영지식 증명 소개

작성: GIST 블록체인 인터넷 경제 연구센터 장재혁 연구원 (센터장 이흥노 교수)

작성일: 2020-09-20

내용

영지식 증명?

영지식 증명(zero-knowledge proof)은 증명 프로토콜로써 증명자가 검증자에게 어떤 명제(statement)가 참임을 납득시키는 프로토콜이다. 이때, 영지식 증명 프로토콜은 검증자에게 그 명제가 참 혹은 거짓이라는 사실 외의 다른 정보는 일절 전달하지 않는 영지식(zero-knowledge) 특성을 가져야 한다.

예를 들어 출입구가 단 하나뿐인 원형 동굴을 상상해보자. 출입구 반대편의 한 지점에는 비밀번호를 필요로 하는 잠금 장치가 있는 문이 있다. 즉, 입구에서 출발하여 원형 동굴을 한 바퀴 일주한 후 출구로 나오기 위해서는 잠금 장치의 비밀번호를 알아야 한다. 이 때, "나는 잠금 장치의 비밀번호를 알고 있다"라는 명제를 타인에게 증명해야하는 상황을 가정하자. 영지식 증명은 잠금 장치의 비밀번호를 알려주지 않으면서 위의 명제가 참임을 납득시키는 프로토콜이다. 이러한 문제를 알리바바 동굴 (The Ali Baba cave) 문제라 한다 (그림 1).

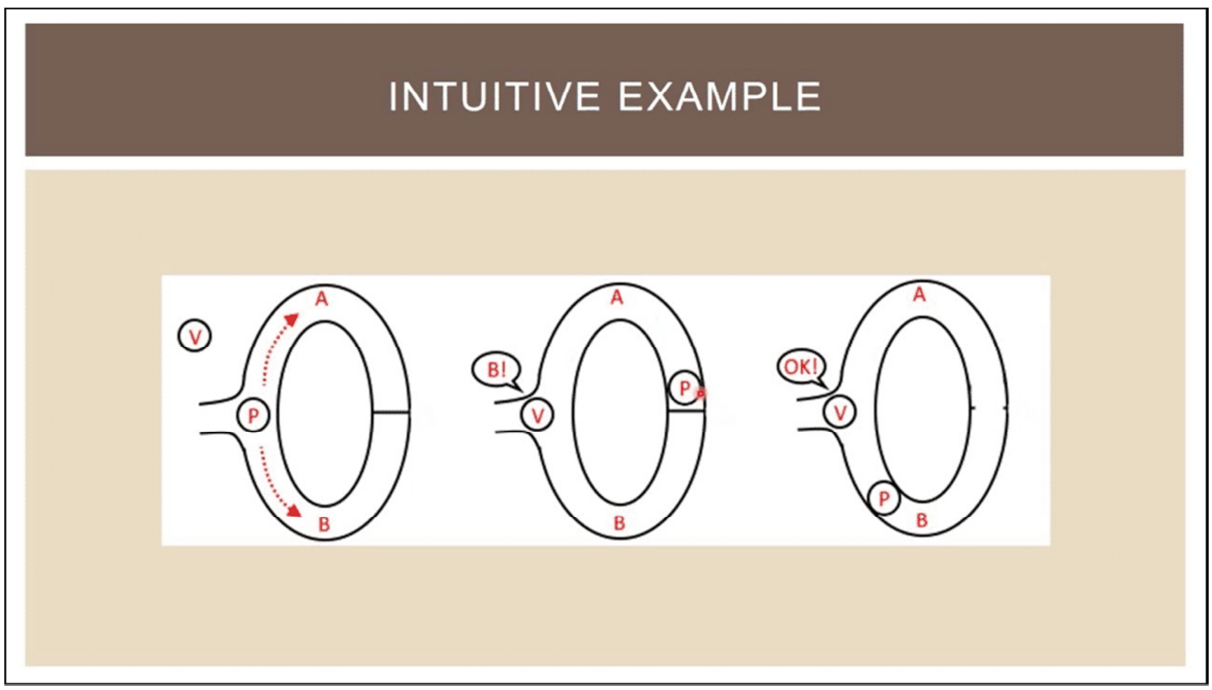


그림 1 알리바바 동굴 문제

영지식성을 배제하면, 위의 예에서 검증자가 명제가 참 혹은 거짓임을 확인 할 수 있는 가장 쉬운 방법은 증명자로부터 비밀번호를 전달받은 후 잠금 장치를 열어보는 것이다. 반면 영지식 증명인 경우, 검증자가 비밀번호를 알아서는 안되기 때문에 증명이 어려워진다. 영지식 증명은 중요한 정보의 공개가 꺼려지는 상황에서 그 정보와 관련된 어떤 인증을 해야 할 때 사용 될 수 있다. 예를 들어 우리나라에서 자주 사용되는 공인인증서 검사 프로그램에 사용 될 수 있다. 영지식 증명이 적용된다면, 내 인증서의 비밀번호를 입력하지 않고도 내가 나임을 인증 할 수 있다. 공인인

증 프로그램은 사용자가 설정한 비밀번호에 보안이 크게 의존되어 있기 때문에, 이러한 응용 예는 실용적일 수 있다.

정리하면, 공개된 정보와 비공개 정보로 구성된 어떤 명제를 s 라 할 때, 어떤 증명 프로토콜이 영지식 증명 프로토콜이 되기 위한 필요충분조건 세 가지는

1. 완전성(completeness): 만약 s 가 참이면, 진실한 증명자가 검증자에게 s 가 참임을 납득시킬 수 있어야 한다.

2. 건실성(soundness): 만약 s 가 참이 아니면, 거짓 증명자가 검증자에게 s 가 참임을 납득시킬 수 없어야 한다.

3. 영지식성(zero-knowledge): 검증자는 s 가 참 혹은 거짓이라는 사실 외에, 어떤 비공개 정보도 알 수 없어야 한다. 즉, 증명자가 s 의 참/거짓에 결정적 관련이 있는 정보들 중 일부를 선택적으로 공개하지 않을 수 있어야 한다.

영지식증명 프로토콜은 증명과정과 검증과정의 두 과정으로 구분 될 수 있다. 증명과정은 증명자가 명제 s 가 참임을 스스로 검증 한 후, 그 증거를 생성하는 과정이다. 검증과정은 검증자가 증명자로부터 증거를 받아 증거에 문제가 없는지를 확인하는 과정이다. 만약 증거에 문제가 없다면, 증명자가 명제 s 의 자가검증을 잘 수행했다는 사실이 수학적으로 보장되어 있어야 한다.

영지식 증명의 활용 예: 온라인 투표 시스템

온라인 투표 시스템은 공동체 구성원들이 직접적인 대면 없이 의사를 전달하고 결정할 수 있도록 해주는 시스템이다. 이는 대표적인 대면 (오프라인) 투표시스템인 선거와 대조적이다. 선거와 같은 기존 대면 투표시스템은 완벽하지는 않지만 표 1의 특성들이 잘 지켜질 수 있도록 체계화 되어있다. 그러나 대면 투표시스템은 시간과 공간의 제약을 크게 받으며, 이로인해 발생하는 인적, 물적 자원 소요 또한 상당하다. 이에 반해 온라인 투표시스템은 경제적이며 언제, 어디서든 의사표현을 할 수 있다는 장점이 있지만, 표 1의 특성을 지키기 어렵다는 단점이 있었다. 일반적인 투표 시스템이 지녀야 할 투표의 특성은 아래 표 1과 같이 나열 할 수 있다.

정확성	모든 정당한 유효투표는 투표결과에 정확히 집계됨
확인성	투표결과 위조방지를 위한 투표결과 검증수단이 필요
완전성	부정 투표자에 의한 방해 차단, 부정투표는 미 집계
단일성	투표권이 없는 유권자의 투표참여 불가
합법성	정당한 투표자는 오직 1회만 참여 가능
기밀성	투표자와 투표결과의 비밀관계 보장
공정성	투표 중의 집계결과가 남은 투표에 영향을 주지 않음

표 1 투표 시스템의 특성

온라인 투표 시스템에서는 모든 과정에서의 산출물이 데이터로써 저장된다. 만약 데이터가 중앙 서버에 저장된다면, 데이터의 위변조가 가능하기 때문에 기술의 도움을 받을 필요가 있다. 이때 도움이 될 수 있는 기술 중 하나가 블록체인이다. 구체적으로, 투표의 특성 중 정확성과 확인성, 완전성, 합법성은 블록체인 기술로 해결 될 수 있다. 블록체인의 의의가 "위변조가 불가능하고 누구나 검증가능한 분산 서버"를 만드는 것이기 때문이다.

한편 온라인 투표의 단일성과 합법성은 digital identity (DID)[1]와 public key infrastructure (PKI)[2] 기술의 도움을 받아 만족시킬수 있다. 일반적인 대면방식의 선거에서는 선거위원이 투표자의 신원을 확인한 후 투표권 (도장이 찍힌 투표용지)을 나눠준다. 만약 온라인 투표라면, 신원확인에 필요한 신분증을 공인인증서와 같은 DID로 대체 할 수 있다. 문제는 온라인 투표권인데, 만약 투표권이 항상 변하지 않는 어떤 디지털 데이터라면, 복제되어 악용될 수 있다. 그러므로 투표권은 투표

[1] DID는 사용자가 자신이 신뢰받을 수 있는 개인 혹은 기관, 장치라는 정보가 기록된 디지털 데이터이다.

[2] PKI는 public-key encryption 기술을 활용해 증서를 발급하는 기반 시설(infrastructure)이다. PKI를 이용하여 DID를 네트워크상에서 안전하게 전송 할 수 있다.

자의 DID에 따라 변하는 데이터여야 한다. 이렇게 DID를 기반으로 투표권을 생성할 수 있는 기술의 예가 PKI이다. 중앙 선거관리 위원회가 PKI를 통해 사용자의 DID를 넘겨받은 후 고유의 투표권 데이터를 다시 사용자에게 배포 할 수 있다. 이후 투표 결과를 검표할때는 투표권 데이터가 중앙 선거관리 위원회에 의해 배포된 투표권이 맞는지 확인할 수 있다. 즉, DID와 투표권 데이터 간의 1대1 대응이 되도록 하는것이다. 만약 DID와 사용자 개인간의 완벽한 1대1 대응이 보장된다면, 단일성과 합법성이 만족 될 수 있다. 그러나 검표과정에서 어떤 DID가 어떤 의사결정을 하였는지가 중앙 선거관리 위원회에 공개되기 때문에, 기밀성을 만족시키지는 못한다.

영지식 증명은 온라인 투표 시스템이 기밀성을 만족하도록 해주는 도구가 될 수 있다. 예를 들어 어떤 투표자가 있고, 이 투표자의 DID를 x, 그리고 x에 의해 생성된 투표권 데이터를 $s(x)$라 하자. 그리고 투표권 데이터를 확인 (verification)하는 함수를 f라 하자. 즉, s가 x로부터 생성된게 맞다면 $f(x,s)=1$ 이며, 그렇지 않다면 $f(x,s)=0$ 이다. 이제 투표자가 증명해야 할 명제 S는 "$f(x,s)=1$"이며, 여기서 공개되지 않아야 할 비공개정보는 x이다. 영지식 증명 프로토콜에 의해 투표자는 명제 S를 스스로 자가검증한 후 그 증거파일 π를 생성하여 투표결과와 함께 투표함에 기록한다. 이후 검표과정에서는 π에 문제가 없는지를 확인한다. 만약 증거파일 π에 문제가 없다면, 영지식 증명의 정의에 의해 해당 투표는 정당한 투표권자에 의한 투표임을 수학적으로 보장받게 되며 (단일성 만족), DID를 유출하지 않았기 때문에 투표자와 투표결과간의 비밀관계도 보장받게 된다 (기밀성 만족).

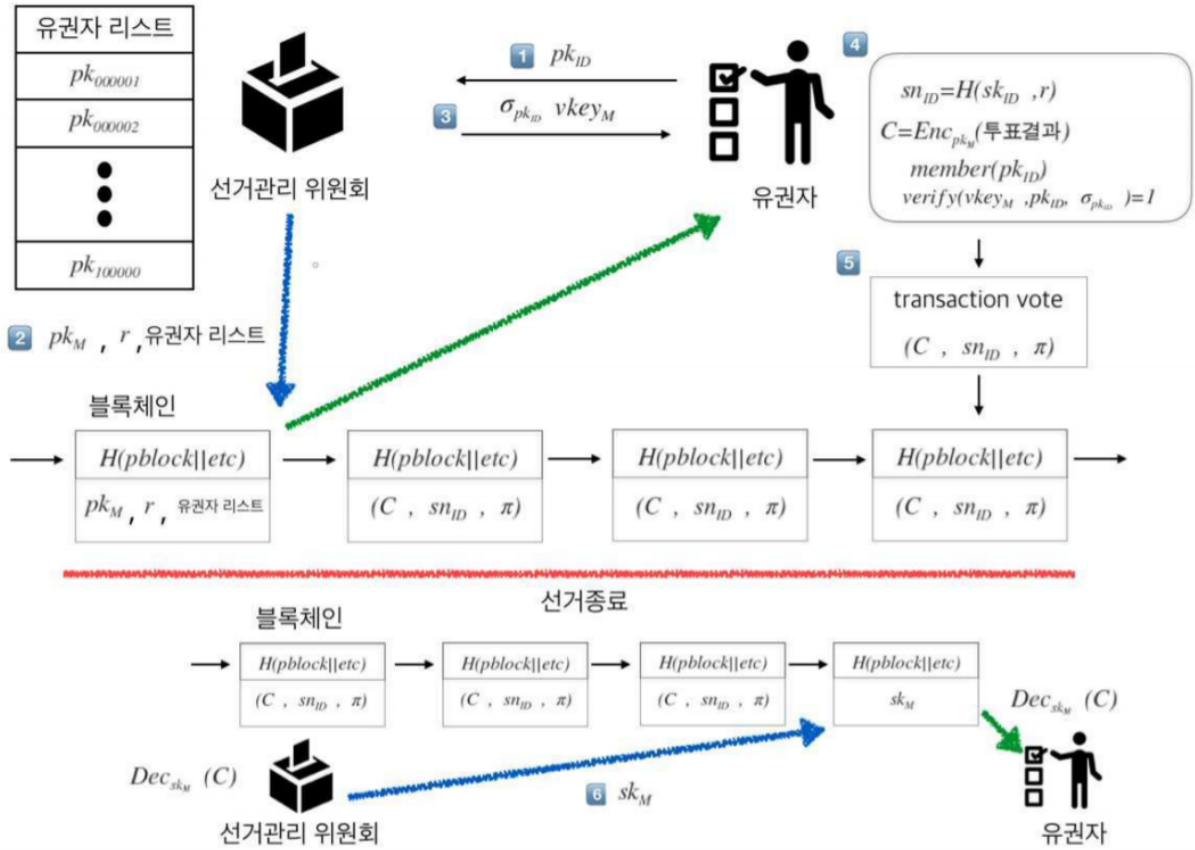


그림 2 한양대학교와 국민대학교 산학협력단이 출원한 영지식 증명이 적용된 블록체인기반 온라인 투표시스템

그림 2는 한양대학교와 국민대학교가 2018년 공개출원한 특허 "비밀 선거가 보장된 블록 체인 기반의 전자 투표를 수행하는 단말 장치 및 서버와, 전자 투표 방법"의 대표도해이다. 그림 2의 온라인 투표시스템에서, 투표자(유권자)의 DID는 pk_{ID} 이다. 선거관리 위원회가 투표자의 DID를 확인 한 후 투표자에게 투표권 데이터인 $\sigma_{pk_{ID}}$ 와 $vkey_M$ 을 준다. 투표권 데이터는 투표자의 DID에 따라 변한다. 투표자가 투표권을 보유한 자인지를 확인하는 함수는 $member(\ )$ 와 $verify(\ )$ 이다. 함수 $member$ 는 투표자의 DID가 투표대상이 맞는지 (유권자 리스트에 포함되어 있는지)를 확인하는 함수이고, 함수 $verify$ 는 투표권 데이터가 투표자의 DID로부터 올바르게 생성된 데이터인지를 확인하는 함수이다.

만약 영지식증명이 적용되지 않은 투표시스템이라면, 두 함수 $member$ 와 $verify$ 는 검표과정에서 검표자 (선거관리 위원회)가 수행해야 할 함수이다. 반면 그림 2과 같이 영지식증명이 적용된 투표시스템에서는 두 함수를 투표자가 스스로 수행하여 자가검증한다. 그 후 자가검증을 잘 수행하였다는 증거 데이터 π 를 자신의 투표 결과와 함께 블록체인에 기록한다. 투표가 끝난 후, 검표자는 증거 데이터 π 에 오류가 없는지 확인함으로써 정당한 투표권자에 의한 투표임을 확인한다.

