 방식은 NTT 관계를 직접 증명하여 회로 인코딩을 피하고 선형 시간 증명자를 달성합니다.

또한 [19]는 단항식 곱셈을 위해 분해 기반 접근을 사용하여 O(N log N) 크기의 추가 회로를 구성하는 반면, 본 프로토콜은 단항식에 대한 배칭 NTT 연산을 선형 시간으로 수행합니다. 마지막으로 [19]는 모듈러스 스위칭 과정에서 FHE 암호문에 추가 노이즈를 도입하는 반면, 본 프로토콜은 이를 회피합니다.

[19]는 약 8ms 의 검증 시간과 약 200KB 의 증명 크기를 달성하는 반면, HasteBoots 는 약 219ms 의 검증 시간과 107MB 의 증명 크기를 필요로 합니다. 다만 HasteBoots 의 증명 크기와 검증 시간은 재귀 기법 도입 및 Brakedown 을 보다 효율적인 PCS 로 대체함으로써 추가적인 최적화가 가능합니다.

3 Primus 네트워크의 설계

이 섹션에서는 에이전트 경제를 위해 설계된 신뢰, 검증 및 프라이버시 레이어인 Primus 네트워크의 아키텍처와 핵심 구성 요소를 제시합니다. Primus 네트워크는 zkTLS 와 FHE 를 포함한 고급 암호 기술을 활용하여, 기존 블록체인 인프라와의 호환성을 유지하면서 검증 가능한 에이전트 행동과 기밀 데이터 처리를 가능하게 합니다.

Primus 는 에이전트 경제를 위한 전용 검증 및 연산 레이어로서, 에이전트가 특화되고 경제적으로 정렬된 인프라를 통해 데이터에 접근하고, API 기반 작업을 실행하며, 기밀 연산을 수행할 수 있도록 합니다. 주요 설계 목표는 다음과 같습니다: (i) 데이터와 행동의 검증 가능한 출처, (ii) 연산의 기밀성, (iii) 결함 허용성과 높은 가용성, (iv) 네트워크 운영자를 위한 명확한 경제적 인센티브.

3.1 데이터 검증 레이어

데이터 검증 레이어는 zkTLS 를 구현하여 기본 데이터의 기밀성을 유지하면서 데이터 진위성, API 호출 무결성, 그리고 행동 출처에 대한 검증 가능한 보장을 제공합니다. 이 레이어는 모듈식이며 확장 가능하도록 설계되어, 에이전트가 데이터를 안전하게 접근하고 외부 서비스와 상호작용하며 독립적으로 검증 가능한 결과를 반환해야 하는 다양한 워크플로우를 지원합니다.

3.1.1 주요 역할

Primus 네트워크는 책임 범위가 명확히 정의된 여러 참여자 역할로 구성됩니다.

- 개발자(Developer) Primus SDK 를 통해 애플리케이션에 Primus 기능을 통합하는 주체입니다. 개발자는 작업 페이로드(데이터 소스, 검증 술어, 예상 출력 형식, 결제 매개변수)를 구성하고, 안전한 랑데부 및 제출 흐름을 시작하며, 반환된 인증서와 증명을 소비하고 검증합니다.
- 클라이언트(Client) 검증 요청을 시작하고 인증서를 소비하는 주체입니다(최종 사용자, 지갑 또는 사용자를 대신해 동작하는 게이트웨이 포함). 클라이언트는 작업 자금을 조달하고(수수료 또는 결제 약정 예치), 작업 선호도(예: 프라이버시, 지연 시간)를 선택하며, 인증자로부터 수신한 인증서를 검증합니다.
- 인증자 노드(Attestor node) zkTLS 작업을 실행하고 컴팩트하며 검증 가능한 인증서를 생성하는 컴퓨팅 노드입니다. 인증자 운영자는 레지스트리 정책에 따라 등록하고 담보를 예치해야 합니다. 노드는 보호된 실행 환경에서 인증자 소프트웨어(웹 및 모바일 런타임 포함)를 실행하며, 정상적으로 완료된 작업에 대해 수수료를 받고, 잘못되거나 누락된 작업에 대해서는 처벌(예: 슬래싱)을 받을 수 있습니다.
- 인증자 노드의 보안 인증자 노드의 보안은 노드 운영자와 클라이언트를 보호하고, 오프체인 zkTLS 실행의 연산 무결성을 보장하는 데 핵심적입니다. 기본 zkTLS 프로토콜만으로는 실행 무결성을 강제할 수 없으므로, Primus 는 하드웨어 기반 보증을 통해 이를 보완합니다. 구체적으로 네트워크는 Phala 의 TEE 플랫폼을 통합하여 런타임 동작을 제약하고, 실행에 대한 검증 가능한 인증을 생성하며, 중요한 비밀 정보를 보호합니다.
- 키 관리 노드 키는 TEE 내부에서 동작하는 KMS 에 의해 관리됩니다. 키는 엔클레이브 내부에서만 생성 및 저장되며, 호스트 운영 체제에는 노출되지 않습니다. zkTLS 세션과 인증에 필요한 모든 서명 및 민감한 암호 연산은 TEE 내부에서 수행되며, 노드가 올바른 프로토콜 코드와 입력을 실행했음을 입증할 수 있을 때만 허용됩니다.
- 버전 및 배포 제어 코드 주입 및 공급망 공격을 방지하기 위해, 서명되고 공식적으로 릴리스된 소프트웨어 이미지만 인증자 배포에 허용됩니다. TEE 는 원격 인증을 통해 런타임 무결성을 검증하고, 검증되지 않은 바이너리를

차단합니다. 업그레이드 및 배포 프로세스는 서명된 릴리스와 재현 가능한 빌드 결과에 기반하며, 운영자는 승인된 버전만 엔클레이브 내부에서 실행되도록 인증된 업데이트 채널을 따라야 합니다.

