Toolchain for Timing leakage Analysis of NIST Lightweight Cryptography Submissions

Adam B. Hansen, *Morten Eskildsen, Eske Hoy Nielsen*

Nist LWC Workshop, 2020

Toolchain for Timing Leakage Analysis

- The NIST Lightweight Crypto Standardisation call
- Timing Side channel Attacks
- Tools and Pipeline
- Results on Reference Implementations

LWC Call overview

- *“There are several emerging areas […] in which highly-constrained devices are interconnected, […] Because the majority of current cryptographic algorithms were designed for desktop/server environments, many of these algorithms do not fit into constrained devices.“* - NIST
- Standardised Authenticated Encryption algorithms for:
  - Small/power limited boards
  - IoT devices
  - Embedded devices
- Current solutions aren’t good enough

LWC Call overview

- *“The implementations of the AEAD algorithms and the optional hash function algorithms should lend themselves to countermeasures against various side-channel attacks, including timing attacks, simple and differential power analysis (SPA/DPA), and simple and differential electromagnetic analysis (SEMA/DEMA).”* -Nist

Motivation: Timing Attacks

- Variable time instructions
- Code branching on secret data
- Cache timing attacks
  - S-box Table Lookups

Branching on Secret Data

- Different length branches can trivially leak data
- Branches with same number of CPU cycles
  - Variable time instructions
  - Cache hits/misses
- Branch Prediction
  - Can be exploited to leak key [AKS06]

- Don’t branch on secret data

Cache Timing Attacks

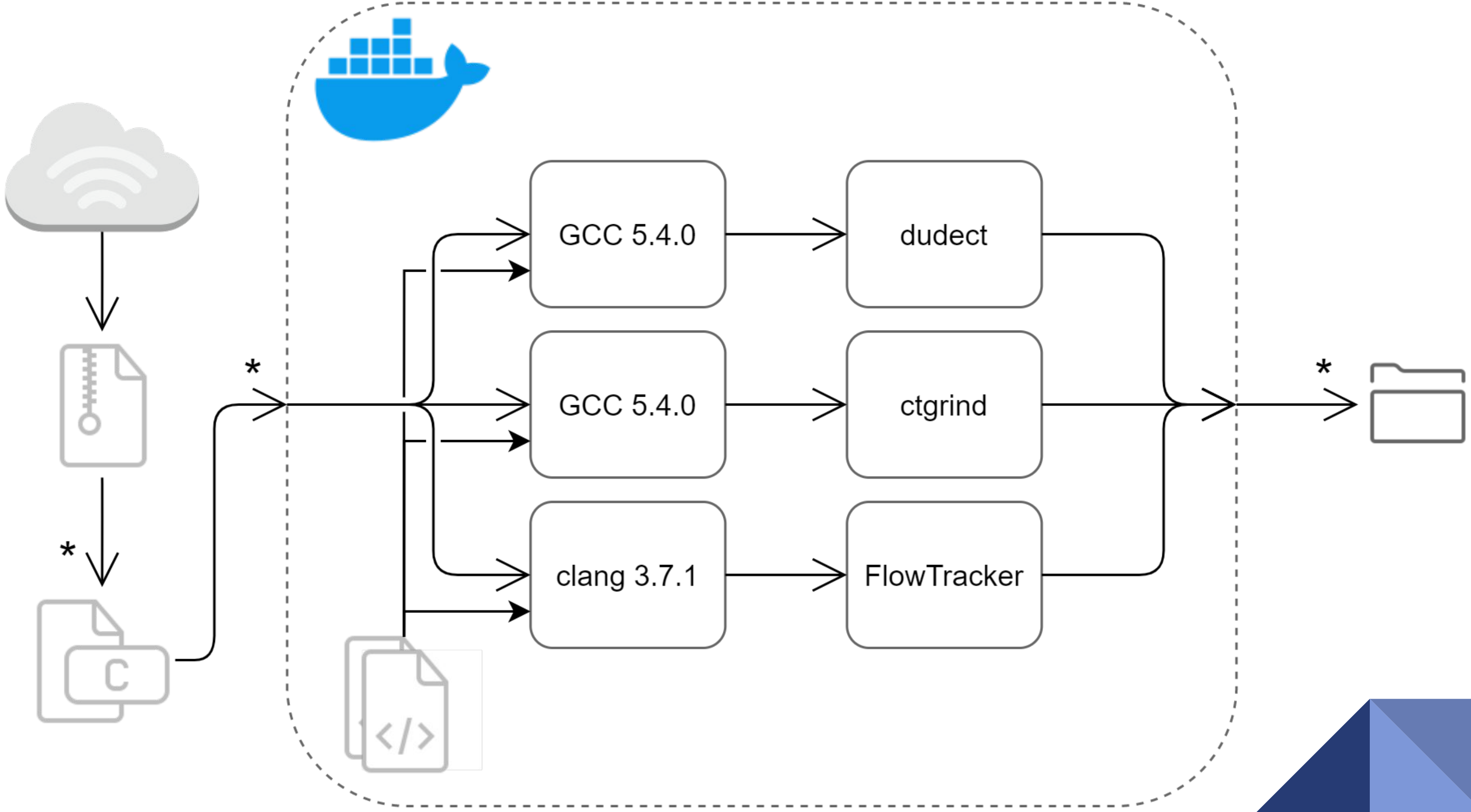
- Leaking information through cache hits/misses
- Cold Boot attacks, Evict + Reload, Prime + Probe...
- S-boxes
  - Can be implemented as in memory lookup tables
  - Attacks on AES[Ber05]
  - Index Keys can leak data[Tez19]
  - Vulnerable even if full S-box fits in cache
  - Potentially Vulnerable even if full S-box fits on one cache line