그림 2과 같은 투표 시스템은 온라인 투표의 실현 가능성을 제고하였으나, 아직 완벽하다고 할

수는 없다. 그 이유는 투표 시스템이 선거관리 위원회라는 신뢰받는 제 3자(trusted third-party)에 의존하고 있기 때문이다. 믿음을 기반으로 동작하는 선거관리 위원회가 타락하면 부정투표가 가능할 수 있다. 예를 들어 선거관리 위원회는 투표자들의 DID들과 각각에 대응되는 투표권 데이터들을 모두 보관하고 있을 수 있다. 이는 곧 선거관리 위원회가 타인의 투표권을 사용 할 수 있다는 의미이다. 만약 선거관리 위원회가 컴퓨터 프로그램이고 서버라면, 해커에 의해 모든 기록들이 탈취 될 가능성 또한 배제할 수 없다. 뿐만 아니라, 선거관리 위원회가 임의로 특정 투표를 누락시킬 수 도 있다.

비록 아직 완벽하지는 않지만, 그림 2과 같이 영지식 증명이 적용된 투표 시스템의 제안은 실용가능성을 보여주었다. 실제로 영지식 증명을 적용하여 기밀성을 확보한 또다른 투표 시스템이 스위스의 업체 Luxoft와 스위스 루체른 대학의 공동연구로 2018년에 개발되었으며, 스위스 Zug시의 주관 하에 72명의 시민을 대상으로 시범투표가 시행되었다. 자세한 내용은 https://www.stadtzug.ch/newsarchiv/615796 에서 확인 할 수 있다.

대화형(Interactive) 영지식 증명, NIZK, 그리고 zk-SNARK:

zk-SNARK는 zero-knowledge succint non-interactive argument of knowledge의 약자이다. zk-SNARK는 이전에 정의된 일반적인 영지 증명 프로토콜의 세 가지의 필요충분조건에서 "간결성(succintness)"과 "비대화형(no interaction)"의 조건이 추가된 발전된 형태의 영지식 증명이다.

영지식 증명의 succintness는 증거의 간결함, 즉 증명과정에 의해 산출되는 증거 데이터의 물리적 크기가 작음을 의미한다. 정확한 설명은 아니지만, 일반적으로 증거의 크기는 증명해야 할 명제의 복잡도에 따라 커진다. 예를 들어 어떤 함수 f 와 값 x 에 대해 증명해야 할 명제가 "$f(x)=1$" 인 경우, f 의 계산 복잡도가 크면 증거의 크기도 커질 수 있다. 연구결과에 따라 다르지만, 간결성이 추가된 영지식 증명 프로토콜이 지향하는 것은 증거의 크기가 명제의 복잡도에 관계 없이 고정되어 있거나, 혹은 명제의 크기가 명제의 복잡도와 선형적인 관계 혹은 그 이하에 있는 프로토콜이다.

증거의 크기가 간결할 때의 이점은 검증자가 검증을 빠르게 끝낼 수 있다는 것이다. 증거의 크기가 작기 때문에, 검증자가 증거의 오류를 검사하기위해 소요해야하는 필요 계산량도 작아지게 된다. 이 뿐만 아니라, 증거를 게시해야 할 서버의 용량이 한정 된 경우에도 유용하다. 그러한 서버의 예가 블록체인인데, 블록의 크기가 제한적이기 때문이다. 따라서 증거의 크기가 크면 블록에 기록될 수 있는 증거의 수가 줄어든다. 이는 시간당 블록체인을 사용할수 있는 사용자의 수가 줄어드는것과 같다. 즉, 블록체인의 확장성(scalibility)이 줄어드는 것이다. 반대로, 증거의 크기가 작아질수록 같은 시간동안 더 많은 사용자가 블록체인을 사용 할 수 있게 되며, 따라서 블록체인의 확장성이 커진다.

비대화형 조건을 다루기 전에 먼저 고전적 형태인 대화형 (interactive) 영지식 증명에 관해 알아볼 필요가 있다. 대화형 영지식 증명의 특징은 증명자와 검증자가 함께 증거를 만드는 것이다. 먼저 증명자가 증거의 일부를 생성하고, 검증자가 증명자에게 도전적 문제 (challenge)를 제출한다. 도전적 문제란 증명자는 비공개 정보를 알지 못하면 응할 수 없는 문제를 말한다. 최종적으로 증명자는 스스로 생성한 일부의 증거와 검증자의 도전적 문제에 답하는 증거를 합쳐 최종 증거를 검증자에게 제출한다. 대화형 영지식 증명의 가장 쉬운 예는 알리바바 동굴 (The Ali Baba cave) 문제[3]이며, 더욱 실용적인 예는 Schnorr 인증 (identification) 프로토콜이다. 알리바바 동굴에 관한 해설은 https://blockgeeks.com/guides/what-is-zksnarks/ 에 친절하게 설명되어 있다. 그리고 이러

[3] Quisquater, Jean-Jacques; Guillou, Louis C.; Berson, Thomas A, "How to Explain Zero-Knowledge Protocols to Your Children," 1990

한 대화형 영지식 증명 프로토콜의 한 일반화로써 시그마(∑) 프로토콜이 있다[4].

비대화형(Non-interactive) 영지식 증명(NIZK, non-interactive zero knowledge)은 이름 그대로 증거 생성과정에서 검증자와 증명자의 대화가 필요 없는 증명을 의미한다. 즉, 증명자 혼자서 증거를 생성하는 것이다. 앞서 다룬 대화형 영지식 증명에서는 증거 생성과정에서 증명자가 검증자의 도전적 문제 (challenge)를 받고 이에 응해야 하기때문에 둘 간의 "대화 (conversation)"가 필요했다. 이 대화 과정을 생략하면서 그 역할을 유지하기 위해 제안된 방법 중 하나가 Fiat-Shamir heuristic[5]이다. Fiat-Shamir heuristic은 "random oracle" 이라는 이상적인 랜덤 프로그램의 존재를 가정하여 검증자의 도전적 문제를 대체한다. 다시 말해, 증명자 스스로 도전적 문제를 랜덤하게 생성 한 후 이를 공개해 누구나 납득할 수 있게 하는것이다.

NIZK의 이점은 증명 과정에서 시간적 제약이 사라지는 것이며, 따라서 증명 프로토콜이 블록체인에 적용되기 적합하다. 대화형 영지식 증명의 경우 증명자와 검증자간의 대화가 필요했기 때문에, 증명을 완료하기 위해서는 두 사람이 같은 시간대에 서로 약속된 온라인 공간에 함께 접속해야하는 시공간적 제약이 있었다. NIZK는 더 이상 대화과정이 필요 없기 때문에, 증명자가 원하는 시간과 장소에 증거 파일을 업로드 해 놓으면, 검증자가 언제든 파일을 다운받아 검증을 수행 할 수 있다. No-interaction 특성은 특히 영지식 증명이 블록체인에 적용될 때 반드시 필요하다. 블록체인은 그 정의에 의해, 사용자가 데이터 업로드를 요청하면, 채굴자 혹은 관리자가 요청된 데이터를 모은 후 어느정도의 시간이 지난 후에 검증 및 업로드를 완료한다. 즉, 사용자가 증명자라면, 증거를 업로드 한 후 언제가 될 지 모를 시간 후에 검증자가 검증을 수행한다는 의미이다. 증명자가 NIZK 프로토콜을 따른다면 그 시간을 온라인상에서 기다리고 있을 필요가 없다.

---

[4] Ivan Damg˚ard, "On On Σ-protocols," *CPT*, 2010, online (https://www.cs.au.dk/~ivan/Sigma.pdf).

[5] https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir_heuristic

대화형 영지식 증명의 예: Schnorr 프로토콜

Schnorr 프로토콜은 대표적이며 단순한 대화형 영지식 증명 프로토콜이다. 증명하고자 하는 명제는 "공개된 값 y 와 공개된 값 $g \in F_p^*$ 에 대하여[6], $y = g^x \bmod p$ 을 만족시는 비공개 값 x ($0 \le x < p-1$)를 알고 있음"을 납득시키는 상황이다. 여기서 p 는 공개된 값이며 소수(prime number)이다. 그리고 g 는 F_p의 primitive element[7]이다. 이 글에서 다룰 Schnorr 프로토콜은 다음과 같다:

프로토콜 1. Schnorr 프로토콜

1) 증명자는 하나의 (비공개) 무작위 값 v $\left(0 \le v < p-1\right)$로 $t = g^v \bmod p$를 계산 하여 검증자에게 보낸다.
2) *도전적 문제:* 검증자는 하나의 무작위 값 c $\left(0 < c < p-1\right)$를 증명자에게 보낸다.
3) 증명자는 $r = v - cx \bmod \left(p-1\right)$를 계산하여 검증자에게 보낸다.
4) 검증자는 $t \equiv g^r y^c \bmod p$ 임을 확인한다.

프로토콜 1에서, 증명자는 검증자에게 x를 직접적으로 전송 하지 않는다. x를 사용한 계산 결과 r을 전송하긴 하지만, r로부터 x를 알아내는 것은 매우 어렵다. 그럼에도 불구하고 검증자는 $t \equiv g^r y^c \bmod p$를 계산함으로써 증명자가 올바른 x를 사용하였는지 확인 할 수 있다. 다음의 분석을 통해 그 이유를 설명하겠다.

Schnorr 프로토콜의 영지식증명 성능 분석

위 프로토콜 1이 영지식 증명의 정의를 만족하는지를 확인하면 다음과 같다.

Q. (완전성) 만약 증명자가 x를 알고있다면, 검증자는 $t \equiv g^r y^c \bmod p$를 확인함으로써 이를 납득할 수 있는가?

A. 그렇다.

---

[6] 집합 F_p는 특성(characteristic)이 p인 유한 체(finite field)이다. $F_p := \{0,1,\cdots,p-1\}$ 이고, 모든 원소간의 더하기 및 곱하기 연산에는 $\bmod p$ 가 적용된다. F_p^*는 F_p에서 원소 0을 제외한 집합이다.

[7] 어떤 원소 $a \in F_p$가 primitive element이면, a의 제곱으로 구성된 집합, 즉, $\left\{a^i, \forall i \in \mathbb{Z}_0^+\right\}$의 원소의 개수는 $p-1$이다. 관련 정보: https://en.wikipedia.org/wiki/Primitive_element_(finite_field)

Proof) 만약 증명자가 과정 3)에서 r을 계산 할 때 사용한 값 x가 $y = g^x \bmod p$를 만족시킨다면, $g^r y^c \equiv g^{v-cx \bmod (p-1)} g^{cx} \bmod p$이다. Fermat's little theorem[8]에 의해, $g^{cx} \equiv g^{cx \bmod (p-1)} \bmod p$이고, 따라서 $g^r y^c \equiv g^v \equiv t \bmod p$이다.

Q. (건실성) 만약 거짓 증명자가 x를 모른다면, 프로토콜을 1회 실행함으로써 검증자에게 $t \equiv g^r y^c \bmod p$임을 납득시킬 수 있는가?

A. 그럴 수 없다.

Proof) 거짓 증명자가 주장하는 값을 x'라 하자. 여기서 $x' \neq x$이다. 거짓 증명자가 검증자를 속이기 위해 목표하는 바는 합동방정식 $t \equiv g^{r'} y^c \bmod p$를 만족시키는 값 r'을 찾는것이다. 여기서 y^c와 t의 값은 고정되어 있으며 증명자에게 알려져 있으므로, 결국 증명자가 풀어야 할 문제는 $\left(y^c\right)^{-1} t \equiv g^{r'} \bmod p$를 만족시키는 값 r'을 찾는 것 이다. 즉, 과정 3)의 계산을 무시하고, 이 합동방적식의 해를 찾아 검증자에게 건네야 한다.

증명자가 풀어야 할 문제는 discrete logarithm 문제이다. 이는 logarithm 연산을 유한 체에서 수행하는 것으로, 실수 집합에서 수행 할 때 보다 계산이 훨씬 어렵다. 거짓 증명자가 택할 수 있는 다른 방법은 F_{p-1}의 원소들 중 무작위로 r'를 선택하여 그 값이 조건을 만족하는지 확인 하는 것이다. 한번 무작위 값을 선택하여 조건이 달성 될 확률은 $(p-1)^{-1}$이다. 일반적으로 p는 아주 큰 숫자이므로, 거짓 증명자가 검증자를 납득시킬 확률은 아주 작다.

Q. (약한 영지식성) 프로토콜 실행을 1회 요청함으로써 검증자는 r로부터 x를 알아 낼 수 있는가?

A. 그럴 수 없다.

Proof) 검증자가 건네받는 $r = v - cx \bmod (p-1)$에서, 검증자가 알고 있는 값은 r, p, c, 그리고 알지 못하는 값은 v와[9] x이다. 탈취함수를 $f : F_{p-1} \times F_{p-1} \to F_{p-1}$, 여기서 $f(v', x') = v' - cx' \bmod (p-1)$이라 정의하겠다. 검증자가 x를 알아내는 행위는 $f^{-1}(r)$의 원소를 찾는것과 동등하다. 그러나 함수 f의 정의역의 원소의 개수는 $(p-1)^2$이고 공역의 원소의 개수는 $p-1$이기 때문에, 함수 f는 injective이다. 즉, inverse image $f^{-1}(r)$의 원소는 유일하지 않다. 검증자는 더 이상의 다른 정보가

---

[8] Fermat's little theorem: 서로소인 a와 p에 대해, $a^{p-1} \bmod p \equiv 1$. 관련 정보: https://en.wikipedia.org/wiki/Fermat%27s_little_theorem

[9] 검증자가 건네받은 또 다른 값인 t는 v로부터 생성된 값인데, t에서 거꾸로 v를 찾는 것은 discrete logarithm 문제로써 확률이 $(p-1)^{-1}$으로 매우 낮고, 계산적으로도 매우 어렵다.

없기 때문에 r로부터 x를 알아 낼 수 없다.

Q. (강한 영지식성) 프로토콜 실행을 다회 요청함으로써 검증자는 r로부터 x를 알아 낼 수 있는가?

A. 가능하다.

Proof) 검증자가 증명자에게 프로토콜 실행을 다회 재요청함으로써 r로부터 x를 알아 내는 다양한 방법이 존재 할 수 있다. 그 중 한가지 예를 들겠다. 프로토콜을 i번째 실행 하였을 때 프로토톨상에서 생성된 비공개 값을 v_i, 공개된 값들을 r_i, c_i, t_i라 하겠다. 같은 증명자가 반복 실행하므로, 비공개 값 x는 변하지 않음을 가정한다. 다음의 알고리즘 2는 프로토콜 1을 반복 실행하여 검증자가 증명자의 비공개 정보를 탈취하는 알고리즘이다:

알고리즘 2. Schnorr 프로토콜에 사용 될 수 있는 비공개 정보 탈취 알고리즘

1. 검증자는 매 i번째 실행마다 증명자로부터 건네받은 (t_i, r_i, c_i)쌍을 저장해 둔다.
2. 매 실행이 끝날 때 마다, 자신이 저장해 놓은 데이터로부터 $t_i t_j \equiv 1 \bmod p$ 이거나 혹은 $t_i \equiv t_j \bmod p$ 이 되도록 하는 서로다른 index 쌍 i, j가 존재하는지 확인하고, 만약 존재한다면 프로토콜 재실행 요청을 멈춘다.
3. 만약 $t_i t_j \equiv 1 \bmod p$ 라면, $v_i + v_j \equiv 0 \bmod (p-1)$ 이고, 따라서 $x = -(r_i + r_j)(c_i + c_j)^{-1} \bmod (p-1)$ 이다.
4. 만약 $t_i \equiv t_j \bmod p$ 라면, $v_i \equiv v_j \bmod (p-1)$ 이고, 따라서 $x = (r_i - r_j)(c_j - c_i)^{-1} \bmod (p-1)$ 이다.

프로토콜 1에서의 정의에 의해 $t_i \equiv g^{v_i} \bmod p$ 이므로, 알고리즘 2는 결국 이전 실행에서 증명자가 선택했던 값 v_i와 일치하는 (혹은 그것의 additive inverse와 일치하는 v_j) 값이 다음 실행에서 선택될 때 까지 재실행이 요청되는 것이 핵심이다. 다시 말해, 과정 2의 조건이 만족되면 검증자는 비공개 정보를 탈취할 수 있다. 검증자가 비공개 정보를 탈취 할 빈도를 Monte carlo test를 통해 아래 그림 **3**과 같이 실험적으로 얻었으며, 그림 **4**는 이론적으로 계산된 확률이다.