3.1.2 네트워크 아키텍처 및 워크플로

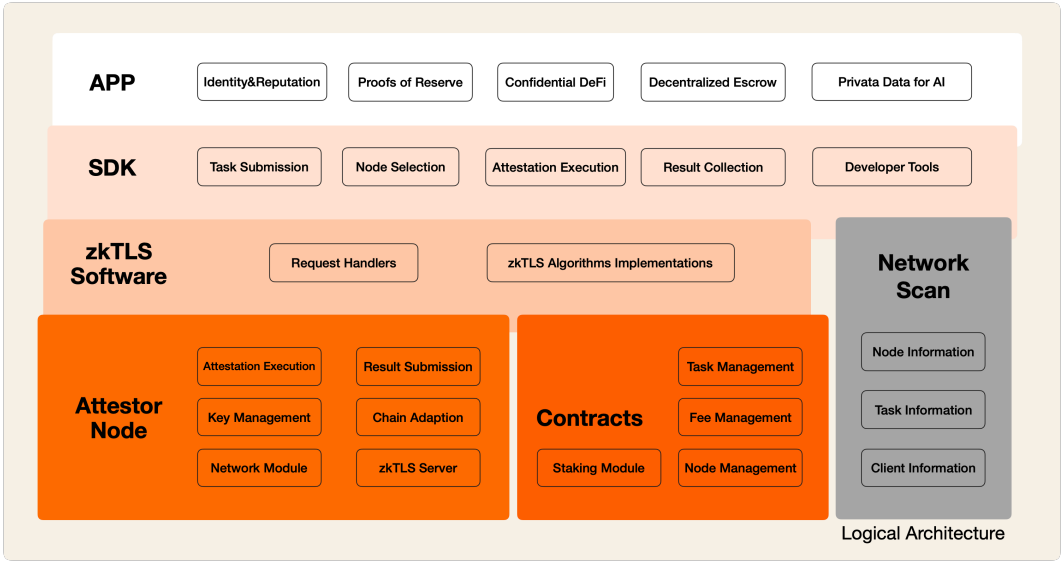

Figure 3: 상위 수준 Primus 네트워크 아키텍처

그림 3 과 같이, 상위 수준에서 Primus 시스템은 온체인 시스템 계약(레지스트리, 스테이킹, 정산), dApp 및 클라이언트가 사용하는 Primus SDK, 인증자 워커, 그리고 선택적 검증자/감사자로 구성됩니다. 일반적인 세션은 그림 4 와 같이 다음 순서로 진행됩니다.

1. 클라이언트는 작업을 생성하고 정산 계약에 필요한 수수료 또는 결제 약정을 게시합니다.
2. 레지스트리는 등록된 인증자 노드 집합에서 후보 인증자를 균등 무작위로 선택합니다.
3. 클라이언트와 인증자는 zkTLS 프로토콜(MPC 또는 프록시 모드)을 수행하여 인증서를 생성하고, 인증자는 결과를 클라이언트에 반환합니다.
4. 인증서는 온체인 검증자에 의해 유효성이 확인되며, 검증 성공 시 정산 계약은 인증자에게 대금을 지급하고 관련 상태(예: 평판 점수, 로그)를 업데이트합니다.

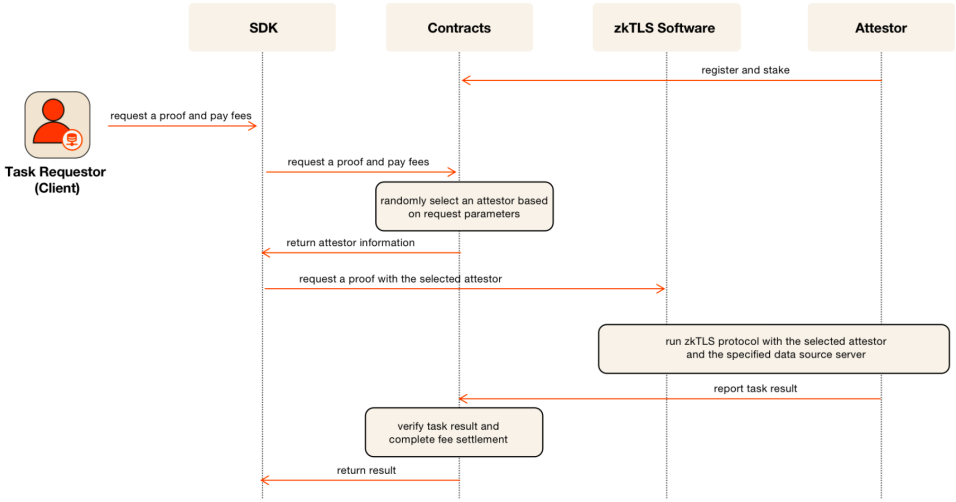

Figure 4: zkTLS 세션을 위한 핵심 작업 흐름

3.1.3 네트워크 경제

클라이언트는 네트워크가 제공하는 검증 서비스를 이용하기 위해 PRIM 또는 기타 지원되는 토큰으로 작업 수수료를 지불할 수 있습니다. 수수료는 작업 실행 전에 시스템 컨트랙트에 사전 납부됩니다. 작업 수행에 따른 수수료는 실제로 zkTLS 세션을 수행한 검증 노드와 프로토콜 간에 분배됩니다. 검증 노드는 네트워크 참여를 위해 일정 수량의 토큰을 스테이킹해야 하며, 일정 대기 기간 이후 정식 노드로 참여할 수 있습니다. 또한 스테이킹 해지 시에는 일정 기간의 언본딩 과정을 거친 후 예치된 자산이 반환됩니다.

3.1.4 처벌 및 분쟁 해결

프로토콜을 올바르게 실행하지 않은 상태에서 zkTLS 세션에 서명하는 등 인증자의 부정행위는 처벌 대상입니다. 프로토콜의 정확성은 TEE 인증과 분쟁 해결 과정에서 제출되는 온체인 또는 오프체인 증거를 통해 입증됩니다.

노드의 불안정성으로 인해 실행 실패나 반복적인 타임아웃이 발생하는 경우, 해당 사건은 시스템 계약에 보고되어야 합니다. 사전에 정의된 허용 범위와 실패 임계값에 따라 처벌 적용 여부가 결정됩니다. 허용 범위 내의 간헐적 실패에는 처벌이 적용되지 않지만, 설정된 타임아웃 또는 실패 임계값을 초과할 경우 제재(예: 경고, 일시 정지, 또는 스테이크 슬래싱)가 적용됩니다.

처벌 메커니즘은 단계적으로 도입됩니다. 초기 단계에서는 자동 슬래싱이 적용되지 않으며 승인된 노드만 참여할 수 있습니다. 징벌적 조치는 네트워크가 운영 안정성을 입증한 이후 도입됩니다. 처벌이 활성화되면, 정확성과 책임을 신뢰성 있게 판단하기 위해 TEE 인증과 합의된 판결 워크플로를 기반으로 작업 실행과 분쟁을 검증합니다.

3.1.5 거버넌스

거버넌스 메커니즘은 네트워크 업그레이드, 파라미터 조정 및 시스템에 영향을 미치는 주요 의사결정을 담당합니다. 초기 단계에서는 네트워크의 안정성과 확장에 집중하기 위해 거버넌스 기능이 활성화되지 않습니다. 향후 거버넌스가 도입될 경우, 검증 노드의 최소 스테이킹 요건, 언본딩 기간, 클라이언트의 최소 참여 요건, 일반 참여자의 언본딩

기간, 그리고 참여자가 위임한 검증 노드에 분배 가능한 프로토콜 기반 인센티브 비율 범위 등의 항목이 포함될 수 있습니다.