- Common Problem among reference implementations

Overview of Tools

- Dudect
  - Dynamic analysis/fuzzing
  - Statistical analysis of execution time
- CTGrind
  - Dynamic analysis
  - Monitors branching on secret data
  - Based on Valgrind
- FlowTracker
  - Static analysis
  - LLVM

Our Pipeline



Results

- Reference Implementations
  - As of June '20
- DudeCT flags 8 candidates
- CTGrind flags 14 candidates
  - DryGascon
  - Comet
  - S-box table lookups
- FlowTracker flags 11 candidates
  - Only 6 overlap with CTGrind
  - Of the 5 unique, at least 3 appear to be false positives

Candidate	dudect	ctgrind	FlowTracker	Notes
ACE	○	○	●	
ASCON	○	○	○	
COMET	●	●	○	
DryGASCON	◑	●	●	
Elephant	◑	◑	○	ctgrind finds more than dudect
ESTATE	◑	●	●	
ForkAE	◑	●	●	
GIFT-COFB	○	○	○	
Gimli	○	○	●	NIST format not followed
Grain-128AEAD	○	○	○	NIST format not followed
HYENA	○	●	○	
ISAP	○	○	○	
KNOT	○	○	●	
LOTUS	○	●	●	
mixFeed	●	●	●	
ORANGE	●	●	●	
Oribatida	○	○	○	
PHOTON-Beetle	○	●	○	Also provided bitsliced asm files
Pyjamask	○	○	○	
Romulus	○	●	○	
SAEAES	●	●	○	
Saturnin	○	○	○	
SKINNY	○	●	○	
SPARKLE	○	○	○	
SPIX	○	○	●	
SpoC	○	○	●	
Spook	○	○	○	
Subterranean 2.0	○	○	○	NIST format not followed
SUNDAE-GIFT	○	○	○	
TinyJambu	○	○	○	
WAGE	○	●	○	
Xoodyak	○	○	○	

DryGascon

- Variable time key loading
- 256bit immediately flagged by dudect
- Ctgrind flags key expansion function
- Requires certain conditions on least significant bits of state

Comet

- Implementations using CHAM, Speck and AES
  - Ctgrind flagged AES S-boxes
- All had conditional jump on one bit of the State

```
Summary of running tools on the provied code

Result of running dudect:
Last 3 iterations gave
meas:   11.70 M, max t: +483.49, max tau: 1.41e-01, (5/tau)^2: 1.25e+03. Probably not constant time.
meas:   12.12 M, max t: +497.16, max tau: 1.43e-01, (5/tau)^2: 1.23e+03. Probably not constant time.
meas:   12.39 M, max t: +518.16, max tau: 1.47e-01, (5/tau)^2: 1.15e+03. Definitely not constant time.
Full dudect report can be found in dudect.out in the output directory

Result of running ctgrind:
==81== ERROR SUMMARY: 6000 errors from 4 contexts (suppressed: 0 from 0)
Full ctgrind report can be found in ctgrind.out in the output directory

Result of running flowtracker:
Vulnerable Subgraphs: 0
Vulnerable Subgraphs can be found in flowtracker directory in the output directory
```