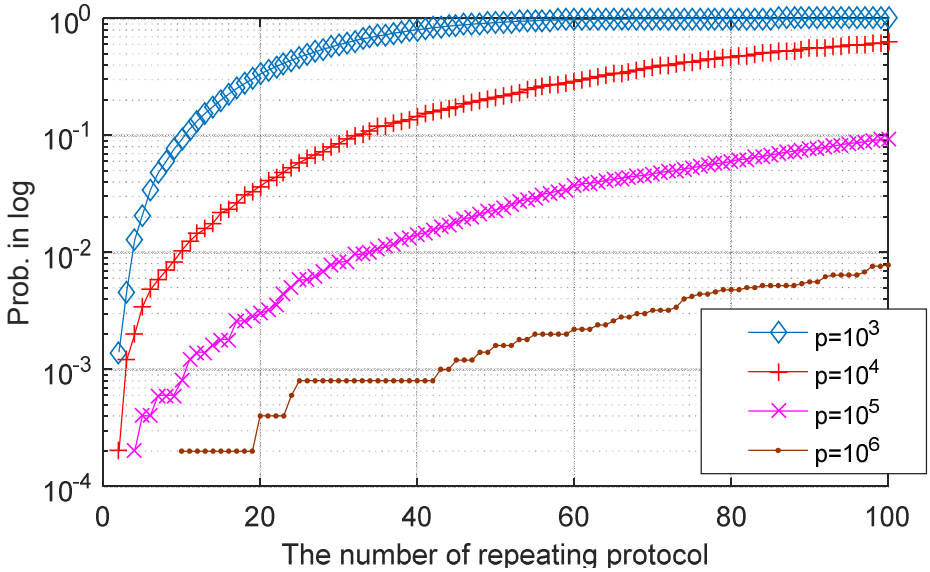


그림 3 실험적으로 얻은 검증자가 비공개 정보 탈취에 성공 할 실험적 확률. 가로축은 증명자에게 프로토콜 재실행을 요청하는 횟수이며, 세로축은 비공개 정보를 탈취 할 확률이다.

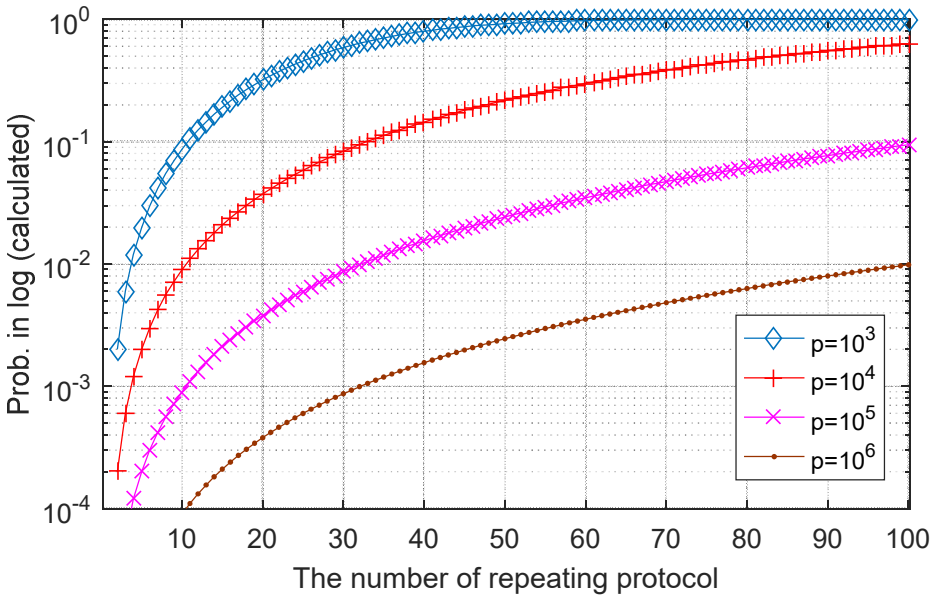


그림 4 계산으로 얻은 검증자가 비공개 정보 탈취에 성공 할 이론적 확률. 가로축은 증명자에게 프로토콜 재실행을 요청하는 횟수이며, 세로축은 비공개 정보를 탈취 할 확률이다.

그림 **4**의 정보 탈취 확률이 무시할 수 없는 수준이라는 것을 더 직관적으로 확인하기 위해, $y = g^x \bmod p$ 의 discrete logarithm 문제를 직접 풀어 정보를 탈취 할 확률과 비교 할 수 있다. $y \equiv g^x \bmod p$ 의 해인 x 를 찾을 확률은 아무 숫자 하나를 골라서 당첨 될 확률로 $(p-1)^{-1}$ 이다. 이보다 1000배 더 큰 $1000(p-1)^{-1}$ 의 확률로 정보를 탈취하기위해 필요로 하는 최소한의 프로토콜 재실행 횟수는 표 2와 같다. 표 2에 의하면, p 의 값이 아무리 크더라도, 프로토콜을 약 33회 정

도만 반복실행 하는 것을 통해 정보를 탈취 할 확률이 아무 숫자 하나를 골라서 당첨 될 확률보다 1000배 더 크다는 것을 알 수 있다.

p	10^3	10^4	10^5	10^6	…	10^{10}
정보 탈취확률	1	10^{-1}	10^{-2}	10^{-3}	…	10^{-7}
최소 프로토콜 재실행 횟수	182	33	33	33	…	33

표 2 검증자가 $1000(p-1)^{-1}$의 확률로 정보를 탈취하기위해 필요로 하는 최소한의 프로토콜 재실행 횟수. p의 크기가 1000근처의 수준인 경우, 검증자는 프로토콜을 약 182번 재실행하면 100%의 확률로 비공개 정보를 탈취 할 수 있다.

프로토콜 1이 강한 영지식성을 가지는 지를 명확히 결론내긴 어렵다. 프로토콜을 활용하는 환경에 따라 정보 탈취확률의 중요도가 다르기 때문이다. 예를 들어, p의 크기가 매우 크고 검증자가 증명자에게 프로토콜의 재실행을 요청할 수 있는 횟수가 제한된 환경이라면, 정보탈취확률은 무시할 수 있을 정도로 작을 것이다. 반면, 네트워크 환경이 불안정하여 프로토콜의 재실행 요청이 불가피한 환경이라면, 악의적 검증자가 정보탈취를 시도 할 가능성을 배제 할 수도 없다.

참고) 검증자가 n 번째 재실행 요청 내에 비공개 정보 탈취에 성공 할 확률 P_n:

짝수 p에 대하여,

$$P_n = 1 - \binom{\frac{p-2}{2}}{n} \frac{n!}{\left((p-2)/2\right)^n} \text{ for } 0 \le n \le \frac{p-2}{2}.$$

홀수 p에 대하여,

$$P_n = 1 - \left(\frac{p-3}{p-2}\right)^n \binom{\frac{p-3}{2}}{n} \frac{n!}{\left((p-3)/2\right)^n} - \frac{n(p-3)^{n-1}}{(p-2)^n} \binom{\frac{p-3}{2}}{n-1} \frac{(n-1)!}{\left((p-3)/2\right)^{n-1}} \text{ for } 0 \le n \le \frac{p-3}{2}.$$

정리 1. 대화형 구조와 도전적 문제(challenge)의 필요성: 건실성 확보

프로토콜 1이 강한 건실성을 만족하는 이유는 도전적 문제에 해당하는 과정 2)의 존재 때문이다. 만약 과정 2)에서 검증자가 아닌 증명자가 임의로 값 c를 선택한다면, 거짓 증명자가 검증자를 기만 할 수 있다. 구체적으로, 거짓 증명자가 임의로 값 c와 r을 미리 선택한 후, $t = g^r y^c \bmod p$를 계산하여 과정 1)을 대체한다. 이후 과정 2)와 3)에서 미리 선택해둔 c와 r을 검증자에게 건넨다. 마지막으로 검증자는 과정 4)를 수행하면 검증이 완료된다. 결과적으로, 거짓 증명자는 x를 알지 못하여도 검증자를 기만 할 수 있게 된다.

과정 2)의 도전적 문제를 검증자가 직접 제출하더라도, 만약 과정 1)과 순서가 바뀌면 위와 같은 이유로 거짓 증명자가 검증자를 기만 할 수 있다. 따라서 프로토콜 1과 같이 과정 1)을 통해 증명자가 먼저 값 t를 제출 하게 한 후, 검증자는 이를 확인하고 도전적 문제 c를 제출하는 방식의 대화형 구조가 반드시 필요하다.

정리 2. 과정 1) (임의의 값 v설정)의 필요성: 영지식성 확보

프로토콜 1이 약한 영지식성을 만족하는 이유는 프로토콜의 과정 1의 덕분이다. 검증자는 x 뿐만 아니라 과정 1에서 생성된 v의 값 또한 알지 못한다. 위의 약한 영지식성 문제에서 살펴보았듯이, 검증자는 $f^{-1}(r)$로부터 x를 알아내기 위해 v까지 함께 알아내야 한다. 그러나 함수 f는 injective이기 때문에 $f^{-1}(r)$에 대응되는 쌍 (v,x)가 유일하지 않다. 만약 과정 1에서 v가 고정된 값이라면, 함수 f의 domain의 원소의 개수가 $(p-1)$이 되어 c값의 선택에 따라 (c와 $p-1$이 서로소인 경우) f가 one-to-one correspondence (bijective)가 될 수 있다. 다시 말해, v가 고정된 값이라면 $f^{-1}(r)$의 유일한 값 x가 존재 할 수 있고, 검증자가 이를 찾을 수도 있다.

NIZK의 예: Schnorr protocol with Fiat-Shamir heuristic

앞선 Schnorr 프로토콜의 예에서 대화형 구조와 도전적 문제 (challenge)의 중요성을 살펴보았다. 도전적 문제가 없으면 Schnorr 프로토콜이 건실성을 만족하지 못하였다. 그리고 도전적 문제는 증명자가 아닌 검증자가 대화의 형태로 제출하였다: 증명자가 먼저 무작위 값을 제출 한 후 검증자가 이어서 도전적 문제를 제출하는 것이었다. 이렇게하지 않으면 거짓 증명자가 도전적 문제를 자신에게 유리하게 조작할 수 있게 되어 거짓 증명이 쉬워지기 때문이었다. 즉, 도전적 문제의 공정성이 훼손되면 증명 프로토콜의 건실성이 위협받는다.

NIZK의 한 예인 Fiat-Shamir heuristic은 "random oracle"이라는 랜덤 프로그램의 존재를 가정한다. 증명자가 random oracle로부터 무작위 값을 수여받아 검증자의 도전적 문제를 대체한다. Random oracle은 매번 다른 무작위 값을 생성하여야 하며, 생성된 무작위 값이 random oracle에 의한 값이라는 것을 누구나 확인 할 수 있어야 한다. 다시 말해, 증명자가 도전적 문제를 정직한 과정을 거쳐 생성하였다는 것을 누구나 검증할 수 있어야 한다.

Random oracle로써 hash 함수가 사용 될 수도 있다[10]. Hash 함수는 입력값이 변함에 따라 출력값이 무작위처럼 변하는 특성이 있으며, 입력값을 알면 누구나 같은 출력값을 재생산 해볼 수 있다. 즉, 도전적 문제가 공정하게 생성된 것이 맞는지 누구나 검증 할 수 있다.

그러나 모든 NIZK가 random oracle의 존재를 가정하는 것은 아니다. 이후에 함께 살펴볼 최신 증명 프로토콜인 Groth16와 같은 프로토콜은 random oracle에 의존하지 않는 대신 암호화를 활용하여 도전적 문제가 공정함을 보장한다.

본론으로 돌아와, Fiat-Shamir heuristic에서는 도전적 문제를 검증자가 아닌 random oracle이 선택한다. Schnorr 프로토콜에 Fiat-Shamir heuristic을 적용하면 다음과 같다. 증명자는 공개 된 값 y, g, p에 대해 $y \equiv g^x \bmod p$를 만족하는 비공개 값 x를 알고 있음을 검증자에게 증명하는 상황이다.

프로토콜 3. Fiat-Shamir heuristic이 적용된 Schnorr 프로토콜

1. 증명자는 하나의 (비공개) 무작위 값 v $(0 \le v < p-1)$를 골라서 $t = g^v \bmod p$를 계산 하여 검증자에게 보낸다.
2. 증명자는 random oracle로부터 c $(0 \le c < p-1)$를 받아 검증자에게 보낸다.

[10] Wikipedia의 Fiat-Shamir heuristic 페이지에서는 random oracle로써 hash함수를 사용한 예를 보여준다. 그러나 이 경우 거짓 증명자가 도전적 문제를 자신에게 유리하게 조작 할 수 없는지에 관한 검증이 필요하다.

3. 증명자는 $r = v - cx \bmod (p-1)$를 계산하여 검증자에게 보낸다.
4. 검증자는 c가 random oracle에게서 받은 값임을 확인하고, $t \equiv g^r y^c \bmod p$ 임을 확인한다.

프로토콜 1과 Fiat-Shamir heuristic이 적용된 프로토콜 3을 비교해 보면, 증명자와 검증자 간의 대화(conversation)과정이 사라진 대신, 증명자가 검증자에게 최종 제출할 증거가 추가되었다. 구체적으로, 프로토콜 1에서는 r과 t 두 개의 값을 검증자에게 제출하였으나, 프로토콜 3에서는 r과 t, c 세 개의 값을 제출한다.

결론적으로, 대화과정을 제거한 대신 증거의 길이가 길어지는 trade-off현상이 발생하였다. 일반적으로 NIZK는 증명해야 할 명제가 복잡해질수록 증거의 길이가 더욱 길어질 수 있다. 증거의 길이가 길어지면 다양한 분야로의 응용이 제한이 될 수 있다. 이러한 이유로 최근의 영지식 증명 연구자들은 증거의 길이를 효율적인 수준으로 줄이는 zk-SNARK를 연구하고 있다.

zk-SNARK의 예: Groth16 프로토콜

Groth16이란 Jens Groth가 2016년 Eurocrypt 학술대회에서 발표한 논문[11]에 수록된 zk-SNARK 프로토콜이다. Groth16은 2013년 B. Parno에 의해 제안된 zk-SNARK 프로토콜인 "Pinocchio"의[12] 원리를 따르고 일부 성능을 개선한 프로토콜이다. 이 외에도 2020년 영국의 Aztec 그룹에서 발표된 "PlonK"도[13] 원리가 비슷한 부분이 많다. 이러한 zk-SNARK 연구자들의 주요 관심사는 증명과정이 더 빠르고, 증거의 크기가 더 작으며, 검증과정이 더 빠른 프로토콜을 설계하는 것이다.

Groth16과 같은 최근 연구된 zk-SNARK 프로토콜들은 그림 5와 같이 단순하게 묘사될 수 있다.

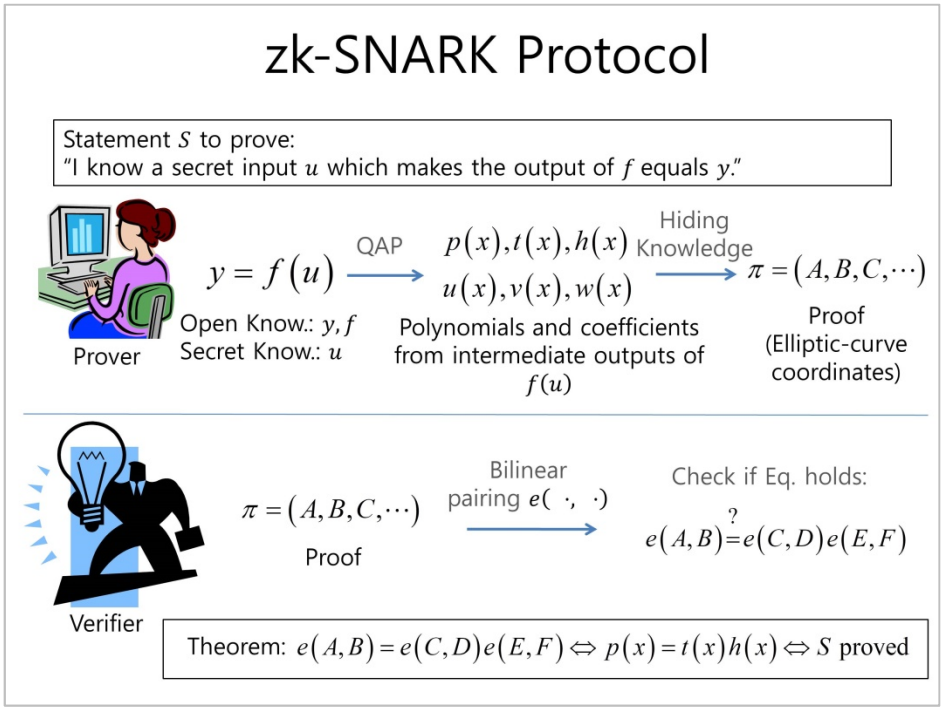


그림 5 최근 zk-SNARK 프로토콜들의 작동 원리

그림 5에서의 명제 S는 "증명자는 비공개 정보 u를 알며, 이를 사용하여 함수 $f(u)$ 충실히 계

---

[11] J. Groth, "On the size of pairing-based non-interactive arguments," EUROCRYPT 2016, pp. 305-326, 2016.