3.2 데이터 연산 레이어

데이터 연산 레이어는 데이터 검증 레이어를 확장하여 보다 풍부한 프라이버시 보존 연산을 가능하게 합니다. 이 레이어는 애플리케이션이 민감한 입력을 드러내지 않고도 복잡한 연산을 수행할 수 있도록 하며, 효율적으로 검증 가능한 출력과 간결한 증명을 생성합니다.

3.2.1 zkVM 을 사용한 DVC 모드

DVC(데이터 검증 및 연산) 모드는 zkTLS 를 사용해 암호화된 TLS 트랜스크립트를 증명함으로써 HTTPS 요청과 응답의 검증 가능한 정확성과 진위성을 확보하고, 인증된 비공개 데이터 처리를 외부 zkVM 에 위탁합니다. 이 모드에서 Primus zkTLS 는 필요한 암호 기본 요소를 지원하는 모든 zkVM 또는 zkDSL 과 통합될 수 있으며, 다양한 애플리케이션이 인증된 데이터에 대해 복잡한 연산을 수행하면서도 zkTLS 의 프라이버시 보장을 활용할 수 있도록 합니다. DVC 모드는 다음과 같은 구성 요소로 이루어집니다.

- **dApp:** 개발자가 Primus SDK 를 사용해 구축한 최종 애플리케이션입니다. 검증 작업을 제출하고, 인증서와 증명을 수신하며, 검증된 출력을 소비합니다.
- **Primus 네트워크 SDK:** 개발자가 zkTLS 작업을 구성하고, 로컬에서 인증서를 검증하며, 애플리케이션 로직에 사용할 평문 출력을 추출할 수 있도록 지원하는 클라이언트 라이브러리 및 도구 모음입니다.
- **Primus 데이터 검증 네트워크:** zkTLS 작업을 실행하고 인증서를 발행하며, 경제적 인센티브와 처벌을 집행하는 탈중앙화 검증 레이어(레지스트리, 스테이킹, 정산 계약, 인증자 노드)입니다.
- **검증 프로그램:** 인증 검증과 도메인 특화 연산을 수행하는, TEE 또는 zkVM 내부에서 실행되는 신뢰된 프로그램입니다.
  - **Primus zkTLS 검증 프로그램:** 엔클레이브 내부에서 인증서 서명을 검증하고, 소스 URL 과 트랜스크립트를 확인하며, 후속 처리를 위해 인증된 평문 페이로드를 추출하는 표준 Rust 구성 요소입니다.
  - **비즈니스 프로그램:** TEE 또는 zkVM 내부에서 실행되는 개발자 제공 코드(Rust 또는 기타 지원 런타임)로, 검증된 평문을 소비하고 애플리케이션 특화 검사를 수행하며, 수행된 연산에 대한 zkVM 증명 또는 인증서를 출력합니다.

DVC 워크플로우 DVC 워크플로우는 밀접하게 결합된 두 단계로 구성됩니다. (A) 봉인된 인증서와 인증된 트랜스크립트를 생성하는 zkTLS 세션, (B) 해당 인증서를 활용해 요청된 연산에 대한 간결한 증명을 생성하는 TEE/zkVM 내부의 실행 단계입니다. 그림 5 는 참여자와 메시지 흐름을 보여주며, 아래에 상세 단계를 설명합니다.

A 단계: zkTLS 세션 및 인증

1. **작업 제출.** dApp 은 작업 기술자(소스 URL, 요청 매개변수, 검증 술어, 예상 출력 필드, 결제 매개변수)를 구성하고, Primus SDK 를 통해 이를 Primus 정산 계약에

제출합니다. 정산 계약은 결제 약정을 기록하고, 작업 식별자와 레지스트리에서 무작위로 할당된 인증자를 반환합니다.

2. **zkTLS 프로토콜.** 선택된 인증자와 클라이언트는 선택된 모드(MPC 또는 프록시)에서 zkTLS 프로토콜을 수행합니다. 인증자는 TEE 내부에서 zkTLS 세션을 실행하여 모든 코드가 예상대로 동작하도록 보장합니다. 클라이언트는 봉인된 메타데이터에 대한 서명을 포함한 인증서를 인증자로부터 수신하며, 해당 인증서는 후속 단계에서 사용됩니다.

3. **비공개 데이터 추출(개발자 로컬).** 인증서가 올바른 실행을 증명하면, SDK 는 dApp 이 애플리케이션 로직에 필요한 인증된 평문 필드를 추출할 수 있도록 지원합니다. 이 과정은 다른 비밀 정보를 네트워크에 노출하지 않고 수행되며, 평문은 로컬에서 복호화됩니다.

