2024 암호분석 경진대회

6번 문제

첨부파일 개요

- `contest.c`: 전체 최적화 기법이 적용된 파일
- `base.c`: 문제에서 제공한 기본 파일
- `oi*.c`: i번째 최적화가 적용된 파일
- `mix*.c`: 최적화 기법을 조합하여 적용한 파일
- `stat.py`: 최적화 기법의 속도 비교를 위한 Python 스크립트

1 목표

본 문제의 목표는 코드에 제시된 암호화/인증생성 알고리즘의 동작방식을 이해하고 최적화를 진행하는 것이다.

2 최적화 기법

본 팀이 적용한 최적화 기법을 아래에 정리하였다. 알고리즘의 일부분에 적용되는 기법은 알고리즘의 처리 순서에 맞추어 소개하고, 전체 알고리즘에 적용되는 기법을 이후에 설명한다.

2.1 알고리즘 실행 단계별 최적화

2.1.1 실행 전 계산된 (Precomputed) 값 사용 (적용 함수: `key_scheduling`)

`key_scheduling` 함수의 `ROL(CONSTANT_X, (i + OFFSET_Y) % 8)`형태의 식은 8라운드마다 동일한 값이 반복되며, 연산에 상수값만을 사용하기 때문에 입력값인 Plaintext와 Master Key에 의존하지 않는다. 따라서, 아래 64개 값을 실행 전에 계산하고 실행 중에 불러오는 방식을 사용하였다.

```
uint32_t cache[64] = {
     27,  57, 203,  56, 163, 183,  59,  57,
     54, 114, 151, 112,  71, 111, 118, 114,
    108, 228,  47, 224, 142, 222, 236, 228,
    216, 201,  94, 193,  29, 189, 217, 201,
    177, 147, 188, 131,  58, 123, 179, 147,
     99,  39, 121,   7, 116, 246, 103,  39,
    198,  78, 242,  14, 232, 237, 206,  78,
    141, 156, 229,  28, 209, 219, 157, 156
};
```

2.1.2 함수 인라인화 및 Loop Unrolling (적용 함수: `block_encryption`)

`block_encryption` 함수는 매 라운드마다 7회의 `ROUND_FUNC` 함수를 호출한다. 해당 함수는 단순 연산 및 대입만을 수행하기 때문에, `block_encryption`함수에 인라인 병합이 가능하다. 또한 반복 횟수가 7회로 고정되어 있기 때문에 loop unrolling을 적용하여 반복문을 제거하였다.

2.2 알고리즘 전체 대상 최적화

2.2.1 8비트 연산을 64비트 연산으로 대체

기존 코드의 모든 연산은 1바이트 (`uint8_t`) 단위로 진행된다. 대입, XOR등 동시 처리해도 결과가 바뀌지 않는 연산들을 8바이트 (`uint64_t`) 단위로 처리하여 연산량을 절감하였다.

2.2.2 개별 함수에 컴파일러 최적화 적용

개별 함수들의 정의에 gcc 컴파일러에서 제공하는 최적화를 적용하여 성능을 향상시켰다.

3 실험 및 결과

3.1 실험 환경

실험 환경은 아래와 같다.

OS	Ubuntu 22.04.4 LTS
VM Host	VirtualBox Version 7.0.8
Compiler	gcc 13.1.0

Table 1: 실험 환경

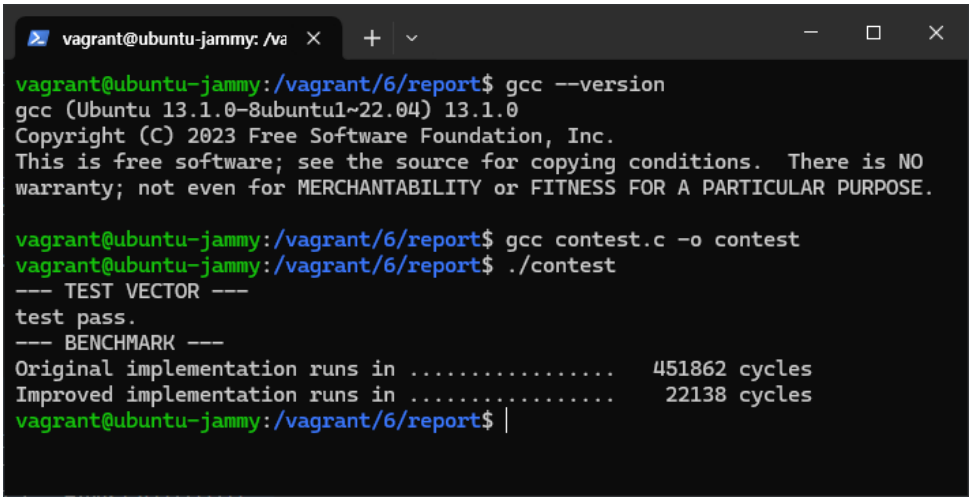

Figure 1: 실험 환경 및 테스트 벡터 확인

3.2 실험 결과

실험은 각 최적화 기법당 100회 실행한 결과의 평균치를 계산하였다. 컴파일, 실행 및 결과 도출은 파이썬 스크립트 (`stat.py`) 를 통해 진행하였다.

	최적화 기법	파일명	CPU Cycle	기존 코드 대비 실행속도
	Baseline	textttbase.c	378,734	100%
1	Precompute	`o1_precompute.c`	424,795	112.2%
2	Inline/Loop Unroll	`o2_inline_unroll.c`	176,434	**46.6%**
3	8bit to 64bit	`o3_8bit_to_64bit.c`	419,726	110.8%
4	gcc Optimization	`o4_gcc_optim.c`	99,795	**26.3%**
	2,4 결합	`mix1_24.c`	24,536	6.48%
	2,3,4 결합	`mix2_234.c`	23,330	6.16%
	전체 최적화	`contest.c`	22,971	**6.07%**

Table 2: 실험 결과

1, 3번 기법은 단일 적용시 비효율적이지만, 다른 기법들과 같이 적용하였을 때 효과를 발휘하였다. 최종적으로는 전체 최적화 기법을 적용한 코드가 제일 효율적이었고, 해당 코드를 최종 제출본으로 결정하였다.