[12] B. Parno, et. al, "Pinocchio: nearly practical verifiable computation," IEEE Symposium on Security and Privacy 2013, pp. 238-252, 2013.

[13] A. Gabizon, et. al., "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge," Mathematics, Computer Science IACR Cryptol. ePrint Arch., 2019.

산 했고, 그 결과로 y를 얻었다"이다. 여기서 함수 f와 출력 값 y는 공개된 정보이다. 그림 5의 증명 프로토콜은 증명과정과 검증과정의 두 과정으로 구분된다. 증명과정은 $f(u)$의 연산과정을 분해하여 *연산조각*을 생성한 후, 연산조각을 암호화하여 최종 증거를 생성한다. 연산조각이란 두 개의 입력, 하나의 산술연산, 그리고 하나의 출력으로 구성된 최소 단위의 연산이다. 연산조각들이 서로 연결되면 $f(u)$의 연산과정 전체를 표현 할 수 있다. 즉, $f(u)$를 연산하는 과정에서 발생하는 모든 중간 계산값들이 연산조각들의 입력과 출력이 되는 것이다. 그러므로 오직 비공개 정보 u를 알고 있는 증명자만이 $f(u)$의 연산 과정을 분해 할 수 있다. 후에 검증자는 연산조각들을 건네 받아 연결 함으로써 $f(u)$의 연산이 올바르게 수행되었는지 확인이 가능하다. 즉, 이 과정은 증명의 완전성을 보장하는 역할을 수행한다. 이 과정을 quadratic arithmetic program(QAP)이라 한다.

그런데 이때 프로토콜의 영지식성과 건실성을 위해서 증명자는 연산조각들을 암호화 할 필요가 있다. 연산조각들을 연결하여 $f(u)$의 연산 과정이 알려지면 비공개 정보가 노출되기 때문이다. 그러므로 증명자는 연산조각을 암호화 하여 건네야 하며, 검증자는 암호화된 연산조각을 활용하여 $f(u)$의 연산 과정은 모르되 연산조각들에 오류가 없다는 것을 확인할 수 있어야 한다. 암호화에는 elliptic curve cryptography(ECC), 검증에는 pairing이라는 수학이 사용된다.

연산을 분해한다?

Groth16에서는 program analysis라는 방법과 QAP를 사용하여 연산 프로그램 $f(u)$의 전 계산 과정을 분해한다. Program analysis는 $f(u)$의 계산 과정을 사칙연산 게이트(gate)들과 이들을 잇는 선(wire)로 치환하는 과정이다. 게이트와 선으로 $f(u)$를 시각화한 도해를 "써킷(circuit)"이라 부른다 (그림 6). 그 후, QAP의 역할은 써킷을 구성하는 모든 게이트와 선들을 "다항식(polynomial)"과 "계수(coefficient)"의 형태로 변환하는 것이다 (그림 7). 각각의 다항식에는 써킷의 각각의 선들과 게이트들간의 연결 관계 (어떤 선이 어떤 게이트와 연결되어 있는지)에 관한 정보들이 저장되어 있다. 따라서 다항식의 총 개수는 선의 개수와 연관이 있다. QAP의 계수들은 선에 저장된 중간 계산 값들을 담고 있다.

$$c_6 = F(c_1, c_2, c_3, c_4)$$
$$= (c_1 + c_2)c_3c_4$$

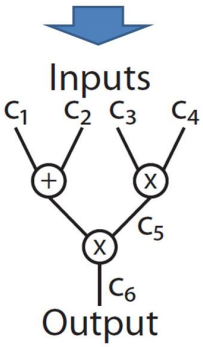
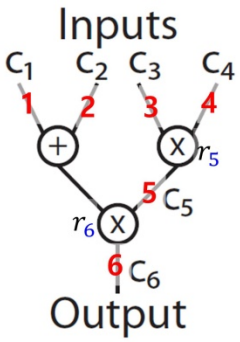


그림 6 Program analysis를 통해 4입력 함수 $F(x_1, x_2, x_3, x_4) = (x_1 + x_2)x_3x_4$ 를 써킷으로 변환한 모습 (써킷 변환 방법은 유일하지 않다). 여기서 4개의 입력 값 c_1, c_2, c_3, c_4와 중간 계산 값 c_5는 증명자만이 알고 있는 비공개 정보 일 수 있다. 출력 값 c_6은 일반적으로 공개된 정보이다.

Polynomial의 건설 예제

- 관련 Wire들에 번호를 붙임 (1,...,6)
- Multiply Gate들에 값을 할당함 (r_5, r_6)
- $t(x) := (x - r_5)(x - r_6)$ 으로 정의함



- $v_k(x)$의 표본을 건설함 (k는 Wire 번호)
  - $v_3(r_5) = 1$, $v_1(r_6) = 1$, $v_2(r_6) = 1$.
  - 그 외 모든 $v_k(r_5) = v_k(r_6) = 0$.
- $w_k(x)$의 표본을 건설함
  - $w_4(r_5) = 1$, $w_5(r_6) = 1$.
  - 그 외 모든 $w_k(r_5) = w_k(r_6) = 0$.
- $y_k(x)$의 표본을 건설함
  - $y_5(r_5) = 1$, $y_6(r_6) = 1$.
  - 그 외 모든 $y_k(r_5) = y_k(r_6) = 0$.
- 표본들을 기준으로 **Polynomial Interpolation**을 수행함 (Lagrange Interpolation 혹은 그 어떤 interpolation 방법이라도 적용 가능)

그림 7 써킷을 다항식과 계수로 변환하는 과정의 예.

Quadratic Arithmetic Program?

QAP는 써킷의 정보를 담고 있는 다항식들과 계수들의 집합이다. 그림 7은 써킷의 구조를 다항식들로 변환하는 과정의 예이다. 다항식 변환 과정은 다음과 같다:

1) 각 게이트에 임의의 값을 할당한다. 이 값들을 "근"이라 부른다 (그림에서 r_5와 r_6).

2) 모든 근을 zero로 갖는 다항식 $t(x)$를 정의한다.

3) 각각의 선에 대응되는 3개의 다항식 $v_k(x)$, $w_k(x)$, $y_k(x)$를 정의한다 (k는 선 번호).

4) 모든 k번 선에 대해, 어느 게이트의 왼쪽 입력이면 $v_k(x)$에서 해당 근을 대입 하였을 때의 출력을 1로 정의한다 (그림에서 $v_3(r_5) = 1$). 그렇지 않으면 출력을 0으로 정의한다 (그림에서 $v_4(r_5) = 0$). 마찬가지로 각 선이 어느 게이트 r의 오른쪽 입력이면, $w_k(r) = 1$로 정의한다, 그렇지 않으면 $w_k(r) = 0$으로 정의한다. 마찬가지로 각 선이 어느 게이트 r의 출력이면, $y_k(r) = 1$로 정의한다. 그렇지 않으면 $y_k(r) = 0$으로 정의한다.

5) 마지막으로, 과정 4)에서 정의한 $v_k(x)$, $w_k(x)$, $y_k(x)$의 표본(sample)들을 토대로 보간법(interpolation)을 사용하여, $v_k(x)$, $w_k(x)$, $y_k(x)$ 연속함수 형태의 다항식들로 만든다.

QAP의 다항식들에는 써킷의 연결구조가 저장되어 있다. QAP의 계수들(그림 6의 c_i)은 모든 선에 저장된 중간 계산 값이다. 다항식들은 함수 f를 알고 있는 누구나 생성 할 수 있다. 반면 계수들은 비공개 정보 u를 알고 있는 증명자만이 생성 할 수 있다. 다항식과 계수를 적절히 조합하면 써킷을 완벽히 복원 할 수 있다. 즉, 만약 증명자가 다항식과 계수를 검증자에게 제출한다면, 검증자는 이들을 조합하여 $f(u)$가 충실히 계산되었는지 확인 할 수 있다.

QAP가 증거가 될 수 있는 이유는?

다항식과 계수를 어떤 방식으로 조합하여야 써킷을 복원 할 수 있는지 구체적으로 파악 할 필요가 있다. 먼저 그림 7과 같은 QAP를 사용하여 다항식 $v_k(x)$, $w_k(x)$, $y_k(x)$, 그리고 $t(x)$를 완성하였다고 가정한다. 이때 사용된 근은 그림 7과 마찬가지로 r_5와 r_6이며, $m=6$개의 선이 있다고 가정 한다. 복원을 위해서 먼저 다음과 같은 "복원다항식" $p(x)$를 정의 한다:

$$p(x) := \sum_{k=1}^{m} c_k v_k(x) \sum_{k=1}^{m} c_k w_k(x) - \sum_{k=1}^{m} c_k y_k(x). \tag{1}$$

정의에 의해, 각 다항식 $v_k(x)$, $w_k(x)$, $y_k(x)$의 입력에 두 근 r_5와 r_6를 대입하면 그 출력은 0 혹은 1이다. 모든 다항식이 그림 7의 과정에 따라 정의되었다면, 식 (1)의 x에 두 근 r_5와 r_6를 대입하면 다음과 같은 두 개의 관계식이 출력된다.

$$p(r_5) = c_3 c_4 - c_5, \tag{2}$$

$$p(r_6) = (c_1 + c_2) c_5 - c_6. \tag{3}$$

만약 $p(r_5) = p(r_6) = 0$이라면 식 (2)와 식 (3)은 각각 써킷의 중간 계산들이 되며, 두 식을 연립하면 그림 6에 묘사된 $c_6 = F(c_1, c_2, c_3, c_4)$함수의 계산과정을 완전히 복원한다.

정리하면, 복원다항식이 써킷을 완전히 복원하기 위한 필요충분조건은 다음과 같은 관계를 만족하는 것이다:

$$p(r_5) = p(r_6) = 0. \tag{4}$$

$p(x)$라는 다항식을 잘 보관하고 있는 자라면 식 (4)의 조건이 만족하는지 확인하기 매우 쉽다. 단순히 r_5와 r_6를 입력으로 대입하여 출력이 0인지를 확인하면 되기 때문이다. 그러나 검증자는 이러한 방법을 사용할 수 없다. 검증자는 중간 계산결과들인 $c_1, c_2, \cdots, c_6$를 모르기 때문에 결국 $p(x)$를 알지 못한다. 따라서 검증자가 식 (4)의 조건이 만족하는지를 확인 할 다른 방법이 필요하다.

식 (4)의 조건을 확인하는 다른 방법은 "근 다항식 (target polynomial) $t(x)$를 활용하는 것이다.

$t(x)$는 모든 게이트의 근을 zero로 갖는 다항식으로써 정의된다. 그림 7의 QAP과정에서는 두 개의 게이트를 다루므로, $t(x) := (x-r_5)(x-r_6)$로 정의 되었다. 다시 말해 $t(x)$에 대해 다음과 같은 두 개의 등식이 만족된다.

$$t(r_5) = t(r_6) = 0. \tag{5}$$

검증자가 식 (4)의 조건이 만족하는지 확인하려면, $p(x) = t(x)h(x)$의 관계를 만족하는 어떤 $h(x)$가 존재하는지를 확인하면 된다. 여기서 $h(x)$는 pole이 없고 zero로만 구성된 다항식이다. 그 말은 즉, $p(x)$가 $t(x)$에 의해 나눠지는지(divisibility)를 확인하는 것이며, $h(x)$는 "몫 다항식"이다. 만약 그런 $h(x)$가 존재한다면, $p(x)$에 $x = r_5$와 $x = r_6$를 대입하였을 때 결과가 0이라는 사실이 보장된다. 이 뿐만 아니라 역으로, 만약 그런 $h(x)$가 존재하지 않는다면, $x = r_5$ 혹은 $x = r_6$을 $p(x)$에 대입하였을 때 적어도 하나의 결과는 0이 아닐 수 밖에 없다. 결과적으로, 증명자가 $p(x)$와 $h(x)$를 암호화 하여 검증자에게 건네주고, 검증자는 이후에 다룰 pairing을 사용하여 다음의 관계가 성립하는지 확인하여야 한다:

$$p(x) = t(x)h(x). \tag{6}$$

정리 3 (divisibility check). 복원다항식 $p(x)$가 써킷을 완벽히 복원할 수 있기 위한 필요충분조건은 $p(x)$를 $t(x)$로 나눴을 때 몫 다항식 $h(x)$가 존재하는 것이다.

QAP로 생성된 다항식 $t(x)$, $p(x)$, 그리고 $h(x)$의 정의와 역할을 표 3에 요약하였다:

다항식	설명
근 다항식 (target polynomial) $t(x) := (x-r_1)(x-r_2)\cdots(x-r_n)$	x에 각각의 근 $r_1, r_2, \cdots, r_n$을 대입하였을 때 결과가 0이 되도록 정의된 방정식
복원 다항식 $p(x) := \sum_{k=1}^{m} c_k v_k(x) \sum_{k=1}^{m} c_k w_k(x) - \sum_{k=1}^{m} c_k y_k(x)$	• QAP가 잘 수행되었는지 확인하는 방정식 • $p(x)$의 x에 각각의 근 $r_1, r_2, \cdots, r_n$을 대입하였을 때 모든 결과가 0이면 QAP가 잘 수행되었음 • QAP가 잘 수행되었음은 $p(x)$가 써킷을 완벽히 복원 할 수 있음을 의미함
몫 다항식 $h(x) = p(x)/t(x)$	• 만약 $p(x)$가 $t(x)$로 나누어 떨어지면, $h(x)$는 그 몫임 • 몫 다항식 $h(x)$가 존재하면 $p(x)$가 써킷을 완벽히 복원 할 수 있음

표 3 QAP로 생성되는 다항식들의 설명과 상호의존관계 요약

QAP 다항식들의 최대 차수

QAP 다항식들의 최대 차수를 정확히 파악하는 것은 이후에 소개될 Groth16 프로토콜을 구현하기 위해 반드시 필요하다. 최대 차수에 맞추어 다항식들을 암호화하는 과정이 수행되기 때문이다.

QAP 다항식들의 최대 차수는 써킷에 존재하는 게이트의 개수와 관련이 있다. 그 이유는 $i=1,\cdots,m$ 에 관한 다항식 $w_i(x), v_i(x), y_i(x)$ 를 생성하는 과정에서 사용된 보간법(interpolation)을 살펴보면 알 수 있다. 앞서 이들 각각의 다항식이 써킷의 골격을 대변함을 논의하였다. 구체적으로, 각각의 다항식은 각각의 게이트와의 연결관계에 따라 0 혹은 1의 표본을 반드시 포함하고 있었다. 따라서 각각의 다항식에 저장된 표본의 개수는 게이트의 개수와 일치한다. 앞서 우리는 각각의 다항식의 표본을 기저로 하여 보간법을 적용하였다. Lagrange 보간법에 의하면, 표본의 개수가 n 개 일때, 생성된 다항식의 최대 차수는 $n-1$ 이다. 따라서 게이트의 개수가 n 개이면 각각의 다항식 $w_i(x), v_i(x), y_i(x)$ 의 최대 차수는 $n-1$ 이다.

식 (1)에서 정의된 복원다항식 $p(x)$ 를 살펴보면 서로 같거나 다른 i 와 j 에 대해 두 다항식의 곱인 $w_i(x)v_j(x)$ 의 항들로 구성되어 있다. $w_i(x)$ 와 $v_j(x)$ 의 최대 차수가 $n-1$ 이므로 $p(x)$ 의 최대 차수는 $2(n-1)$ 이다. 여기서 n 은 게이트의 개수이다.

한편, 근 다항식인 $t(x)$ 는 표 3의 정의에 의해 최대 차수의 개수가 근의 개수와 일치하고, 근의 개수는 게이트의 개수와 일치한다. 따라서 $t(x)$ 의 최대 차수는 n 이다.

마지막으로, $h(x)=p(x)/t(x)$ 이므로, 최대 차수는 $n-2$ 이다.

요약하면, 표 4는 써킷의 선의 개수가 m 이고 게이트의 개수가 n 일 때, QAP 다항식들의 최대 차수를 정리 한 결과이다:

QAP 다항식	최대 차수
$w_i(x), v_i(x), y_i(x)$ for $i=1,\cdots,m$	$n-1$
$p(x)$	$2(n-1)$
$t(x)$	n
$h(x)$	$n-2$

표 4 써킷에 존재하는 선의 개수가 m 이고 게이트의 개수가 n 일 때 QAP 다항식들의 최대 차수

보안 목적의 ECC기반 암호화(cryptography)와 pairing

QAP가 사용 된 증명 프로토콜의 *완전성*은 *정리 3*에 의해 보장된다. 증명자가 써킷을 완벽히 복원 할 수 있음은 곧 증명자가 함수 $f(u)$ 를 정직하게 계산하였음을 의미하기 때문이다. 그러나

QAP가 증명 프로토콜의 *건실성*까지 보장하지는 못한다. 거짓 증명자가 QAP의 계수들을 모르더라도, *정리* ***3***의 조건을 만족시키는 다항식들을 특수한 조합을 사용하여 만들어 낼 수도 있기 때문이다. 이 뿐만 아니라 *영지식성* 또한 보장받지 못한다. $p(x)$가 노출되면 정직한 증명자의 QAP 계수가 검증자 혹은 제 3자에게 탈취당할 수 있다.