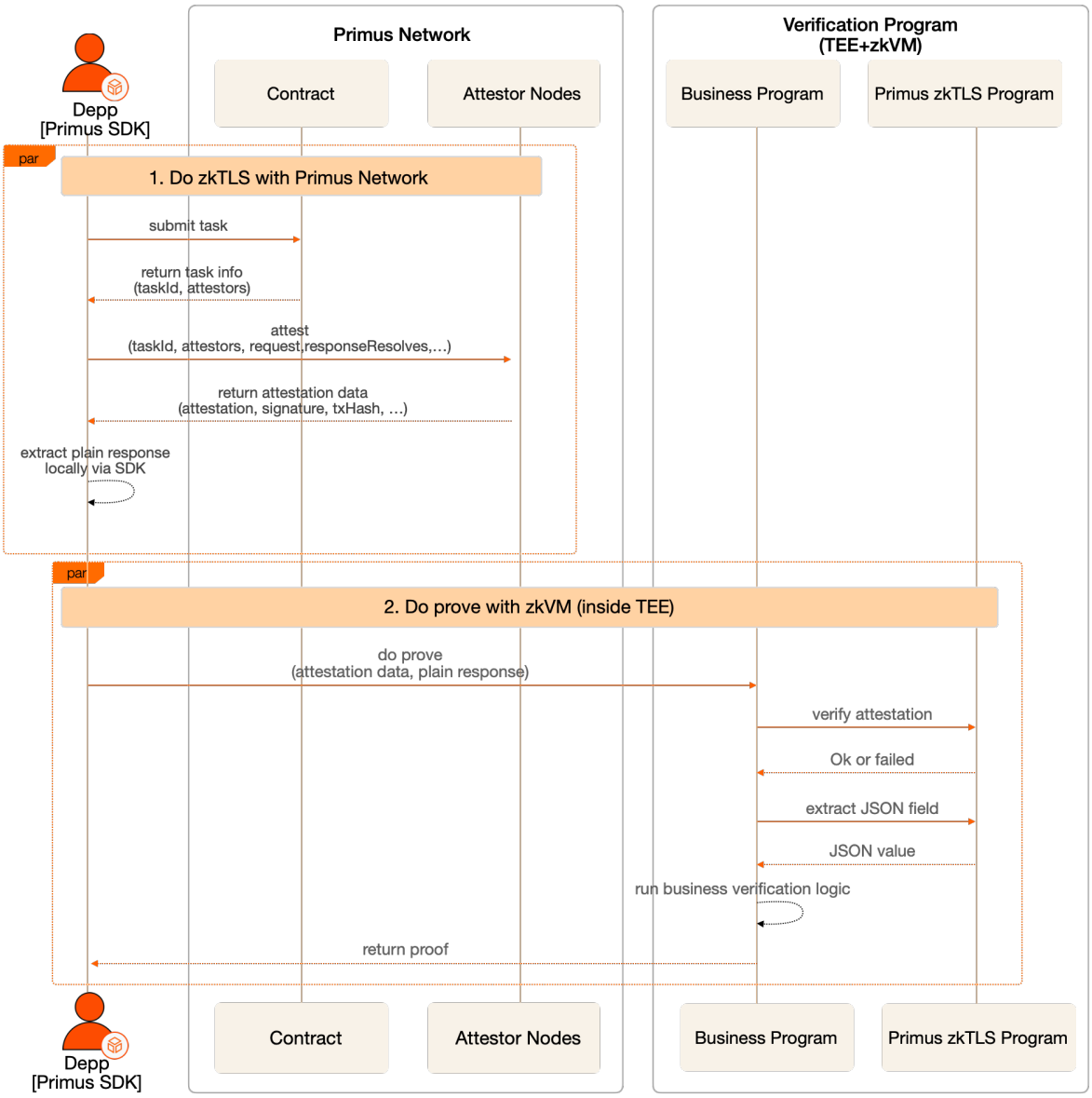

Figure 5: DVC 워크플로우

B 단계: zkVM 증명 및 검증

4. **검증 프로그램으로 전달.** dApp 은 최소한의 인증서 패키지와 추출된 평문을 TEE 또는 zkVM 내부에서 실행되는 검증 프로그램으로 전달합니다. 이 메시지는 일반적으로 인증된 채널을 통해 전송되며, 작업 ID 와 비즈니스 로직에 필요한 매개변수를 포함할 수 있습니다. zkVM 에서 프로그램은 아래 단계를 검증하고, 올바른 실행에 대한 간결한 증명을 생성합니다.
5. **인증 검증.** 먼저 Primus zkTLS 검증 프로그램을 호출하여 인증서를 재검증합니다. 여기에는 엔클레이브 서명 검증, 엔클레이브 측정값 검증(표준 검증자 바이너리가 실행 중인지 확인), 트랜스크립트 해시와 소스 URL 검증, 그리고 재생 방지(논스, 타임스탬프 또는 온체인 작업 바인딩)의 적용이 포함됩니다.
6. **도메인 특화 처리.** 인증 검증이 성공하면, 비즈니스 프로그램은 필요한 JSON 필드 또는 기타 입력을 추출하고, 정의된 애플리케이션 특화 연산을 수행합니다.
7. **증명 반환.** 생성된 zkVM 증명을 dApp 에 반환합니다. dApp 은 해당 증명을 온체인 검증자 또는 정산 계약에 제출할 수 있습니다.

3.2.2 FHE 를 사용한 기밀 연산

프라이버시 보장을 한층 강화하기 위해 Primus 는 FHE 를 데이터 연산 레이어에 통합합니다. 이를 통해 암호화된 데이터 위에서 직접 연산을 수행할 수 있으므로, 처리 전 과정에서 민감한 정보의 기밀성이 유지됩니다.

블록체인은 투명성, 보안, 신뢰 최소화 측면에서 강점을 가지지만, 이러한 투명성에는 절충이 따릅니다. 거래, 계약 호출, 계정 상태와 같은 온체인 활동은 공개되므로 금융이나 의료와 같은 민감한 활용 사례에는 제약이 존재합니다. 완전 동형 암호(FHE)는 암호문에 대한 연산을 가능하게 함으로써 이러한 문제를 완화합니다. 블록체인은 평문이 아닌 암호화된 데이터와만 상호작용하며, 권한이 있는 당사자는 복호화를 통해 올바른 결과를 복구할 수 있습니다. 결과적으로 스마트 계약은 탈중앙성과 보안을 유지하면서도 프라이버시를 보존한 상태로 동작할 수 있습니다.

아키텍처. Primus 는 블록체인 환경에서 프라이버시 보존 연산을 지원하기 위한 모듈식 계층 구조인 FHETransform 을 도입합니다. FHETransform 은 온체인 스마트 계약과 오프체인 처리 계층을 결합하여 안전하고 효율적인 완전 동형 암호 워크플로를 제공합니다.

주요 구성 요소. FHETransform 은 다음과 같은 구성 요소로 이루어집니다.

- **FHE Solidity 라이브러리(일명 FHE 시스템 계약):** 개발자가 표준 Solidity 환경에서 암호화된 데이터 타입과 연산을 활용하여 기밀 계약을 작성할 수 있도록 지원합니다.

  FHE 라이브러리는 암호화 데이터 연산에 대해 온체인에서 기호적 연산을 수행하고, 계산 비용이 높은 FHE 연산은 오프체인 서비스 레이어(AlphaTrion)에 위임합니다. 이 라이브러리는 계약이 평문을 노출하지 않고도 암호화된 값과 연산 기술자를 참조할 수 있도록 컴팩트한 API 를 제공합니다.

  FHE 라이브러리는 주로 다음 두 계약으로 구성됩니다.

  - FHEExecutorContract: 암호화 데이터 처리(데이터 타입 및 연산자 인코딩)를 담당합니다.

- ACLContract: 접근 제어를 관리하며, 특정 주소가 암호문 핸들에 접근하거나 이를 복호화할 권한이 있는지 판단합니다.

이 라이브러리는 비용이 높은 FHE 연산을 온체인에서 직접 실행하지 않습니다. 대신 AlphaTrion 이 오프체인에서 실제 암호화 연산을 수행할 수 있도록 기호적 연산 기술자(예: 연산 식별자, 인수 해시)를 기록합니다.

