Python 에서 Go 로의 언어 전환을 통한 Accountable Ring Signature 구현 성능 향상

박지은 [1], 문현솔 [1], 김연희 [2], 김종길 [3]
[1] 이화여자대학교 사이버보안학과 학부생
[2] 이화여자대학교 인공지능융합전공 박사과정
[3] 이화여자대학교 사이버보안전공 교수

lucykorea414@ewha.ac.kr, tranquil@ewha.ac.kr, heedong@ewha.ac.kr, jongkil@ewha.ac.kr

Performance Improvement of Accountable Ring Signature through Language Transition from Python to Go

Jee-Eun Park[1], Hyun-Sol Mun[1], Yeon-Hee Kim[2], Jong-Kil Kim[3]
[1]Dept. of Cyber Security, Ewha Womans University
[2]Dept. of Artificial Intelligence Convergence, Ewha Womans University
[3]Dept. of Cyber Security, Ewha Womans University

요 약

본 논문은 Accountable Ring Signature(ARS) 프로토콜의 기존 Python 구현을 Go 언어로 전환하여 성능을 향상하고 실제 서비스 환경에서의 적용 가능성을 평가하였다. 실험 결과, Go 기반 구현은 전체 실행 시간에서 약 3.6 배의 성능 향상을 보였으며, 정적 타입 기반 구조와 명시적 연산 구현을 통해 유지보수성과 확장성 또한 개선되었다. 이를 통해 Go 언어가 고성능 암호 프로토콜 구현에 효과적이며 ARS 의 실용적 적용 가능성을 높이는 데 기여할 수 있음을 확인하였다.

1. 서론

Accountable Ring Signature(ARS)[1]는 서명자의 익명성을 보장하면서도 권한을 가진 개체가 서명자를 식별할 수 있는 암호학적 서명 기법이다. 전통적인 링 서명(Ring Signature)에 기반하여 ElGamal 암호화를 활용한 식별 메커니즘을 추가한 ARS 는 전자 투표, 블록체인과 같이 프라이버시 보호와 책임성이 동시에 요구되는 다양한 분야에 유용하게 활용될 수 있다. 기존의 ARS 구현은 Python 언어를 기반으로 이루어졌으나[2] 큰 정수 및 반복적 지수 연산 수행 시 성능 한계를 나타내 실제 서비스 환경 적용에 제약이 있다.

본 연구는 ARS 프로토콜을 Go 언어[3]로 재구현하고 기존 Python 구현과 성능 및 구조를 비교한다. Go 언어는 정적 타입 시스템과 컴파일 기반 실행 구조를 갖추어 정확하고 효율적인 큰 정수 연산이 가능하며 경량 병렬 처리(goroutine)를 통해 고성능 암호 프로토콜 구현에 적합한 환경을 제공한다. 본 연구에서는 두 구현 간의 성능 및 정확성을 평가하고, ARS 프로토콜의 실제 서비스 적용 가능성을 검증한다.

2. ARS 구현 분석 및 Go 언어 기반 재구성

2.1 전체 구현 구조 비교

Python 기반 ARS 구현은 클래스와 함수 중심의 단순한 구조로 구성되어 있어 알고리즘 검증 및 프로토타이핑 단계에서는 높은 유연성과 개발 편의성을 제공한다. 그러나 동적 타입과 인터프리터 기반 실행 방식으로 인해, 큰 정수 연산이나 타입 안정성이 요구되는 환경에서 성능 저하 및 오류 발생 가능성이 내재한다. 반면 Go 는 컴파일러 기반의 정적 타입 언어로 구조체 중심 설계를 통해 데이터 흐름과 연산 구조를 명확히 정의하며 빌드 단계에서 오류를 사전 검출할 수 있다. 또한 math/big 패키지를 활용한 정수 연산 지원을 통해 연산 정확성과 실행 안정성을 확보할 수 있으며 이는 코드의 유지보수성과 실행 신뢰성을 동시에 향상시키는 데 기여한다.

2.2 주요 기능 구현 비교

ARS 프로토콜의 핵심 기능인 키 생성, 서명, 검증, ElGamal 암복호화 구현에서는 Python 과 Go 언어 간의 구조적 차이가 뚜렷하다. Python 은 간결한 문법과 오버플로우 없는 정수형 덕분에 개발과 이해가 용이하나, 예외 처리나 입력 검증이 미흡해 안정성과 디버깅 측면에서 제약이 있다. 반면 Go 는 big.Int 를 활

용해 정밀한 연산이 가능하고, 각 연산 단계에서 명시적 검증과 오류 처리를 통해 신뢰성과 유지보수성이 우수하다. 특히 ElGamal 암복호화는 정수 기반으로 정확히 구현되어 계산 일관성과 서명자 식별의 신뢰성을 높인다.