QAP를 사용하는 영지식증명 프로토콜의 *건실성*과 *기밀성* 보장을 위하여 ECC와 pairing을 적용할 수 있다. ECC는 비공개 정보를 암호화하여 숨긴다. ECC의 암호화는 복호가 매우 어렵다. Pairing은 비공개 값(정보)가 ECC에 의해 암호화된 상태에서도 더하기 및 곱하기 연산의 결과를 검증 가능하도록 해준다. 구체적으로 어떤 비공개 정보 x와 y가 있고 공개정보 $z = xy$일 때, ECC로 암호화한 결과를 각각 $ECC(x)$와 $ECC(y)$, $ECC(z)$라 하면, pairing의 역할은 $ECC(xy) = ECC(z)$가 맞는지의 확인을 $ECC(x)$와 $ECC(y)$의 복호 없이 수행할 수 있게 해주는 것이다[14]. 따라서 증명자는 $p(x)$와 $h(x)$를 암호화 한 후 증거로써 제출하고, 검증자는 pairing을 활용하여 *정리* ***3***의 조건 $p(x) = t(x)h(x)$이 성립하는지를 복호 없이 확인 할 수 있다.

Elliptic curve?

ECC는 타원곡선(elliptic curve)이라 불리는 곡선상의 점(point)들을 사용하여 정보를 감춘다. 타원곡선은 그 정의가 다양한데, 한가지의 예로 $S: y^2 = x^3 + ax + b$의 형태로 정의되는 곡선을 들 수 있다. 여기서 a와 b는 상수이며 정수이다. 그림 8은 다양한 a와 b에 따른 곡선 S를 그린 결과이다. 타원곡선 S상에서 정의된 점이란 (x, y)의 좌표 형태로 표현되며, x와 y를 타원곡선 방정식에 대입하였을 때 등호가 성립되는 점이다. ECC에서는 타원곡선상의 두 점간의 "점 덧셈" 연산을 정의하여 사용한다. 이 점 덧셈의 기호는 우리가 일반적으로 다루는 숫자덧셈의 기호와 "+"로 같지만, 그 정의는 완전히 다르다. 두 실수의 덧셈, 1+2=3 이라는 정의가 점 덧셈에서는 사용되지 않는다. 구체적으로, 타원곡선상의 두 점 A와 B의 덧셈 $A + B$는 다음과 같은 순서로 수행된다[15]:

1. A와 B가 서로 다를 경우, 두 점을 잇는 직선 l을 그린다.
2. A와 B가 서로 같을 경우, 그 점에서 타원곡선 S와 접하는 접선 l을 그린다.
3. 선 l과 타원곡선 S가 교차하는 새로운 점을 찾는다.

---

[14] Pairing에는 $ECC(x)$와 $ECC(y)$만을 사용하여 $ECC(x+y)$를 계산해주는 기능은 없다. 이러한 기능이 지원되는 암호화를 동형암호 (homomorphic encryption)라 부른다.

[15] 타원 곡선의 점 덧셈의 구체적인 정의는 Wikipedia의 elliptic curve 문서에 잘 설명되어 있다. 링크: https://en.wikipedia.org/wiki/Elliptic_curve

4. 만약 교차하는 새로운 점이 없다면, 덧셈의 결과는 항등원인 O 이다.

5. 만약 교차하는 새로운 점이 있다면, 그 점의 x 축 대칭점이 덧셈의 결과인 $A+B=C$ 이다.

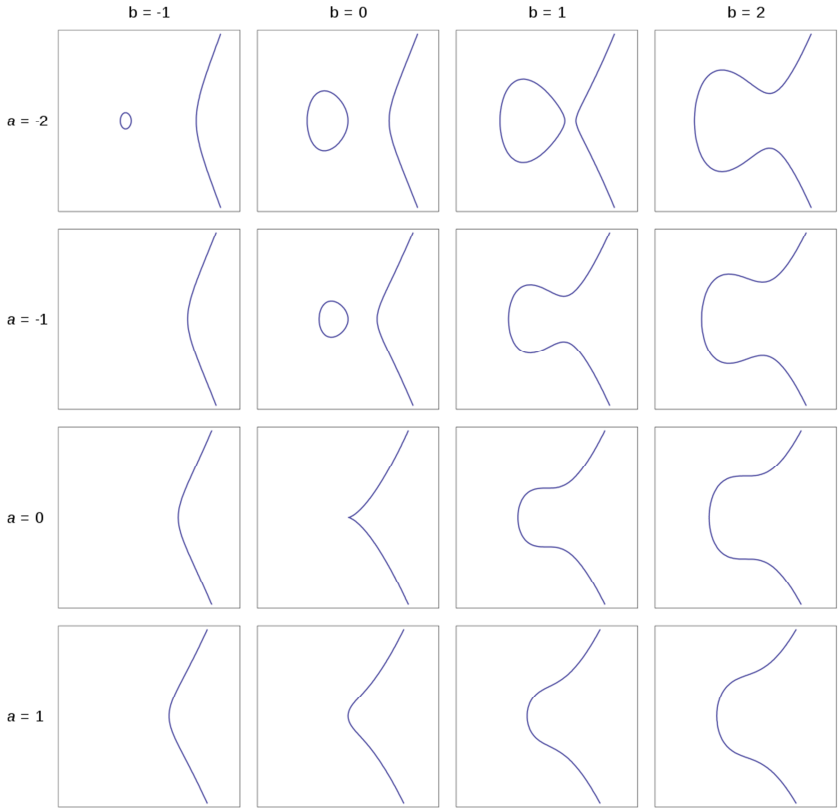


그림 8 타원곡선 $y^2=x^3+ax+b$ 의 a 와 b 의 값에 따른 plot. 각 plot의 가로축은 x 축이고 세로축은 y 축이며 각각의 범위는 -3부터 3까지이다. (출처: Wikipedia)

위의 좌표덧셈 연산에서 점 O 와 대칭점이 언급되었다. 점 O 는 point at infinity로 불리는 좌표이며, 다른 좌표들과는 달리 (x,y) 의 형태로 표현 될 수 없다[16]. 점 O 는 항등원의 역할을 한다. 다시 말해, 어떤 점 A 에 대해 $A+O=A$ 이며, $O+O=O$ 이다. 어떤 점 $A=(x,y)$ 의 대칭점은 $-A$ 이며, 그 좌표는 $(x,-y)$ 이다. 점 덧셈의 정의에 의해 $A+(-A)=O$ 이다.

Elliptic curve를 이용하는 cryptography?

---

[16] 이 문서에서 다루는 타원곡선은 affine 좌표계(coordinate system)라 불리는 (x,y) 좌표계에서 표현되고 있다. 그러나 타원곡선을 표현하는 다른 좌표계도 존재한다. 대표적으로 projective 좌표계가 있는데, 이는 타원곡선상의 점을 (x,y,z) 로 표현한다. Projective 좌표계에서는 $O=(0,1,0)$ 으로 표현 할 수 있다. 참고로 좌표계란 타원곡선을 표현하는 방법일 뿐, 좌표계를 변환한다고 해서 곡선의 모양이 변하지는 않는다.

타원곡선의 점 덧셈 연산은 역 추적이 어렵다. 예를 들어 어떤 점 A를 5번 더한 결과를 B, 즉 $B=5A$라 하면, B를 보고 이 점이 A를 몇 번 더한 결과인지 유추해내는 과정이 매우 어렵다는 의미이다. 반면 실수 덧셈의 경우, 숫자 3을 5번 더해 15를 만든 후, 그 결과 15를 보고 3을 몇 번 더하였는지 쉽게 알 수 있다. EC의 이러한 역 추적 문제를 "discrete logarithm" 문제라 부른다[17]. Discrete logarithm 문제를 활용한 암호화가 ECC이다.

암호화를 수행하기 전에, 먼저 사용할 타원곡선 S와 그 위의 한 점 G를 약속 해 둔다. 이 때 G는 "generator"라 불리는 특수한 점이다[18]. G를 사용하여 어떤 정수 값 x를 암호화 하는 연산을 $[x]_G$라 표현하겠다. 암호화 결과를 X라 할때, 암호화 결과는 다음과 같다:

$$X=[x]_G := xG. \tag{7}$$

암호화 결과 X는 좌표로 표현되는 타원곡선상의 한 점이며, 점 G에 x회의 점 덧셈을 수행 한 결과이다. G를 알때, X로부터 x를 유추하는 것은 매우 어렵다.

정의 (7)에 의해, 두 정수 a와 b에 대해 암호화 연산 $[\cdot]_G$는 linear 연산이다. 즉, 다음과 같은 additivity와 scalability (homogeneity)를 갖는다:

$$[a+b]_G=[a]_G+[b]_G, \tag{8}$$

$$a[b]_G=[ab]_G. \tag{9}$$

Bilinear pairing

Pairing $e(\cdot,\cdot)$는 입력이 2개의 타원곡선 점들이고 출력이 숫자 값인 함수이다[19]. Pairing이 지닌 특성 중 가장 중요한 특성은 "Bilinearity"이다. Bilinear pairing의 대표적인 예로 Weil pairing, Tate pairing, Ate pairing이 있으며, 최근 동향에서는 Ate pairing 중 성능이 최적화된 optimal Ate pairing이 가장 많이 사용된다[20].

---

[17] Discrete logarithm problem이란 $y=g^x \bmod p$에서 y와 g, p를 알 때 x를 찾는 문제이다. 이러한 문제는 수학적으로 어려운 문제로 잘 알려져 있다. ECC에서도 같은 용어가 사용되어, elliptic curve discrete logarithm problem (ECDLP)로 불리고 있다.

[18] Generator에 점 덧셈 연산을 반복적으로 수행하면 타원곡선상의 모든 점들이 생성된다.

[19] Pairing을 두 점의 내적(inner product)으로 표현하기도 한다.

[20] Weil pairing의 mapping rule은 D. Boneh와 M. Franklin의 2003년 논문인 "Identity-based encryption from the weil pairing"의 Appendix에 간결하게 설명되어 있다. 이 외의 다른 Pairing들

세 점 X, Y, Z와 정수 c와 d에 대하여 bilinearity를 정리하면 다음과 같다:

$$\begin{aligned} &e(X+Y,Z)=e(X,Z)e(Y,Z),\\ &e(X,Y+Z)=e(X,Y)e(X,Z),\\ &e(cX,dZ)=e(X,Z)^{cd}. \end{aligned} \tag{10}$$

이러한 Bilinear 특성은 ECC로 암호화된 정보의 더하기와 곱셈 연산을 복호 없이 수행하는데 사용될 수 있다. 예를 들어, 어떤 점 G와 H가 있고, 이들을 사용하여 비공개 정보 (정수 값) x와 y, z, w를 암호화 하였다고 가정하자. 비공개 정보들은 $xy+z=w$라는 관계를 만족한다는 것이 알려져 있다고 가정하자. 암호화된 결과를 사용하여 비공개 정보들이 위의 관계를 만족하는지 확인하는 방법은 다음의 등호가 성립하는지를 확인하는 것이다:

$$e(xG,yH)e(zG,H)=e(wG,H). \tag{11}$$

식 (10)의 bilinearity에 의해, 식 (11)은 다음과 같이 정리 될 수 있다:

$$\begin{aligned} &e(xG,yH)e(zG,H)=e(wG,H)\\ &\Leftrightarrow e(G,H)^{xy}\,e(G,H)^{z}=e(G,H)^{w}\\ &\Leftrightarrow e(G,H)^{xy+z}=e(G,H)^{w}\\ &\Leftrightarrow e((xy+z)G,H)=e(wG,H). \end{aligned} \tag{12}$$

결론적으로, 암호화 결과들인 xG, yH, zG, wG에 pairing을 적용함으로써 비공개 정보들의 복호 없이 $xy+z=w$라는 등식이 만족하는지 확인 할 수 있다.

편의를 위해, 두 정수 x,y와 두 점 G,H에 대해, xG와 yH의 pairing을 다음과 같이 표기하겠다:

$$e(xG,yH)=:[x]_G\cdot[y]_H. \tag{13}$$

그리고 다음과 같은 표기를 사용하겠다:

$$[xy]_{G\cdot H}:=[x]_G\cdot[y]_H=e(xG,yH). \tag{14}$$

새롭게 정의된 연산 $[\cdot]_{G\cdot H}$의 bilinearity를 다음과 같이 표기한다:

$$\begin{aligned} e(xG,H)e(yG,H)&=[x]_{G\cdot H}+[y]_{G\cdot H}\\ &=[x+y]_{G\cdot H}\\ &=e((x+y)G,H). \end{aligned} \tag{15}$$

도 Weil pairing과 최종 연산 방식만 다를 뿐 원리는 모두 같다. 그러므로 서로 변환 가능하다.

이러한 표기의 장점은 식 (12)과 같은 표현을 선형대수적 표현으로 변환하여 직관적으로 읽기 쉽게 해주기 때문이다. 예를 들어 식 (12)은 다음과 같이 표기될 수 있다:

$$\begin{aligned} & e(xG, yH)e(zG, H) = e(wG, H) \\ \Leftrightarrow & [xy+z]_{G\cdot H} = [w]_{G\cdot H}. \end{aligned} \tag{16}$$

식 (12)의 목적이 비공개 정보들이 $xy+z=w$ 의 관계를 만족하는지 확인하는 것임을 고려할 때, 식 (16)의 두 번째 등식이 그 목적을 더욱 직관적으로 알아볼 수 있게 해준다.

Groth16 프로토콜: QAP와 ECC를 활용한 zk-SNARK 프로토콜

하나의 증명상황을 가정하겠다. 증명하고자 하는 명제 S 는 "증명자는 비공개 정보 u_1, u_2 를 알며, 이를 사용하여 함수 $f(u_1, u_2)$ 충실히 계산 했고, 그 결과로 출력 y_1, y_2 를 얻었다"이다. 여기서 함수 f 와 출력 값 y_1, y_2 는 공개된 정보이다. 이 예시에서는 비공개 정보를 2개, 공개 정보를 2개로 설정하였지만, Groth16프로토콜에서는 일반적으로 비공개정보의 개수와 공개 정보의 개수를 제한하지 않는다.

증명자는 QAP를 사용하여 $f(u_1, u_2)$ 를 분해하였다고 가정하겠다. QAP의 결과로 생성된 다항식들은 $v_i(x)$, $w_i(x)$, $y_i(x)$, $t(x)$ 이며 계수는 c_i 이다. 여기서 $i=1,\cdots,m$ 이고, $c_1=u_1$, $c_2=u_2$, $c_{m-1}=y_1$, $c_m=y_2$ 이다. 즉, $c_1, c_2, \cdots, c_{m-2}$ 는 모두 비공개 정보이며, c_{m-1} 과 c_m 은 공개된 정보이다. 증명자는 식 (1)을 사용하여 복원다항식 $p(x)$ 를 계산하였고, *정리* ***3***의 조건을 만족시키는 $h(x)$ 를 계산하였다고 가정하겠다.

덧붙여 함수 f 는 공개된 정보이기 때문에 검증자를 포함한 누구나 $i=1,\cdots,m$ 에 대한 QAP 다항식들 $v_i(x)$, $w_i(x)$, $y_i(x)$, $t(x)$ 를 얻을 수 있다. 그러나 증명자 이외의 그 누구도 비공개 정보 u_1, u_2 를 알지 못하는 한 QAP 계수들 $c_1, c_2, \cdots, c_{m-2}$ 를 알 수는 없고, 그러므로 $h(x)$ 를 알지 못한다.

QAP 결과로 생성된 다항식과 계수들을 공개정보 혹은 비공개정보의 기준으로 분류하면 표 5와 같다.

공개 정보	QAP 계수들 중 c_{m-1}, c_m
	QAP 다항식 $v_i(x), w_i(x), y_i(x)$ for $i=1,\cdots,m$
	QAP 근 다항식 $t(x)$
비공개 정보	QAP 계수들 중 $c_1, \cdots, c_{m-2}$
	QAP 복원다항식 $p(x)$
	QAP 몫 다항식 $h(x)$

표 5 공개/비공개 특성에 따른 QAP 결과의 분류

Groth16 프로토콜의 증명-검증 과정이 프로토콜 4에 소개되어 있다. 프로토콜에서 사용되는 연산들에서 정수간의 사칙연산과 타원곡선 점 덧셈연산, 그리고 타원곡선 점간의 pairing을 잘 구분하여 읽을 필요가 있다. 예를 들어, 같은 $+$ 기호더라도 앞 뒤의 피연산자가 정수이면 정수덧셈이며, 피연산자가 타원곡선의 점이면 점 덧셈연산이고, 피연산자가 pairing 결과이면 정수 곱셈임을 인지하여야 한다.