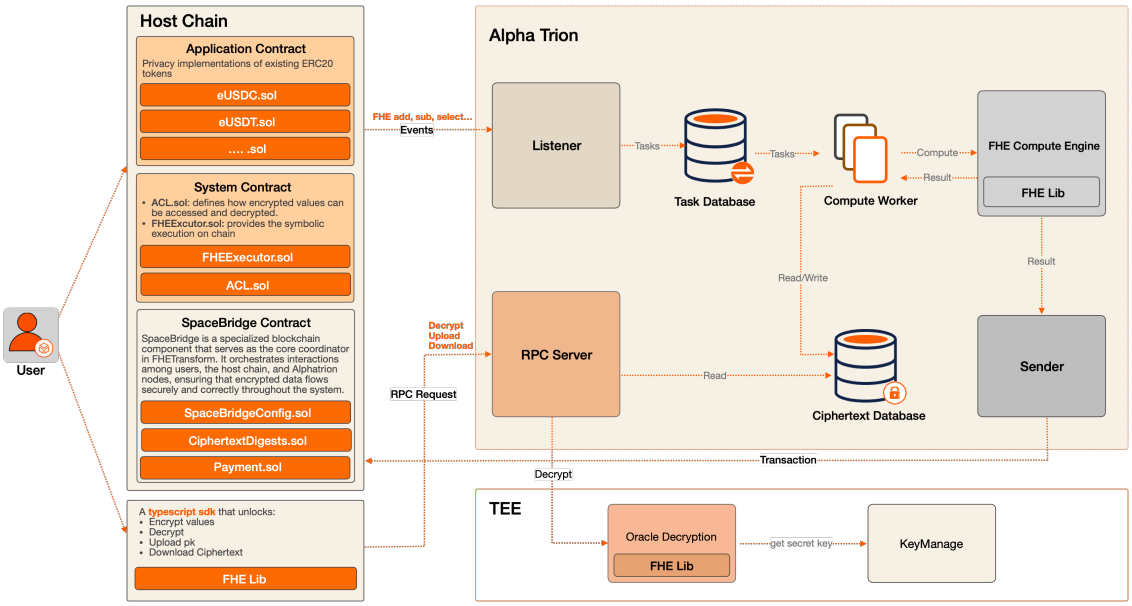

Figure 6: FHETransform 아키텍처

- **AlphaTrion:** AlphaTrion 은 FHE 워크로드를 실행하는 오프체인 연산 구성 요소입니다. CPU 및 GPU 가속을 지원하는 TFHE 라이브러리 위에 구현되며, 호스트 체인에서 발생하는 FHE 관련 이벤트를 모니터링합니다. 이벤트는 작업 데이터베이스에 큐잉되고, 하나 이상의 연산 워커가 이를 처리합니다. 완료되면 계산된 암호문과 결과 메타데이터가 데이터베이스에 기록됩니다. 여기서는 두 가지 실행 모드를 구분합니다.

  기호 실행(온체인): 스마트 계약이 FHE 연산을 호출할 때, 온체인 FHE 계약은 실제 FHE 연산을 수행하지 않습니다. 대신 다음과 같은 기호 실행을 수행합니다.

  - bytes32 핸들 값을 입력으로 받습니다.
  - 입력 핸들, 연산 유형, 결과 유형, 매개변수를 기반으로 새로운 출력 핸들을 생성합니다.
  - AlphaTrion 이 오프체인에서 실제 연산을 수행하도록 알리기 위해 관련 핸들을 포함한 온체인 이벤트를 발생시킵니다.

  실제 실행(오프체인): AlphaTrion 워커는 큐에 등록된 작업을 가져와 다음과 같이 수행합니다.

  - 입력 핸들을 사용해 데이터베이스에서 암호문을 조회합니다.
  - TFHE 라이브러리를 사용해 실제 FHE 연산을 수행합니다.
  - 결과 암호문을 압축하여 데이터베이스의 출력 핸들에 저장합니다.
  - 검증을 위해 암호문의 암호학적 다이제스트(digest)를 체인에 업로드합니다.

각 AlphaTrion 연산 또는 입력 검증에는, 제 3 자가 결과의 정확성을 독립적으로 검증할 수 있도록 암호학적 약속과 디지털 서명이 포함됩니다.

- **SpaceBridge:** 사용자, 호스트 체인, AlphaTrion 노드 간 상호작용을 중개하는 블록체인 측 코디네이터입니다. SpaceBridge 는 검증, 합의 집행, 결제 정산을 포함하여 암호화 데이터 연산의 전체 생애주기를 관리합니다.

  SpaceBridge 는 구성, 암호문 장부 관리, 결제 로직을 담당하는 소수의 온체인 계약으로 구성됩니다. 주요 계약은 다음과 같습니다.

  - SpaceBridgeConfig: SpaceBridge 설정의 단일 기준점입니다. AlphaTrion 노드 레지스트리(트랜잭션 발신자 및 서명자 주소, 합의 임계값)를 관리하며, 네트워크 토폴로지와 검증 규칙에 대한 인증된 업데이트를 지원합니다.
  - CiphertextDigests: EVM 호환 체인에서 암호문 다이제스트를 저장하고 관리합니다. 암호화된 입력을 검증하고 AlphaTrion 노드가 제출한 다이제스트를 기록합니다. 다수의 노드가 동일한 다이제스트를 제출하면, 해당 결과에 대한 합의를 확정합니다.
  - PaymentContract: FHE 서비스(예: 공개 복호화, 사용자별 복호화, 콜백 실행)에 대한 가격 책정과 수수료 징수를 관리합니다. 결제 정산을 수행하고 프로토콜 규칙에 따라 AlphaTrion 노드에 보상을 분배합니다.

  이들 계약은 함께 SpaceBridge 가 온체인 감사 가능성과 경제적 인센티브를 유지하면서, 분산된 코프로세서 네트워크 전반에서 신뢰 최소화형 프라이버시 보존 연산을 조정할 수 있도록 합니다.
- **전처리기(Preprocessor):** 기존 EVM 스마트 계약을 FHE 지원 계약으로 변환하여, 개발자 워크플로우 변경을 최소화하면서 기밀 트랜잭션을 가능하게 하는 오프체인 구성 요소입니다.
- **키 관리 서비스(KMS):** FHE 키를 생성, 순환, 관리하고, 권한이 부여된 경우 안전하고 검증 가능한 복호화를 수행하는 임계값 MPC 기반 네트워크입니다.

4 활용 사례

다음 시나리오들은 Primus 가 에이전트 경제의 핵심 수직 영역 전반에서, 에이전트가 입증 가능한 행동과 프라이버시를 보존하는 데이터 접근을 바탕으로 작동할 수 있도록 하는 방식을 보여줍니다.

4.1 에이전트가 관리하는 과소담보 대출

에이전트는 비공개 오프체인 인증과 온체인 행동 신호를 결합함으로써 더 효율적인 신용 시장을 열 수 있습니다. 차입자는 대출 에이전트에 권한을 위임하고, 해당 에이전트는 zkTLS 를 통해 급여 제공자, 은행 API, 상환 서비스로부터 인증된 응답을 수집한 뒤, 소득 구간, 준비금 하한, 상환 점수와 같이 정책상 관련 있는 사실만을 포함한 컴팩트한 인증을 생성합니다. 반대편에서는 대출기관 에이전트가 이러한 인증을 온체인 활동 이력 및 실시간 시장 가격과 결합하여 신용 결정을 계산하고, 동적인 담보 비율을 설정하며, 위험 조정 이자 일정을 제안합니다.