2.3 다항식 연산 구현 비교

ARS 프로토콜에서는 다항식 곱셈이 핵심 연산 중 하나로 사용된다. 기존 Python 구현은 외부 라이브러리인 sympy.poly 를 활용하여 다항식을 심볼릭 객체로 정의하고 연산을 수행하며 구현의 직관성과 간결성이 뛰어나다. 그러나 객체 생성, 동적 타입 검사, 타입 변환 등이 수반되어 실행 환경에서 성능 저하를 유발할 수 있다.

```
func polyMulExact(p1, p2 []*big.Int) []*big.Int {
  deg1 := len(p1)
  deg2 := len(p2)
  result := make([]*big.Int, deg1+deg2-1)

  for i := range result {
    result[i] = big.NewInt(0)
  }
  for i := 0; i < deg1; i++ {
    for j := 0; j < deg2; j++ {
      term := new(big.Int).Mul(p1[i], p2[j])
      result[i+j].Add(result[i+j], term)
    }
  }
  return result
}
```

(그림 1) ARS 프로토콜에서 사용되는 다항식 곱셈 함수의 Go 구현

반면 Go 는 다항식 연산을 위한 전용 라이브러리를 기본적으로 제공하지 않기 때문에 본 연구에서는 ARS 프로토콜 내 해당 연산을 직접 구현하였다. 다항식은 []*big.Int 형태의 계수 벡터로 표현되며 서명자 인덱스 표현 및 링 구성원 증명과 같은 핵심 연산에 사용된다. (그림 1)은 본 구현에서 실제로 사용된 다항식 곱셈 함수인 polyMulExact 의 코드이다. 이 함수는 ARS 서명 생성 및 검증 과정에서 사용되는 다항식 곱셈을 수행하며 모든 연산은 big.Int 를 기반으로 정수 오버플로우 없이 처리된다. 직접 구현된 이 연산은 연산 병목을 줄이고 외부 라이브러리 의존성을 제거함으로써 전체 프로토콜의 성능과 이식성을 향상시키는 데 기여한다.

3. 실험 및 분석

<표 1>은 ARS 프로토콜의 주요 단계에 대한 실행 시간 비교 결과를 나타낸다. 측정 항목은 서명 생성(Signing), 서명 검증(Signature Verification), 개봉(Open), 개봉 검증(Opening Verification), 그리고 전체 실행 시간(Total Time)으로 구성된다.

<표 1> ARS 프로토콜 핵심 단계별 성능 비교

단계	Python 실행 시간 (ms)	Go 실행 시간 (ms)
Signing	8.5	3.2
Signature Verification	4.5	1.0
Opener Open	0.5	0.2
Opener Verification	1.3	0.5
Total Time	18.6	5.2

서명 생성(Signing) 단계에서 Go 구현은 Python 대비 약 2.7 배 빠르게 수행되었으며, 서명 검증(Signature Verification) 단계에서는 4.5 배 빠른 성능을 보였다. 개봉(Opener Open) 및 개봉 검증(Opener Verification) 단계 역시 각각 2.5 배, 2.6 배 이상 빠르게 실행되었다. 특히 전체 실행 시간(Total Time)은 Python 이 18.6ms 인 데 반해 Go 는 5.2ms 로, 약 3.6 배에 달하는 성능 향상이 확인되었다. 성능 외에도 Go 는 정적 타입 시스템과 직접 구현한 다항식 연산을 통해 코드의 유지보수성과 확장성에서 우수한 특성을 보였다.

4. 결론

본 연구는 ARS 프로토콜의 기존 Python 기반 구현을 Go 언어로 정밀하게 재구현하고 성능 및 구조적 측면에서 두 구현을 비교·분석하였다. 실험 결과, Go 구현은 서명 생성, 서명 검증, 개봉, 개봉 검증 등 주요 단계에서 Python 대비 2.5 배에서 최대 4.5 배 이상의 성능 개선을 보였으며 전체 실행 시간도 약 3.6 배 단축되었다. 또한 Go 의 정적 타입 기반 구조와 연산 로직의 명시적 구현은 코드의 일관성과 안정성을 높였고 외부 라이브러리 의존 최소화를 통해 유지보수성과 이식성 또한 향상되었다. 이러한 결과는 ARS 프로토콜이 실제 환경에서도 효율적으로 적용 가능함을 보여주며, Go 언어가 고성능 암호 프로토콜 구현에 적합한 개발 환경을 제공함을 확인하였다.

ACKNOWLEDGEMENT

이 논문은 2023 년도 한국연구재단 한중협력연구사업 지원을 받아 수행된 연구임 (No. RS-2023-NR119899).
이 논문은 2024 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. RS-2024-00357189).

참고문헌

[1] Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C., Short Accountable Ring Signatures Based on DDH, Computer Security – ESORICS 2015, Vienna, Austria, 2015, pp. 243–265.

[2] MollyIKnight, AccountableRingSigDDH, https://github.com/MollyIKnight/AccountableRingSigDDH, 2025.04.14 에 확인함

[3] Meyerson, J., The Go Programming Language, IEEE Software, vol. 31, no. 5, pp. 104–104, Sept.-Oct. 2014.