프로토콜 4. Groth16 프로토콜

Setup: 제 3자가 실행한다. Common reference string (CRS)를 생성한다.

1. 무작위 비공개 정수 리스트 $\tau := (\alpha, \beta, \gamma, \delta, x)$ 를 생성한다 (τ 는 증명자에게 알려져서는 안된다).
2. 타원곡선상의 두 점(generator) G 와 H 를 선택하여 공개한다.
3. 첫 번째 CRS인 σ_G 를 생성한다:

$$\sigma_G := \left(\begin{array}{l} [\alpha]_G, [\beta]_G, [\delta]_G, \left\{ \left[x^i \right]_G \right\}_{i=0}^{n-1}, \\ \left\{ \left[\frac{\beta v_i(x) + \alpha w_i(x) + y_i(x)}{\gamma} \right]_G \right\}_{i=m-1}^{m}, \\ \left\{ \left[\frac{\beta v_i(x) + \alpha w_i(x) + y_i(x)}{\delta} \right]_G \right\}_{i=1}^{m-2}, \\ \left\{ \left[\frac{x^i t(x)}{\delta} \right]_G \right\}_{i=0}^{n-2} \end{array} \right).$$

4. 두 번째 CRS인 σ_H 를 생성한다:

$$\sigma_H := \left([\beta]_H, [\gamma]_H, [\delta]_H, \left\{ \left[x^i \right]_H \right\}_{i=0}^{n-1} \right).$$

5. σ_G 와 σ_H 를 공개한다.

Prove: 증명자가 수행한다. 증명자만이 아는 비공개 정보와 CRS σ_G, σ_H 를 조합하여 증거를 생성한다.

1. 무작위 비공개 정수 r, s 를 생성한다 (r, s 는 검증자에게 알려져서는 안된다).
2. CRS σ_G 를 사용하여 첫 번째 증거 $[A]_G$ 를 생성한다:

$$[A]_G := [\alpha]_G + \sum_{i=1}^{m} c_i \sum_{k=0}^{n-1} a_{i,k} \left[x^k \right]_G + r[\delta]_G,$$

여기서 $v_i(x) = \sum_{k=0}^{n-1} a_{i,k} x^k$.

3. CRS σ_H 를 사용하여 두 번째 증거 $[B]_H$ 를 생성한다:

$$[B]_H := [\beta]_H + \sum_{i=1}^{m} c_i \sum_{k=0}^{n-1} b_{i,k} \left[x^k \right]_H + s[\delta]_H,$$

여기서 $w_i(x) = \sum_{k=0}^{n-1} b_{i,k} x^k$.

4. CRS σ_G 를 사용하여 세 번째 증거 $[C]_G$ 를 생성한다:

$$[C]_G := \sum_{i=1}^{i=m-2} c_i \left[\frac{\beta v_i(x) + \alpha w_i(x) + y_i(x)}{\delta} \right]_G + \sum_{k=0}^{n-2} d_k \left[\frac{x^k t(x)}{\delta} \right]_G + s[A]_G + r[B]_G - rs[\delta]_G,$$

여기서 $h(x) = \sum_{k=0}^{n-2} d_k x^k$ 이고, $[B]_G := [\beta]_G + \sum_{i=1}^{m} c_i \sum_{k=0}^{n-1} b_{i,k} \left[x^k \right]_G + s[\delta]_G$.

5. 세 개의 증거 리스트 $\pi := \left([A]_G, [B]_H, [C]_G\right)$를 공개한다.

Verify: 검증자가 수행한다. 검증자가 아는 공개정보와 증거 π, 그리고 CRS σ_G, σ_H를 조합하여 증거를 생성한다.

1. 증거 π의 데이터들이 모두 타원곡선상의 점이 맞는지 확인한다.
2. 증거 π를 사용하여 $LHS := [A]_G \cdot [B]_H$를 계산한다.
3. 증거 π와 CRS를 사용하여 다음을 계산한다:

$$RHS := [\alpha]_G \cdot [\beta]_H + \left(\sum_{i=m-1}^{m} c_i \left[\frac{\beta v_i(x) + \alpha w_i(x) + y_i(x)}{\gamma} \right]_G \right) \cdot [\gamma]_H + [C]_G \cdot [\delta]_H .$$

4. $LHS = RHS$를 확인하여, 만약 등호가 성립하면 증거를 인정하고, 그렇지 않으면 증거를 부정한다.

Groth16의 영지식 증명 성능 분석

Q. (완전성) 만약 증명자가 비공개정보를 알고있다면, 검증자는 $LHS = RHS$를 확인함으로써 이를 납득할 수 있는가?

A. 그렇다.

연산 $[\cdot]_G$와 $[\cdot]_{G \cdot H}$의 정의를 사용하여 LHS와 RHS를 각각 정리하면 *정리* ***3***의 조건 인 $p(x) = t(x)h(x)$를 유도 할 수 있다. *정리* ***3***에 의해 증명자는 써킷을 완벽히 복원 할 수 있고, 따라서 모든 비공개 정보를 다 알고 있다.

먼저 $[A]_G$, $[B]_H$를 정리하면 각각 다음과 같다:

$$\begin{aligned} [A]_G &= \left[\alpha + \sum_{i=1}^{m} c_i v_i(x) + r\delta \right]_G , \\ [B]_H &= \left[\beta + \sum_{i=1}^{m} c_i w_i(x) + s\delta \right]_H . \end{aligned} \tag{17}$$

그리고, $[C]_G$를 정리하면 다음과 같다:

$$[C]_G = \left[\frac{\sum_{i=1}^{i=m-2} c_i \left(\beta v_i(x) + \alpha w_i(x) + y_i(x) \right) + h(x)t(x)}{\delta} + sA + rB - rs\delta \right]_G . \tag{18}$$

먼저 LHS를 정리하면,

$$\begin{aligned} LHS = [A]_G \cdot [B]_H &= \left[\alpha + \sum_{i=1}^{m} c_i v_i(x) + r\delta \right]_G \cdot \left[\beta + \sum_{i=1}^{m} c_i w_i(x) + s\delta \right]_H \\ &= \left[\alpha\beta + \sum_{i=1}^{m} c_i v_i(x) \cdot \sum_{i=1}^{m} c_i w_i(x) + (\alpha + r\delta) \sum_{i=1}^{m} c_i w_i(x) + (\beta + s\delta) \sum_{i=1}^{m} c_i v_i(x) + \alpha s\delta + \beta r\delta + rs\delta^2 \right]_{G \cdot H} . \end{aligned}$$

그리고 RHS 를 정리하면,

$$\begin{aligned} RHS &= [\alpha]_G \cdot [\beta]_H + \left(\sum_{i=m-1}^{m} c_i \left[\frac{\beta v_i(x) + \alpha w_i(x) + y_i(x)}{\gamma} \right]_G \right) \cdot [\gamma]_H + [C]_G \cdot [\delta]_H \\ &= \left[\alpha\beta + \sum_{i=1}^{m} c_i \left(\beta v_i(x) + \alpha w_i(x) + y_i(x) \right) + h(x)t(x) + s\delta A + r\delta B - rs\delta^2 \right]_{G \cdot H}. \end{aligned} \tag{20}$$

LHS 와 RHS 에 공통으로 존재하는 항들을 제거하면 다음과 같다:

$$\begin{aligned} \frac{LHS}{RHS} &= \left[\sum_{i=1}^{m} c_i v_i(x) \cdot \sum_{i=1}^{m} c_i w_i(x) - \sum_{i=1}^{m} c_i y_i(x) - h(x)t(x) \right]_{G \cdot H} \\ &+ \left[s\delta \sum_{i=1}^{m} c_i v_i(x) + r\delta \sum_{i=1}^{m} c_i w_i(x) \right]_{G \cdot H} \\ &+ \left[\alpha s\delta + \beta r\delta + 2rs\delta^2 - s\delta A - r\delta B \right]_{G \cdot H}. \end{aligned} \tag{21}$$

식 (21)에 (17)의 A 와 B 를 대입하면,

$$\frac{LHS}{RHS} = \left[\sum_{i=1}^{m} c_i v_i(x) \cdot \sum_{i=1}^{m} c_i w_i(x) - \sum_{i=1}^{m} c_i y_i(x) - h(x)t(x) \right]_{G \cdot H}. \tag{22}$$

식 (1)의 복원방정식 $p(x)$ 의 정의를 대입하면 식 (22)은 다음과 같다:

$$\frac{LHS}{RHS} = \left[p(x) - h(x)t(x) \right]_{G \cdot H}. \tag{23}$$

결과적으로, $LHS = RHS \Leftrightarrow p(x) = h(x)t(x)$ 이고, 정리 1에 의해 관계식 $LHS = RHS$ 와 명제 "증명자가 써킷을 완벽히 복원 할 수 있다"는 것이 등가이다.

Q. (건실성) 만약 거짓 증명자가 비공개 정보를 모른다면, 제출한 거짓 증거가 $LHS = RHS$ 를 만족시킬 수 있는가?

A. CRS의 내용이 암호화 되어있지 않으면 거짓 증명자가 검증자를 쉽게 기만 할 수 있다. 그러나 Groth16 프로토콜에서는 *CRS가 암호화 되어 있기 때문에 그렇게 하기가 어렵다.*

만약 Setup 과정에서 비공개 정수 리스트 $\tau = (\alpha, \beta, \gamma, \delta, x)$ 의 값들이 공개된다면 거짓 증명자가 검증자를 속이는 것은 매우 쉽다. x 가 더 이상 변수가 아닌 정수 상수이기 때문에 $v_i(x)$, $w_i(x)$, $y_i(x)$, $t(x)$, 그리고 $h(x)$ 도 더 이상 다항식이 아닌 어떤 정수이다. 다음과 같이 정의하자: $V := \sum_{i=1}^{m-2} c_i v_i(x)$, $W := \sum_{i=1}^{m-2} c_i w_i(x)$, $Y := \sum_{i=1}^{m-2} c_i y_i(x)$, $H := h(x)$. 이제 거짓 증명자가 할 일은 식 (22)의 계산 결과가 1이 되도록 하여 정리 1의 조건을 만족시키는 적절한 정수 V, W, Y, H 를 임의로 선택하는 것이다. 예를 들어, V, W, Y 를 무작위로 선택 한 후, 다음의 조건을 만족하는 정수 H 를 찾는다:

$$\left(V+\sum_{i=m-1}^{m} c_i v_i(x)\right)\left(W+\sum_{i=m-1}^{m} c_i w_i(x)\right)-\left(Y+\sum_{i=m-1}^{m} c_i y_i(x)\right)=t(x)H. \tag{24}$$

마지막으로 거짓 증명자가 최종적으로 제출 할 거짓 증거는 다음과 같다:

$$\pi'=\begin{pmatrix} [A']_G \coloneqq [\alpha]_G+[V]_G+\left[\sum_{i=m-1}^{m} c_i v_i(x)\right]_G+r[\delta]_G, \\ [B']_H \coloneqq [\beta]_H+[W]_H+\left[\sum_{i=m-1}^{m} c_i w_i(x)\right]_H+s[\delta]_H, \\ [C']_G \coloneqq \left[\frac{(\beta V+\alpha W+Y)}{\delta}\right]_G+\left[\frac{t(x)H}{\delta}\right]_G+s[A']_G+r[B']_G-rs[\delta]_G \end{pmatrix}. \tag{25}$$

거짓 증거를 사용하여 Verify과정의 LHS 와 RHS 를 계산 해 보면 $LHS=RHS$ 인 것을 알 수 있다.

Groth16 프로토콜에서는 Setup 과정에서 생성되는 비공개 정수 리스트 $\tau=(\alpha,\beta,\gamma,\delta,x)$ 가 암호화 후 공개된다. 그렇기 때문에 증명자는 τ 의 값들을 알지 못한다. 구체적으로, x 를 모르기 때문에 (24)과 같이 정리 1의 조건을 만족시키는 값 V,W,Y,H 도 찾지 못한다. 이 뿐만 아니라, α,β,δ 를 모르기 때문에 (25)와 같은 거짓 증거를 만들어 낼 수도 없다.

Q. (영지식성) 검증자는 증거 π 로부터 비공개 정보 $c_1,\cdots,c_{m-2}$ 를 알아 낼 수 있는가?

A. ECC에 의해 암호화된 증거를 복호하여 *비공개 정보를 추출하는 것은 매우 어렵다.* 비록 비공개 정보를 추출하지 못하더라도, 악의적인 검증자가 증거를 재사용 하려는 경우가 있을 수 있다. 이러한 *재사용 시도는 증명자가 임의로 생성하는 비공개 값 r,s 에 의해 차단 된다.*

만약 r,s 가 존재하지 않는다면, 혹은 r,s 의 값이 노출된다면, 악의적인 검증자가 증거를 조작하여 재사용 할 수 있다. 여기서 악의적인 검증자가 Setup 과정에서 생성된 비공개 정수 리스트 $\tau=(\alpha,\beta,\gamma,\delta,x)$ 를 알고 있다고 가정하겠다. 동일한 증명자의 비공개 정보는 일정하다고 가정하겠다. 증명자는 프로토콜을 N 회 사용하였으며, 따라서 검증자는 N 개의 증거를 보유하고 있다. 여기서 $N>m-2$ 이다. $k=1,\cdots,N$ 번째 제출된 증거를 $\pi_k=([A_k]_G,[B_k]_H,[C_k]_G)$ 라 하겠다. 증명자가 k 번째 증거를 생성할 때 사용 한 파라미터를 $\tau_k=(\alpha_k,\beta_k,\gamma_k,\delta_k,x_k), r_k, s_k$ 라 하겠다. 다음의 알고리즘은 τ_k,r_k,s_k 의 값이 검증자에게 알려져 있을 때, 검증자가 증거를 재사용 할 수 있도록 하는 알고리즘이다:

알고리즘 5. 보안 파라미터인 τ_k,r_k,s_k 가 공개된 경우 Groth16 프로토콜에 사용 될 수 있는 증거 재사용 알고리즘

1. 검증자는 $k=1,\cdots,N$ 에 해당하는 각각의 증거에 대하여 다음을 계산한다:

$$D_k \coloneqq [A_k]_G-[\alpha_k]_G-[r_k\delta_k]_G, \tag{26}$$

$$E_k \coloneqq [B_k]_H-[\beta_k]_H-[s_k\delta_k]_H. \tag{27}$$

2. 계산 결과는 다음의 관계식을 갖는다: $D_k = \sum_{i=1}^{m-2} v_i(x_k)[c_i]_G$, $E_k = \sum_{i=1}^{m-2} w_i(x_k)[c_i]_H$.
3. $[c_i]_G$와 $[c_i]_H$에 대하여, D_k와 E_k로부터 선형 연립방정식에 해를 구한다.
4. $[c_i]_G$와 $[c_i]_H$를 사용하여 $[C_N]_G$로부터 다음을 계산한다:

$$\left[\frac{h(x_N)t(x_N)}{\delta_N}\right]_G = [C_N]_G - \sum_{i=1}^{i=m-2} \frac{(\beta_N v_i(x_N) + \alpha_N w_i(x_N) + y_i(x_N))}{\delta_N}[c_i]_G - s_N[A_N]_G - r_N[B_N]_G - [r_N s_N \delta_N]_G. \quad (28)$$

5. (28)의 계산 결과를 사용하여 다음을 계산한다:

$$[h(x_N)t(x_N)]_G = \delta_N \left[\frac{h(x_N)t(x_N)}{\delta_N}\right]_G. \quad (29)$$

6. 새로운 $\tau_{N+1} := (\alpha, \beta, \delta, \gamma, x)$에 관하여 다음의 재사용 증거를 생성한다:

$$\pi' = \begin{pmatrix} [A']_G := [\alpha + r\delta]_G + \sum_{i=1}^{m} v_i(x_N)[c_i]_G, \\ [B']_H = [\beta + s\delta]_H + \sum_{i=1}^{m} w_i(x_N)[c_i]_H, \\ [C']_G = \sum_{i=1}^{i=m} \frac{(\beta v_i(x_N) + \alpha w_i(x_N) + y_i(x_N))}{\delta}[c_i]_G - \sum_{i=m-1}^{i=m} \frac{(\beta v_i(x) + \alpha w_i(x) + y_i(x))}{\delta}[c_i]_G \\ + \frac{[h(x_N)t(x_N)]_G}{\delta} + [sA' + rB' - rs\delta]_G. \end{pmatrix}. \quad (30)$$

알고리즘 5. 보안 파라미터인 τ_k, r_k, s_k가 공개된 경우 Groth16 프로토콜에 의해 생성된 재사용 증거 π'의 완전성을 조사하기 위해 LHS와 RHS를 계산 해 보면, 각각 (19)과 (20)의 관계에서 x 대신 x_N이 대입된 것과 같다. 따라서 x가 아닌 x_N에 대하여 (23)의 등호를 만족하므로, 재사용 증거 π'는 완전하다.