S-box Table Lookups

```
const unsigned char sbox[16] = {12,6,9,0,1,10,2,11,3,8,5,13,4,14,7,15};
//...//

void SubCell(unsigned char state[4][4]){
    int i,j;
    for(i = 0; i < 4; i++)
        for(j = 0; j < 4; j++)
            state[i][j] = sbox[state[i][j]];
}
```

Figure 1: Substitution step in the ForkAE implementation, using a 4 bit S-box

S-box Table Lookups

- Attacks are practical
- Example: Mixfeed
  - Indexes into 8 bit S-box with XOR of roundkey and plaintext

S-box lookup issues - mitigations

- Hardware support
  - AES-NI op-codes on modern x86 processors
  - Misses the point of this contest
- Bitslicing
  - Rewriting code/table lookups as binary operations
  - Can increase speed and guarantees constant time execution
- Implementing Bitslicing
  - AES
  - SKINNY
  - Gift

Results: tools + pipeline

- DudeCT
  - Fuzzing + Statistical test
  - “No” false positives
  - Black box
- CTGrind
  - Dynamic memory analysis
  - Very precise reporting
- FlowTracker
  - Full code coverage -> Potentially not as relevant in symmetric crypto?
  - Many false positives?
  - Negatively impacted by shared libraries and pointer arithmetic

FlowTracker

- Static analysis vs Dynamic Analysis
- False positives

```
const unsigned char rate_bytes256[8] = {8,9,10,11,24,25,26,27};
(...)
for ( i = 0; i < 8; i++ )
    state[rate_bytes256[i]]^=k[i];
```

Figure 2: One of the SPIX lines flagged by FlowTracker

Our Pipeline

- Aimed at supporting development/local testing
- Compiled all tools in a docker image targeting competition API
  - Wrapper script takes input folder and output folder, optional settings file
- Provide prebuilt image
  - blatchley/ct-analysis:latest
- Source code to build locally, Readme
  - https://github.com/blatchley/Timing-Analysis-Pipeline

In Context of Competition

- “These are just reference implementations”
  - Some candidates still not submitting constant time versions
  - Reference implementations are being benchmarked and compared
  - Good demonstration of types of leakages our tooling can detect
- AES vs Skinny/Gift/others
  - Table lookup AES is fast
  - Was selected when table lookups were not seen as variable time
  - Some see the point of this contest to be replacing AES for lightweight devices
- We expect new focus on side channel security for round 3
  - Provide our Pipeline to help with development
- SuperCop/TimeCop

Side Channel Analysis of NIST Lightweight Cryptography Submissions

Adam B. Hansen, *Morten Eskildsen, Eske Hoy Nielsen*

Thanks to Associate Professor Diego F. Aranha

DryGascon

```
for(unsigned int i=0;i<DRYSPONGE_CAPACITYSIZE;i++){
    ctx->c[i] = key[i%DRYSPONGE_KEYSIZE];
}

// ... SNIPPET ...

DRYSPONGE_CoreRound(ctx,0);

unsigned int modified=1;
while(modified){
    modified=0;
    for(unsigned int i=0;i<DRYSPONGE_XSIZE32-1;i++){
        for(unsigned int j=i+1;j<DRYSPONGE_XSIZE32;j++){
            uint32_t ci,cj;
            DRYSPONGE_load32(&ci,ctx->c+i*sizeof(uint32_t));
            DRYSPONGE_load32(&cj,ctx->c+j*sizeof(uint32_t));
            if(ci==cj){
                DRYSPONGE_CoreRound(ctx,0);
                modified=1;
                break;
            }
        }
        if(modified) break;
    }
}
memcpy(ctx->x,ctx->c,DRYSPONGE_XSIZE);
memcpy(ctx->c,key,DRYSPONGE_XSIZE);
```

Comet Patch

```
if(Z_[p-1] & 0x80){                    /*10000000*/
        Z[0] ^= 0x1B;      /*00011011*/
}
```

Figure 4: Variable time code in COMET found by ctgrind

```
u8 a = Z[0]^0x1B;
u8 b = Z[0];
u8 bit = Z_[p-1] & 0x80;
    u8 mask = (bit | -bit) >> (sizeof(u8) * CHAR_BIT - 1);
    u8 ret = mask & (b^a);
Z[0] = ret ^ b;
```

Figure 5: Constant time version of figure 4