이 흐름을 통해 차입자는 원시 금융 기록을 공유하지 않고도 신용 시장에 참여할 수 있으며, 대출기관은 더 높은 신뢰도의 감사 가능한 신호를 얻습니다. 그 결과 언더라이팅

주기가 빨라지고, 자본 효율성이 향상되며, 사용자가 에이전트가 어떤 인증을 캐시, 재사용, 철회할 수 있는지 세밀하게 통제할 수 있는 더 안전한 위임 모델이 가능해집니다.

4.2 에이전트 기반 준비금 증명

재무 또는 커스터디 기능을 관리하는 에이전트는 계정 단위 세부 정보를 공개하지 않고도 인증되고 프라이버시를 보존하는 준비금 증명을 생성할 수 있습니다. 준비금 에이전트는 zkTLS 를 통해 수탁기관과 거래상대방으로부터 인증된 잔액 및 부채 데이터를 수집하고, 이를 기밀 실행 환경 내에서 집계한 뒤 감사인이나 시장 참여자가 독립적으로 검증할 수 있는 커버리지 지표에 대한 컴팩트한 인증을 발행합니다.

이 아키텍처는 독점적 포지션과 거래상대방 관계를 기밀로 유지하면서도 더 높은 빈도의 인증과 거의 실시간에 가까운 준비금 모니터링을 지원합니다. 시장 참여자는 검증 가능한 증명을 통해 신뢰를 확보하고, 발행자는 거래 전략을 보호하며, 감사인은 자동화된 컴플라이언스 파이프라인에 직접 통합할 수 있는 휴대 가능하고 취소 인지형(revocation-aware) 인증을 활용할 수 있습니다.

4.3 프라이버시 보존 신원 및 위임

에이전트는 신원에 민감한 맥락에서 사용자를 대신해 행동하는 경우가 많습니다. 신원 에이전트는 zkTLS 기반 API 캡처와 선택적 술어 증명을 결합하여 정책상 필요한 결과만을 도출합니다(예: 특정 기준 이상의 연령, 관할권 적격성, 또는 KYC 검증 통과). 이후 이를 최신성, 범위, 선택적 취소 메타데이터를 포함한 인증으로 구성합니다. 의존 당사자는 원시 문서나 기저 개인정보에 접근하지 않고도 이러한 인증을 검증할 수 있습니다.

이 모델은 반복적인 신원 확인 마찰을 줄이고, 사용자가 자신의 에이전트에 대해 제한적이고 시간 기반의 동의를 부여할 수 있게 하며, 개인화된 온보딩, 자동 권한 검증, 컴플라이언스 게이팅을 포함한 다양한 에이전트 서비스를 가능하게 합니다. 동시에 불필요한 데이터 노출을 방지합니다.

4.4 에이전트가 조율하는 온/오프 램프

에이전트는 지불자 프라이버시를 유지하면서 검증 가능한 정산 증명을 생성하는 다중 레일의 법정화폐-암호화폐 및 암호화폐-법정화폐 흐름을 조율할 수 있습니다. 전형적인 흐름에서 구매자 에이전트는 선택된 레일에서 결제를 시작하고, 결제 인증 에이전트는 인증된 API 를 통해 정산 상태를 모니터링하며 필요한 정보만을 포함한 zkTLS 기반 인증을 발행합니다. 이후 검증 에이전트 또는 zkVM 이 해당 인증을 검증하고 온체인 에스크로에서 토큰 해제를 실행합니다.

이러한 구조는 다양한 결제 레일과 관할권에 걸쳐 확장 가능한 비수탁형이며 감사 가능한 정산을 가능하게 합니다. 사용자는 경로 선택, 정산 대사(reconciliation), 컴플라이언스 단계를 에이전트에 위임하면서도, 입증 가능한 정산 보장과 최소한의 정보 공개를 유지할 수 있습니다.

4.5 기관 에이전트를 위한 기밀 DeFi 워크플로우

기관 에이전트는 FHE, 기밀 실행 레이어, 검증 가능한 인증을 결합함으로써 전략과 고객 데이터를 기밀로 유지한 채 DeFi 에 참여할 수 있습니다. 에이전트는 암호화된 주문장, 포지션 벡터, 정책 제약을 제출하고, 실행 레이어는 암호문 위에서 직접 네팅, 가격 산정,

위험 계산, 컴플라이언스 검사를 수행한 뒤, 정책 준수를 입증하는 인증과 함께 암호화된 정산 결과를 반환합니다.

대표적인 응용으로는 프라이빗 매칭, 다크풀 실행, 양자 간 대출을 위한 기밀 신용 평가, 주문 흐름을 보호하는 암호화된 AMM 유동성 공급 등이 있습니다. 이러한 워크플로우는 정보 유출과 선행 매매를 완화하고, 기관이 프라이버시와 감사 가능성을 유지한 상태에서 실행을 에이전트에 위임할 수 있도록 합니다.

4.6 에이전트를 통한 소셜 및 신원 기반 결제

에이전트는 발신자와 수신자를 대신해 신원 검증과 정산 과정을 추상화함으로써 소셜 식별자 기반 결제를 단순화합니다. 발신자 에이전트는 신원 약속(identity commitment) 또는 시간 잠금 청구를 기준으로 자금을 잠그고, 수신자 에이전트는 해당 식별자 또는 자격 증명에 대한 통제권을 zkTLS 기반 인증으로 증명한 뒤 검증된 지갑으로 이를 상환합니다. 필요 시 이의 제기 기간과 다중 인증자 검증을 포함할 수 있습니다.

이 구조는 Web2 사용자의 온보딩 마찰을 줄이고, 신원-지갑 매핑 오류를 감소시키며, 창작자 경제 및 국경 간 송금 등 다양한 시나리오에서 확장 가능한 신원 기반 결제를 가능하게 합니다. 동시에 개인 식별 정보는 온체인에 노출되지 않고 보호됩니다.

요약. 이러한 시나리오 전반에서 Primus 는 에이전트에게 검증 가능한 행동과 프라이버시를 보존하는 데이터 접근을 제공합니다. 이 조합은 사용자가 안심하고 위임할 수 있게 하고, 플랫폼이 에이전트 권한을 안전하게 확장하며, 시장이 더 깊은 유동성과 더 넓은 서비스를 실현할 수 있는 기반을 형성합니다.

5 경제 모델

이 섹션에서는 Primus 네트워크의 네이티브 경제 모델과, 네트워크 유틸리티, 분배 구조, 가치 포착 메커니즘을 포함한 PRIM 토큰의 설계 근거를 요약합니다.