결과적으로, 악의적인 검증자가 보안 파라미터인 τ, r, s를 알고 있을 때 알고리즘 5. 보안 파라미터인 τ_k, r_k, s_k가 공개된 경우 Groth16 프로토콜의 방법으로 증거를 재사용 할 수 있음을 보였다. 그러나 다행히도, Groth16은 Prove과정에서 증명자에게 임의의 정수 값 r과 s를 선택하게 하여 공개하지 않도록 한다. 설령 검증자가 τ를 알고 있다 하여도 r, s를 모르는 한, (26)와 (27)을 계산 할 수 없기 때문에 적어도 알고리즘 5와 같은 방법으로는 증거를 조작하여 재활용 할 수 없다.

Q. (간결성) 증거의 길이는 간결한가?

A. Groth16에서 증거 데이터는 타원곡선의 좌표들이다. 따라서 증거의 길이의 단위를 타원곡선 좌표의 개수로서 정의하겠다. 결과적으로 증거의 길이는 $3n+m+5$으로, 연산 프로그램의 크기에 *선형적으로 비례한다.* 따라서 증거가 간결하다고 말할 수 있다.

프로토콜의 보안문제, 즉 건실성과 영지식성을 보장하기 위하여 매 증거 생성시마다 CRS가 갱신되어야 한다. 그리고 검증자는 검증을 위해 CRS와 증거 π를 함께 확보하여야 한다. 즉, 증거 π 뿐만 아니라 CRS도 매번 공유되어야 하는 데이터이기 때문에, CRS의 길이 또한 증거로 포함할

수 있다.

프로토콜 4를 확인하면 증거 π의 길이는 3으로 고정되어 있음을 알 수 있다. 반면 증거 CRS의 길이는 연산 프로그램의 크기에 따라 변한다. 어떤 연산 프로그램을 program analysis한 결과로서 써킷에 선의 개수가 m개, 곱셈 게이트의 개수가 n개가 존재한다면, CRS의 길이는 총 $3n+m+5$이다. 결과적으로, 증거 π와 CRS의 총 길이는 $3n+m+8$이며, 이는 연산프로그램의 크기인 m과 n에 선형적이다.

정리) 보안 파라미터 τ와 r,s의 중요성

요약하면, Groth16 프로토콜의 건실성을 보장하기위한 필요조건은 파라미터 리스트 τ가 거짓증명자에게 공개되지 않는것이다. 위의 분석을 통해 만약 τ가 거짓증명자에게 알려지면, 거짓증명자는 검증자를 기만 할 수 있음을 보였다. 그렇기 때문에 프로토콜에서는 τ를 ECC를 사용하여 암호화 한 후, 그 결과를 CRS라는 이름으로 공개하고, 원본 τ는 즉각 소멸시킨다. 이렇게 사용 후 소멸시켜야 하는 파라미터들을 "보안 폐기물(toxic waste)"이라 부른다.

한편 Groth16 프로토콜은 영지식성 보장을 위해 τ와 r,s라는 이중보안장치를 사용한다. 만약 τ와 r,s 모두 악의적인 검증자에게 공개된다면, 악의적인 검증자가 알고리즘 5를 이용해 기 제출된 증거들을 수집하여 재활용 할 수 있음을 보였다. 만약 τ나 (r,s)중 하나라도 노출되지 않는다면, 알고리즘 5를 사용 할 수 없다. (r,s)또한 τ와 같이 보안 폐기물이다.

다음은 τ 혹은 (r,s)가 노출되었을 때 발생할 수 있는 보안 문제를 정리한 표이다:

노출 파라미터	건실성	영지식성
τ	침해 가능	-
(r,s)	-	-
τ와 (r,s)	침해 가능	침해 가능

표 6 보안 파라미터 τ 혹은 (r,s)가 노출되었을 때 건실성과 영지식성의 침해 가능 여부

보안 파라미터인 τ를 생성하는 주체는 신뢰받는 제 3자(trusted third-party)인 것이 가장 이상적이지만, 제 3자를 정말로 신뢰할 수 있는가의 실용적인 문제가 고려 될 수 있다. 구체적으로 제 3자에 대한 신뢰를 법과 정책, 혹은 컴퓨터 프로토콜로써 완벽히 보장 할 수 있는가와 같은 문제이다. 여기서 "완벽한 신뢰"라는 표현은 τ가 증명자나 검증자에게 알려질 가능성이 전혀 없음을 의미한다. 실용적인 환경에서는 제 3자에게 완벽환 신뢰를 기대하기 어렵다. 한 가지 예로 보안인증 시스템과 같이 검증자가 서버이고 증명자가 클라이언트인 경우가 있을 수 있다. 이 경우 검증자가 제 3자를 선별 및 구성한다. 검증자가 악의적인 의도가 있다면 제 3자와 모의하여 증명자를 기만할 수 있다.

보안 파라미터 r,s 의 역할은 제 3자를 신뢰할 수 없는 경우에도 증명자의 비밀정보를 보호해주는 것이다. 파라미터 r,s 는 증명자가 생성하고 사용 후 소멸시키기 때문에 검증자 혹은 제 3자가 그 값을 알아내는 것이 불가능하다. 결과적으로, 파라미터 r,s 의 존재 덕분에 제 3자에 대한 의존이 필요 없어지게 된다.

참고) Multi-party computation of CRS: "Powers of Tau ceremony"

분석 결과와 같이, Groth16 프로토콜의 건실성과 영지식성은 파라미터 리스트 τ (tau)에 의존하고 있다. Tau는 증명자에게는 절대 공개되어서는 안되고, 검증자에게도 공개되지 않으면 더 좋다. 이 때문에 Groth16 프로토콜에서는 제3자가 tau를 생성 한 후 암호화하여 공개하고 원본 tau는 즉시 소멸한다.

제 3자에 대한 의존을 완화하기 위해, 파라미터 tau의 생성을 분산화 하려는 노력이 시도된 적이 있다. ZCash가 2017년에 개최한 "Powers of Tau ceremony"가 그러한 노력이다[21]. Powers of Tau는 전 세계의 참가자들이 공동으로 tau를 생성할 수 있도록 하는 행사이다. 누구나 기여 할 수 있지만, 누구도 최종 생성된 tau의 값은 알지 못한다. 다만 tau의 암호화 결과가 공개 될 뿐이다[22]. 2019년에는 Plonk 프로토콜을 개발한 회사 Aztec protocol에서도 Powers of Tau와 유사한 "multiparty computing ceremony"를 개최하였다[23].

아직 ZCash의 Powers of Tau가 정말로 안전한가에 관한 논란이 있을 수 있다. ZCash와 같이 ceremony의 형태로 다수가 참여하여 tau를 생성하면, tau의 값을 수시로 변경하기가 어렵다. 만약 tau가 증거를 생성하는데 한번 사용 된 직후 바뀌지 않는다면, 악의적인 검증자는 추가적인 조작 없이도 증거를 재사용 할 수 있다. 그럼에도 불구하고 분산화 계산을 도입하는 것은 훌륭한 접근이기도 하다. 더 많은 사람들이 참여할수록 tau를 악의적으로 조작하는 것이 어려워지기 때문이다. 실증연구를 통해 참여자 수와 tau의 변경 주기간의 trade-off 관계가 사용 환경에 따라 적절히 최적화 된다면, 보다 안전한 증명 프로토콜이 될 수 있을 것이다.

---

[21] Zcash 세레모니 개최 포스트: https://www.zfnd.org/blog/powers-of-tau

[22] Zcash 세레모니 종료 포스트: https://www.zfnd.org/blog/conclusion-of-powers-of-tau

[23]Aztec 세레모니 개최 포스트: https://medium.com/aztec-protocol/aztec-announcing-our-ignition-ceremony-757850264cfe

zk-SNARK: MATLAB 실습

저자는 소규모의 연산 프로그램에 대해 Groth16의 증명 프로토콜을 실습 할 수 있도록 도와주는 오픈소스 구현체를 제공하고 있다 (링크: https://codeocean.com/capsule/8850121/tree/v2). 구현체는 MATLAB으로 구현되었으며, 링크를 통해 MATLAB이 설치되지 않은 PC에서도 실행이 가능하다. 먼저 program analysis와 QAP 등이 어떻게 구현되었는지 살펴 본 후, 간단한 예제를 통해 프로토콜을 수행해 보는 실습을 하겠다.

구현 설명서

앞서 그림 5와 함께 살펴보았듯이, Groth16은 program analysis, QAP, ECC, 그리고 pairing의 과정을 거친다. Program analysis는 주어진 연산 프로그램 f 를 연산조각들로 분해하는 과정이다. QAP는 연산조각들을 다항식과 계수의 집합으로 치환한다. 다항식과 계수를 조합하여 CRS와 증거를 생성한다. ECC는 증거를 암호화하여 증명 프로토콜의 건실성과 영지식성을 보장한다. Pairing은 증거가 암호화된 상태에서도 프로토콜이 완전성을 보장받을 수 있도록 한다.

Program analysis의 구현

프로토콜의 구성요소들 중 program analysis는 R1CS라는 방식으로 구현되었다[24]. R1CS는 그림 6과 같은 써킷을 행렬연산 형태로 표현하는 것이다. 구체적으로, R1CS는 벡터 $\mathbf{R}$ 과 행렬 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 를 생성하고, 이들은 다음과 같은 관계를 만족한다:

$$\begin{bmatrix} \mathbf{R} \end{bmatrix}^T \underbrace{\begin{bmatrix} \mathbf{A} \end{bmatrix}}_{m \times n} \circ \begin{bmatrix} \mathbf{R} \end{bmatrix}^T \begin{bmatrix} \mathbf{B} \end{bmatrix} = \begin{bmatrix} \mathbf{R} \end{bmatrix}^T \begin{bmatrix} \mathbf{C} \end{bmatrix}, \tag{31}$$

여기서 연산 $\circ$는 Hadamard 곱, 즉 성분별 곱(elementwise product)이며, 크기가 같은 두 벡터 혹은 두 행렬에 대하여 같은 크기의 벡터 혹은 행렬을 출력하고, 출력 벡터 혹은 행렬의 성분은 같은 위치의 입력 성분끼리 곱한 결과이다. 예를 들어, $[1 \quad 2] \circ [3 \quad 4] = [3 \quad 8]$ 이다. 써킷에 존재하는 모든 곱셈 게이트에 대하여, 행렬 $\mathbf{A}$ 는 게이트의 왼쪽 입력, 행렬 $\mathbf{B}$ 는 게이트의 오른쪽 입력, 행렬 $\mathbf{C}$ 는 게이트의 출력이 기록되어 있으며, 행렬의 서로 다른 게이트의 정보는 서로 다른 열에 기록된다. 벡터 $\mathbf{R}$ 에는 써킷의 모든 곱셈 게이트의 중간 계산 값들이 기록된다. 즉, $\mathbf{R}$ 의 값들이

[24] R1CS는 이더리움의 github에 python으로도 구현되어있다:
https://github.com/ethereum/research/blob/master/zksnark/code_to_r1cs.py

프로토콜 4에서 사용되는 QAP 계수들 c_i 이다. 따라서 각 행렬들의 크기는 $m \times n$ 이다. 예를 들어, 그림 6과 같은 $f : C_6 = (C_1 + C_2) C_3 C_4$ 에 대응되는 써킷은 다음과 같은 R1CS 형태로 표현될 수 있다:

$$\underbrace{\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix}^T}_{\mathbf{R}} \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}_{\mathbf{A}} \circ \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix}^T \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}}_{\mathbf{B}} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix}^T \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{C}}. \tag{32}$$

식 (32)을 계산 해 보면, 첫 번째 열은 $C_3 C_4 = C_5$ 를 표현하며, 두 번째 열은 $(C_1 + C_2) C_5 = C_6$ 을 표현하기 때문에 $\mathbf{R}, \mathbf{A}, \mathbf{B}, \mathbf{C}$ 가 그림 6의 써킷을 대변한다고 말할 수 있다. 한편, 주어진 연산 프로그램 f 에 대하여, 써킷의 형태가 다양 할 수 있는 것과 마찬가지로 R1CS의 결과도 다양할 수 있다. 예를 들어, 그림 6에 사용된 연산 프로그램이 다음과 같은 R1CS로써 표현될 수도 있다:

$$\begin{bmatrix} 1 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_7 \\ C_8 \\ C_9 \\ C_{10} \\ C_6 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_7 \\ C_8 \\ C_9 \\ C_{10} \\ C_6 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_7 \\ C_8 \\ C_9 \\ C_{10} \\ C_6 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{33}$$

식 (32)의 R1CS와 식 (33)의 R1CS는 모두 $f : C_6 = (C_1 + C_2) C_3 C_4$ 라는 같은 연산프로그램을 다르게 표현 한 것이다 (C_7, C_8, C_9, C_{10} 은 중간 계산 값이다). 식 (33)의 R1CS는 식 (32)의 R1CS에 비해 행과 열의 개수가 더 많다. R1CS의 행의 개수는 써킷에 존재하는 선의 개수와 같으며, 열의 개수는 써킷에 존재하는 곱셈 게이트의 개수와 같다. 이전의 분석을 통해 써킷의 선과 곱셈 게이트가 많을수록 증거의 길이가 길어짐을 확인하였다. 따라서 식 (32)의 R1CS가 식 (33)보다 더 간결한 증거를 야기한다.

QAP의 구현

Program analysis (R1CS) 이후 QAP과정을 거쳐 다항식을 생성한다. QAP는 써킷의 구조를 다항식의 형태로 저장하는 것인데, 이전의 R1CS에서 이미 써킷의 구조를 행렬의 형태로 저장해 놓았다. 행렬 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 가 바로 그것이다. 따라서 QAP과정에서 해야 할 일은 행렬 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 를 다항식으로

변환하는 것이다. 먼저 임의의 근(root)들을 설정 한 후, 근 다항식 $t(x)$을 정의한다. 근의 개수는 R1CS 행렬의 열 개수와 일치하도록 정한다. 이제 행렬 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 각각의 행을 다항식으로 변환한다. 다항식의 입력 변수를 x 출력 변수를 y라 할 때, 각각의 행의 성분들이 y의 표본이 되고, 근 다항식의 근들이 x의 표본이 되도록 설정한다. 이렇게 정의한 표본을 바탕으로 보간법을 수행하여 연속형태의 다항식을 생성한다. 이 과정을 모든 행렬의 모든 $3m$개의 행에 대하여 반복하여 총 $3m$개의 다항식을 보간법으로 생성한다. 예를 들어, 식 (32)의 R1CS인 경우 근의 개수는 2개이며, 각각의 근을 $r_1 = 1$, $r_2 = 2$와 같이 설정 할 수 있다. 그러면 근 다항식은 $t(x) = (x-1)(x-2)$가 된다. 이 예시에서 행렬 $\mathbf{B}$의 네 번째 행에 대응되는 다항식의 표본은 $(1,1),(2,0)$이며, 이 두 개의 표본을 바탕으로 보간법을 수행한다.

보간법은 선형대수를 사용하여 수행 할 수 있다. 번호 $i = 1,\cdots,n$에 대하여 근 다항식의 근을 r_i라 명명하겠다. 여기서 근의 개수는 써킷의 곱셈 게이트의 개수 n과 같다. 어떤 QAP 다항식 $y = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$ 에 대해 n개의 보간 표본이 $(r_1, y_1),(r_2, y_2),\cdots,(r_n, y_n)$일 때, 표본의 x성분과 y성분 사이에는 다음과 같은 선형 관계가 성립된다:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \underbrace{\begin{bmatrix} r_1^0 & r_1 & \cdots & r_1^{n-1} \\ r_2^0 & r_2 & \cdots & r_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ r_n^0 & r_n & \cdots & r_n^{n-1} \end{bmatrix}}_{\mathbf{V}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}. \tag{34}$$

이때 전달행렬 $\mathbf{V}$가 Vandermonde 행렬[25]이라는 특수한 형태가 되고, Vandermonde 행렬은 항상 역행렬이 존재한다. 이제 연속 다항식의 계수 $a_0,\cdots,a_n$을 찾는 보간법은 다음의 선형 역문제를 푸는 것이다:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \mathbf{V}^{-1} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{35}$$

근 다항식의 근 r_i는 $i = 1,\cdots,n$에 대해 서로 다르기만 하다면 임의로 설정하여도 문제가 없지만, 보간법을 더 계산효율적으로 수행하기 위해 특수하게 설정하는 경우도 있다. 전달행렬 $\mathbf{V}$는 특수성으로 인해 언제나 역행렬 $\mathbf{V}^{-1}$가 존재하기는 하지만, $\mathbf{V}^{-1}$를 계산하는 것은 여전히 많은 계산량을 필요로 한다. 계산 량을 줄이기 위해, 다항식의 근을 적절히 설정하여 행렬 $\mathbf{V}$를 DFT 행렬로 만들 수 있다. DFT행렬이란 Vandermonde 행렬의 특수한 경우로, k,l번째 성분이 $e^{j2\pi\frac{kl}{n}}$인 행

[25] Vandermonde 행렬: https://en.wikipedia.org/wiki/Vandermonde_matrix

렬이다. 근 다항식의 근들을 $i=1,\cdots,n$ 에 대해 $r_i=e^{j2\pi\frac{i}{n}}$ 로 설정하면 $\mathbf{V}$ 를 DFT 행렬로 만들 수 있다. 여기서 $j=\sqrt{-1}$ 는 허수이다. $\mathbf{V}$ 가 DFT 행렬이 되면 좋은 점은 fast Fourier transform(FFT)[26] 알고리즘을 사용 할 수 있게 된다는 것이며, 그러면 (35)의 역문제를 $O(n\log n)$ 이라는 아주 적은 계산 량으로 풀 수 있다.