5.1 PRIM 이란 무엇인가

PRIM 은 Primus Network 의 네이티브 유틸리티 및 거버넌스 토큰으로, 인증된 Web2 데이터 검증과 프라이버시 보호 연산을 결합한 탈중앙화 검증형 기밀 컴퓨팅 인프라를 기반으로 합니다.

PRIM 은 네트워크 내에서 핵심적인 조정 메커니즘으로 작동하며, 검증 가능한 연산에 대한 결제, 인증 노드 지원, 거버넌스 참여, 그리고 운영자, 개발자 및 기업 사용자 간의 활동 조정을 가능하게 합니다.

PRIM 은 노드 참여, 작업 수행 자격, 네트워크 안정성을 지원하기 위한 프로토콜 수준의 스테이킹 메커니즘에 활용될 수 있습니다. 참여자는 네트워크 운영에 대한 기여도 및 작업의 정확한 수행에 따라 프로토콜에 의해 정의된 인센티브를 받을 수 있습니다.

Important Notice: PRIM 은 Primus Network 내에서의 기능적 사용을 목적으로 설계된 유틸리티 토큰입니다. 이는 증권, 금융상품 또는 투자 상품을 의도하지 않으며, 소유권, 수익, 배당 또는 자산에 대한 권리를 제공하지 않습니다. 또한 토큰의 기능은 시간에 따라 변경될 수 있으며, 각 관할 지역의 규제 요건에 따라 달라질 수 있습니다.

5.2 토큰 공급 및 분배

- 총 공급량: 1,000,000,000 PRIM
- 토큰 표준: BEP-20(BNB Chain) 및 ERC-20(Base)
- 토큰 발행 및 유통량은 아래에 설명된 베스팅 일정에 따라 결정됩니다.

초기 토큰 할당:

- **생태계 (33%):** 인프라 인센티브, 검증 노드 참여, 연구개발, 애플리케이션 통합, 전략적 파트너십 및 프로토콜 확장 등을 포함한 네트워크 수준 인센티브와 장기적인 생태계 성장을 위해 사용됩니다.
- **커뮤니티 (18%):** 개발자 프로그램, 커뮤니티 참여, 사용자 온보딩 및 생태계 참여 확대를 통해 커뮤니티 성장과 개발자 채택을 지원하기 위해 배정됩니다.
- **초기 투자자 (16.81%):** 엔젤, 시드 및 시리즈 A 투자자를 포함한 전략적 및 기관 투자자에게 배정되며, 초기 개발 및 네트워크 구축을 지원합니다.
- **핵심 기여자 (18%):** 프로토콜 개발 및 유지보수를 담당하는 엔지니어, 연구원 및 운영 인력을 포함한 핵심 기여자를 위해 배정됩니다.
- **에어드롭 및 커뮤니티 분배 (4.49%):** TGE 에어드롭, 해커톤 기여자, 초기 사용자, Points 및 Passport 참여자 등을 포함한 커뮤니티 분배 프로그램을 통해 초기 채택과 생태계 성장을 지원하기 위해 사용됩니다.
- **유동성 (3%):** 시장 안정성과 효율적인 토큰 거래를 지원하기 위한 유동성 공급에 사용됩니다.
- **자문단 (6.7%):** 암호학, 연구, 전략 수립 및 글로벌 시장 확장을 지원하는 자문단에 배정됩니다.

5.3 베스팅 및 토큰 락업 일정

- **생태계:** 전체 토큰 공급량의 4%는 제네시스 시점에 언락되며, 나머지 토큰은 TGE 이후 48 개월 동안 선형적으로 베스팅됩니다.
- **커뮤니티:** 전체 토큰 공급량의 2%는 TGE 시점에 언락되며, 나머지 토큰은 TGE 이후 48 개월 동안 선형적으로 베스팅됩니다.
- **초기 투자자:** 배정 시점으로부터 12 개월 클리프 적용 후, 이후 36 개월까지 매월 선형적으로 베스팅됩니다.
- **핵심 기여자:** 배정 시점으로부터 12 개월 클리프 적용 후, 이후 36 개월까지 매월 선형적으로 베스팅됩니다.
- **에어드롭 및 커뮤니티 분배:** 전체 토큰 공급량의 4%는 TGE 시점에 언락되며, 나머지 토큰은 TGE 이후 12 개월 동안 선형적으로 베스팅됩니다.
- **유동성:** 100% TGE 시점에 언락됩니다.
- **자문단:** 배정 시점으로부터 12 개월 클리프 적용 후, 이후 36 개월까지 매월 선형적으로 베스팅됩니다.

TGE 시점에는 전체 토큰 공급량 중 일부만 유통되며, 나머지 물량은 상기 베스팅 일정에 따라 순차적으로 유통됩니다.

5.4 PRIM 가치 포착 및 유틸리티 PRIM 의 가치 및 활용

PRIM 은 Primus Network 내에서 네트워크 활동을 조정하고 서비스 접근을 가능하게 하는 데 사용됩니다.

- **연산 비용 지불:** PRIM 은 검증 및 연산 작업에 대한 비용 지불에 사용될 수 있으며, 클라이언트와 네트워크 노드 간의 상호작용을 지원합니다.
- **API 접근:** PRIM 은 개발자가 네트워크 서비스 및 인프라에 접근하는 데 사용될 수 있습니다.
- **참여 메커니즘:** PRIM 은 노드 참여, 서비스 안정성 및 네트워크 보안을 지원하기 위한 프로토콜 수준의 메커니즘에 활용될 수 있습니다.
- **거버넌스 참여:** PRIM 보유자는 제안, 투표 및 프로토콜 파라미터 조정 등 거버넌스 과정에 참여할 수 있습니다.
- **프로토콜 인센티브 및 안정성:** PRIM 은 네트워크 참여를 지원하고, 프로토콜 메커니즘을 통해 비활성 또는 악의적인 행위를 억제하는 데 사용될 수 있습니다.

5.5 경제적 역할 및 참여자

Primus 경제는 네 가지 주요 참여자로 구성됩니다.

- **노드:** 네트워크에 연산 자원과 인프라를 제공하는 참여자입니다. 노드 참여를 위해 일정 수준의 PRIM 스테이킹이 요구될 수 있습니다. 노드는 증명 생성 및 기밀 연산을 포함한 검증 및 연산 작업을 수행합니다.
- **개발자:** 개발자는 네트워크의 API 및 SDK 에 접근하여 검증 및 프라이버시 보호 연산 기능을 활용한 애플리케이션을 개발합니다.
- **클라이언트:** 클라이언트는 검증 가능한 증명 및 프라이버시 보호 연산이 필요한 작업을 네트워크에 요청합니다.
- **토큰 보유자:** 토큰 보유자는 프로토콜 규칙에 따라 거버넌스 및 기타 네트워크 기능에 참여할 수 있습니다.