ECC의 구현

MATLAB 구현체에서 ECC를 위해 사용하는 타원곡선은 $E:y^2=x^3+x$ 이며, x 와 y 는 유한 체 F_p 에서 정의되었다. 여기서 체의 크기는 $p=71$ 이다. 이렇게 정의된 타원곡선 E 는 supersingular 타원곡선[27]이라 불리며, pairing에 유리한 특성을 가지고 있다.

타원곡선 E 를 pairing에 조금 더 유리하게 사용하기 위해서 확장 체 (extension field)의 개념을 함께 사용한다. 유한 체 F_p 의 order가 2인 확장 체는 F_{p^2} 으로 표기된다. 한 예로, 확장체는 F_p 의 원소에 허수인 $j=\sqrt{-1}$ 의 사용을 허용함으로써 확장 가능하다. 예를 들어, $F_3=\{0,1,2\}$ 의 확장 체는 $F_{3^2}=\{0,1,2,j,j2,1+j,1+j2,2+j,2+j2\}$ 일 수 있다. 만약 $F_p\times F_p$ 의 한 원소인 (x,y) 가 타원곡선 E 상의 한 점이라면, $(-x,jy)$ 또한 타원곡선 E 상의 한 점인 것을 알 수 있다. 그리고 그 점 $(-x,jy)$ 는 $(-x,jy)\in F_{p^2}\times F_{p^2}$ 라는 것을 알 수 있다. 다시 말하면, F_p 상에서 정의된 E 의 점들을 F_{p^2} 으로 확장하는 것은 간단히 수행 가능하다.

프로토콜 4에 소개된 Groth16의 증명 프로토콜에서는 타원곡선 E 의 점들 중 두 개의 generator 인 G 와 H 를 사용한다. MATLAB 구현체에서도 G 와 H 를 사용하며, pairing에 유리하도록 사용하기 위해 G 와 H 가 다음과 같은 관계가 되도록 선택하였다: G 는 타원곡선 E 의 generator들 중 임의로 선택되었다; $G=(x_G,y_G)$ 라 할 때, $H=(-x_G,jy_G)$ 이다. 따라서 $G\in F_p$ 이고 $H\in F_{p^2}$ 이다.

Pairing의 구현

Pairing은 Weil pairing을 선택하여 구현하였다. 다른 pairing들인 Ate pairing, Tate pairing도 모두 Weil pairing의 계산원리에서 조금만 변형하면 구현이 가능하다. Weil pairing은 D. Boneh와 M. Franklin의 논문인 "Identity-based encryption from the Weil pairing"[28]의 Appendix에 기술된 정의

---

[26] Fast Fourier transform: https://en.wikipedia.org/wiki/Fast_Fourier_transform

[27] p 의 값은 $p\equiv 3 \bmod 4$ 를 만족시키는 선에서 조절 가능하다. Supersingular elliptic curves에 관한 설명은 다음의 문헌에서 참조할 수 있다: D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *SIAM J. of Computing*, 2003.

[28] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *SIAM J. of*

와 계산 방법을 그대로 사용하여 구현하였다. 계산에는 Miller algorithm[29]이 사용된다. 특이사항으로, $O\left(p^{-1}\log p\right)$ 의 확률로 pairing에 실패 할 수 있다. 그러나 p 가 커질수록 이 확률은 매우 작아져서 실용적인 환경에서는 실패 하는 경우가 거의 없고, 만약 실패하더라도 다시 시도하면 된다.

Groth16 증명 프로토콜 사용 예제

그림 9은 MATLAB으로 구현된 Groth16 프로콜의 실행 결과이다. 연산 프로그램은 $f: y=(c_1+c_2)(c_1-4)$ 이다. 비공개 정보는 $c_1=2$ 과 $c_2=4$ 이다. Program analysis (R1CS) 결과로 6개의 써킷 선과 3개의 곱셈 게이트가 형성되었다. R1CS의 결과를 토대로 QAP 다항식들과 복원 다항식 $p(x)$, 몫 다항식 $h(x)$ 를 성공적으로 생성하였다. 보안 파라미터와 QAP 결과를 종합하여 CRS와 증거를 성공적으로 생성하였고, CRS의 길이는 $3n+m+5=20$ 이며, 증거의 길이는 3 이다. CRS와 증거를 사용하여 검증이 성공적으로 수행되었다.

```
Program to prove:
(c1 + c2)*(c1 - 4)

with inputs
     2     4

Program analysis result:
Elapsed time is 0.780524 seconds.
The number of wires: 6
The number of multiplication gates: 3
R1CS has successfully been made

QAP result:
Elapsed time is 0.142742 seconds.
P(x) is valid
H(x) has been computed
H(x) is valid
```

```
Groth16 Setup result:
Elapsed time is 29.664750 seconds.
CRS has been made
Length of CRS is: 20(elliptic curve points)
CRS is valid

Groth16 Prove result:
Elapsed time is 16.238499 seconds.
Proof has been made
Length of proof is: 3(elliptic curve points)
Proof is complete

Groth16 Verify result:
Elapsed time is 0.086056 seconds.
Verification Success
```

그림 9 MATLAB으로 구현된 Groth16 증명 프로토콜의 실행 결과

R1CS의 결과로 생성된 행렬 $\mathbf{A},\mathbf{B},\mathbf{C}$ 와 벡터 $\mathbf{R}$ 은 각각 다음과 같다:

Computing, 2003.

[29] V. S. Miller, "The Weil Pairing, and Its Efficient Calculation," *Journal of Cryptology*, 2004.

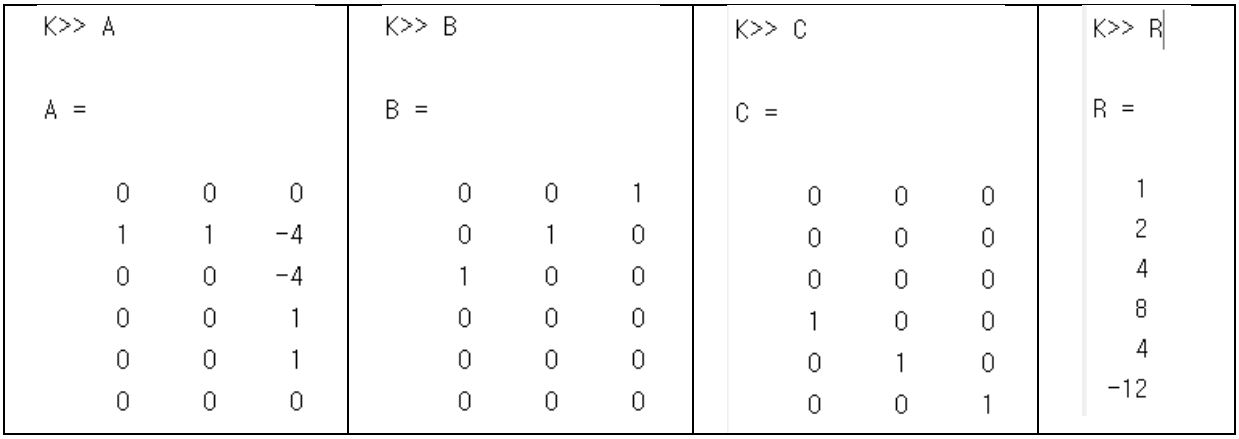


그림 10 R1CS로 생성된 행렬 A,B,C와 벡터 R

그림 10에서 생성된 $\mathbf{R}$ 은 $\mathbf{R}(c_1,c_2)=\left[1,c_1,c_2,c_3=c_1c_2,c_4=c_1^2,y=-4c_2-4c_1+c_3+c_4\right]$ 에 각각 $c_1=2$ 와 $c_2=4$ 를 대입한 결과이다. $\mathbf{R}(c_1,c_2)$ 와 행렬 $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ 를 사용하여 (31)을 계산하여 왼쪽 열부터 읽으면 다음의 관계식들을 순서대로 얻을 수 있다:

$$\begin{cases} c_1c_2=c_3, \\ c_1^{\,2}=c_4, \\ -4c_1-4c_2+c_3+c_4=y. \end{cases} \tag{36}$$

관계식 (36)는 연산 프로그램 f 와 일치하므로, R1CS가 성공적으로 수행되었음을 알 수 있다.

QAP를 수행하기 위해, 근 다항식 $t(x)$ 의 근을 $r_1=1$, $r_2=2$, $r_3=3$ 으로 설정하였다. 따라서 $t(x)=(x-1)(x-2)(x-3)$ 이다. 행렬 $\mathbf{A}$ 의 두 번째 행에 대응되는 다항식, 즉 $w_2(x)$ 의 표본은 $(1,1),(2,1),(3,-4)$ 이며, 보간법을 적용한 결과는 $w_2(x)=-2.5x^2+7.5x-4$ 로 계산되었다. 마찬가지로, 행렬 $\mathbf{C}$ 의 4번째 행에 대응되는 다항식, 즉 $y_4(x)$ 의 표본은 $(1,0),(2,0),(3,0)$ 이며, 보간법을 적용한 결과는 $y_4(x)=0.5x^2-1.5x+1$ 로 계산되었다. 이러한 방법으로 $i=1,\cdots,6$ 에 대하여 모든 다항식 $w_i(x)$, $v_i(x)$, $y_i(x)$ 를 생성하였다 (그림 11). 이를 토대로 $p(x)$ 를 계산하였고, 마지막으로 $h(x)=p(x)/t(x)$ 를 계산하였다. $p(x)$ 에 $k=1,\cdots,3$ 에 대하여 $x=r_k$ 를 대입한 후 $p(r_k)=0$ 임을 확인하여 QAP의 결과가 올바른지 확인하였다 (그림 12).

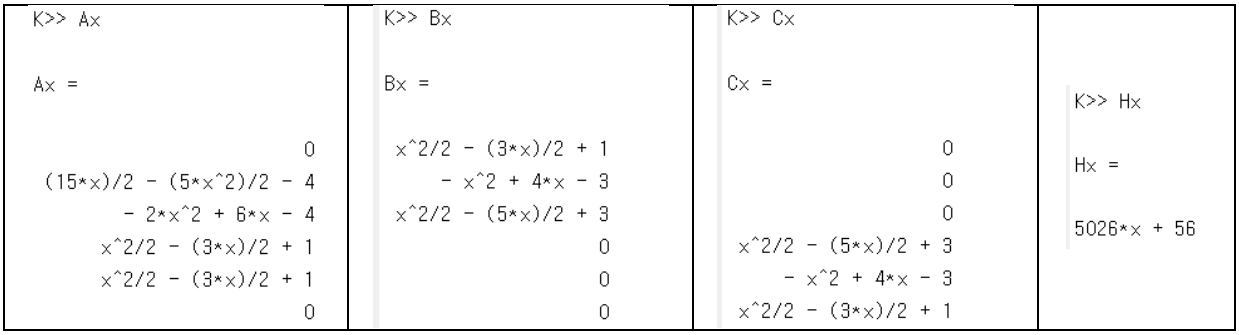


그림 11 QAP 왼쪽부터 다항식 $w_i(x)$, $v_i(x)$, $y_i(x)$와 몫 다항식 $h(x)$의 계산 결과. 행렬 Ax와

Bx, Cx 의 첫 번째 행부터 마지막 행까지 순서대로 $i=1,\cdots,6$ 에 대응된다.

```
%% Debuging code: Check the validity of P(x)
checksum=0;
for i=Roots_of_Z
    checksum=checksum+mod(double(subs(Px,x,i)),q);
end
display('QAP result:')
toc;
if checksum==0
    display('P(x) is valid')
else
    display('Invalid P(x), check the Lagrange interpolation')
end
```

그림 12 복원 다항식 $p(x)$가 올바르게 생성되었는지 확인하는 코드

한편 MATLAB으로 구현한 프로토콜 4의 QAP는 유한 체 F_q 에서 작동한다. 여기서 $q=p^2-1$ 이다[30]. 따라서 QAP 다항식들의 계수들이 F_q 의 원소여야 한다. 따라서 $i=1,\cdots,6$ 에 대한 모든 다항식 $w_i(x)$, $v_i(x)$, $y_i(x)$ 의 계수들에 존재하는 실수(유리수)를 정수로 변환하기 위해 임의의 상수 u 를 사용 할 수 있다. 이때 u 는 모든 계수들의 분모의 최소 공배수이다. QAP의 완전성을 위한 관계식인 $\left(\sum_{i=1}^{6}c_i w_i(x)\right)\cdot\left(\sum_{i=1}^{6}c_i v_i(x)\right)-\left(\sum_{i=1}^{6}c_i y_i(x)\right)=h(x)t(x)$ 의 관계를 유지하면서 다항식의 계수들을 정수로 변환하는 방법은 $w_i(x)\leftarrow uw_i(x)$, $v_i(x)=uv_i(x)$, $y_i(x)=u^2y_i(x)$, $h(x)\leftarrow u^2h(x)$ 로 변환하는 것이다. 이렇게 하면 $\left(u\sum_{i=1}^{6}c_i w_i(x)\right)\cdot\left(u\sum_{i=1}^{6}c_i v_i(x)\right)-\left(u^2\sum_{i=1}^{6}c_i y_i(x)\right)=u^2h(x)t(x)$ 이 되어 QAP의 완전성이 유지된다.

[30] QAP가 F_q 에서 작동하면, F_{p^2} 에서 정의된 타원곡선 점들의 pairing을 사용하여 QAP의 완전성, 즉 $p(x)=h(x)t(x)$를 검증 할 수 있다.

```
[NUM,DEN]=rat([Ap;Bp;Cp]);
denoms=unique(DEN);
mul=1;
for i=1:length(denoms)
    mul=lcm(mul,denoms(i));
end
Ap=round(Ap*mul);
Bp=round(Bp*mul);
Cp=round(Cp*mul^2);
```

그림 13 유리수 계수들의 분모의 최소공배수를 곱하여 계수를 정수로 변환하는 코드

QAP 결과를 ECC로 암호화하기에 앞서, 먼저 타원곡선의 두 generator로 사용될 G와 H를 선택하였다. G의 좌표는 $(11,8)$이고, H의 좌표는 $(-11 \bmod 71, j8 \bmod 71) = (60, j8)$이다 (그림). 타원곡선의 모든 점과 각 점의 특성은 링크 https://graui.de/code/elliptic2/ 에서 확인 할 수 있으며, MATLAB 구현체에서도 주어진 타원 곡선의 모든 점을 찾는 함수(*EC_points.m*)와 generator를 찾는 함수(*EC_order.m*)가 제공된다.

```
K>> g

g =

        q: 71
        k: 1
        a: 1
        b: 0
        x: 11
        y: 8
    order: 72
```

```
K>> h

h =

        q: 71
        k: 2
        a: 1
        b: 0
        x: 60
        y: 0.000000000000000 + 8.000000000000000i
    order: 72
```

그림 14 Generator G와 H

준비된 QAP 결과와 타원곡선을 사용하여, 프로토콜 4의 Setup, Prove, Verify를 순서대로 수행한다. Setup과정과 Prove과정에서 선택되는 보안 파라미터인 τ와 r,s는 매번 무작위로 선택되도록 설계되었다. Setup 과정에서 CRS 데이터인 길이 14의 σ_G와 길이 6의 σ_H를 생성하며, 구현체에서는 σ_G가 `sigma1_1`, `sigma1_2`, …, `sigma1_5`에, 그리고 σ_H가 `sigma2_1`, `sigma2_2`에 분리되어 저장된다. Prove 과정에서는 길이 3의 증거 π를 생성하며, 구현체에서는 `proof`로 명명되었다. Verify 과정에서 계산되는 LHS와 RHS는 모두 F_p의 원소이며, $LHS = RHS$임을 확인하여 최종 검증결과를 출력한다.