5.6 지속 가능성 및 성장

분배 및 베스팅 구조는 네트워크 보안과 개발자 채택을 촉진하는 동시에, 장기적인 토큰 공급을 네트워크 활동과 정렬하는 것을 목표로 합니다. 생태계 자금은 기밀 연산에 대한 실제 수요에 맞춰 점진적으로 방출되어 인플레이션 압력을 완화하면서 지속 가능한 성장을 지원합니다.

6 로드맵

Primus 는 외부 데이터에 대한 검증 가능한 접근을 위한 zkTLS 와 기밀 연산을 위한 FHE 를 결합하여 에이전트 경제를 위한 신뢰 및 프라이버시 레이어를 구축하고 있습니다. 아래 로드맵은 기초 인프라에서 생태계 확장, 그리고 AI 에이전트 기반 애플리케이션을 위한 장기 거버넌스에 이르기까지 네트워크의 발전 경로를 보여줍니다.

지난 마일스톤

- **2024 년 4 분기 — Primus 핵심 네트워크 완성:** zkTLS 와 FHE 를 갖춘 탈중앙화 검증 및 연산 인프라를 구축하여 AI 에이전트를 위한 신뢰 가능한 신원 및 프라이버시 보존 기능을 제공했습니다.
- **2025 년 2 분기 — 초기 MVP 출시:** zkCredential, 프로그래밍 가능한 소셜 결제, 검증 규칙 기반 홍바오, 준비금 증명(PoR), 기밀 거래를 출시하여 AI 에이전트의 자율적 검증 및 실행을 가능하게 했습니다.
- **2025 년 4 분기 — 개발자 도구 배포:** API, SDK, DevHub 를 배포하여 생태계 통합을 지원하고, 제 3 자 애플리케이션이 에이전트 네트워크와 상호작용할 수 있도록 했습니다.

향후 마일스톤

- **2026 년 2 분기 — 네트워크 배포 및 최적화:** 탈중앙화 검증 및 연산 네트워크를 전면 배포하고, 프라이버시 보존 스마트 계약 및 규정 준수형 기밀 온체인 전송을 위한 zkTLS 와 초기 FHEVM 애플리케이션을 최적화하여 AI 에이전트를 위한 검증 가능한 실행을 지원합니다.
- **2026 년 3 분기 — 노드 온보딩 및 초기 애플리케이션 채택:** PRIM 스테이킹 및 보상 메커니즘을 통해 노드 운영자를 온보딩하고 인센티브를 제공하며, AI 및 핀테크 애플리케이션의 초기 채택을 지원하여 에이전트가 자율적으로 행동과 결제를 실행할 수 있도록 합니다.
- **2026 년 4 분기 — 첫 번째 생태계 애플리케이션 물결:** 프라이버시 보존 AI 데이터 학습, 온체인 검증 가능 투표, 기밀 결제를 출시하여 실제 환경에서 에이전트의 신뢰 가능하고 프라이버시를 보존하는 실행을 입증합니다.
- **2027 년 상반기 — 커뮤니티 및 개발자 확장:** 해커톤, 개발자 인센티브, 제 3 자 애플리케이션 통합을 위한 생태계 그랜트를 확대하여 에이전트 생태계 성장을 촉진합니다.
- **2027 년 하반기 이후 — 지속 가능한 생태계 및 거버넌스:** 프라이버시 데이터 마켓플레이스를 구축하고 Web3 및 핀테크 전반으로 기업 채택을 확대하며, 온체인 거버넌스를 강화하고 개발자, 노드, 사용자에 대한 지속 가능한 토큰 인센티브를 유지함으로써 장기적인 AI 에이전트 경제 성장을 지원합니다.

요약

Primus 는 이미 기초 인프라, 초기 제품, 개발자 도구를 구축했습니다. 다음 단계는 네트워크를 확장하고 애플리케이션 채택을 확대하며, 지속 가능한 에이전트 경제를 위한 거버넌스 및 인센티브 체계를 구축하는 데 초점을 맞춥니다.

References

[1] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, August 2008.

[2] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, August 2018.

[3] Fan Zhang, Zeyu Liu, Zhi-Jian an, Sanket Kanjalkar, and Ari Juels. DECO: liberating web data using decentralized oracles for TLS. In Proceedings of the 2020 ACM

SIGSAC Conference on Computer and Communications Security, CCS '20, Virtual Event, USA, November 9-13, 2020, pages 1447-1466, 2020.

[4] Xiang Xie, Kang Yang, Xiao Wang, and Yu Yu. Lightweight authentication of web data via garble-then-prove. Cryptology ePrint Archive, Paper 2023/964, 2023. https://eprint.iacr.org/2023/964

[5] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. Cryptology ePrint Archive, Paper 2021/076, 2021.

[6] Xiang Xie and Xiao Wang. Unearthing the reality of zktls: A benchmarking and cryptanalysis report. HackMD, 2023. URL: https://hackmd.io/X-NSD1UtQoC66aPOyGo5NA

[7] Primus. https://primuslabs.xyz/

[8] TLSNotary. https://tlsnotary.org/

[9] Reclaim. https://www.reclaimprotocol.org/

[10] Pluto. https://pluto.xyz/

[11] Primus. https://github.com/primus-labs/zktls-bench/tree/main

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Leveled fully homomorphic encryption without bootstrapping. In Proceedings of ITCS, 2012. Foundational lattice-based leveled FHE constructions (BGV).

[13] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, 2012/144, 2012. BFV-style scheme optimized for integer arithmetic.

[14] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In ASIACRYPT 2017, 2017. Introduces CKKS for efficient approximate arithmetic.

[15] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors and the gsw framework. IACR Cryptology ePrint Archive, 2013. Matrix/GSW-style ciphertexts enabling gate evaluation and bootstrapping.

[16] Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, Advances in Cryptology - EUROCRYPT 2015, Lecture Notes in Computer Science, pages 617-640. Springer, 2015.

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Tfhe: Fast fully homomorphic encryption over the torus. IACR Cryptology ePrint Archive, 2016/421, 2016. Practical gate-bootstrapping FHE library and algorithmic improvements.

[18] Fengrun Liu, Haofei Liang, Tianyu Zhang, Yuncong Hu, Xiang Xie, Haisheng Tan, and Yu Yu. HasteBoots: Proving FHE bootstrapping in seconds. Cryptology ePrint Archive, Paper 2025/261, 2025.

[19] Louis Tremblay Thibault and Michael Walter. Towards verifiable fhe in practice: Proving correct execution of tfhe's bootstrapping using plonky2. ePrint, 2024.

[20] The RISC Zero Team. Universal zero knowledge. https://risczero.com/

[21] The SuccinctLabs. The fastest, most feature-complete zkvm for developers. https://github.com/succinctlabs/sp1