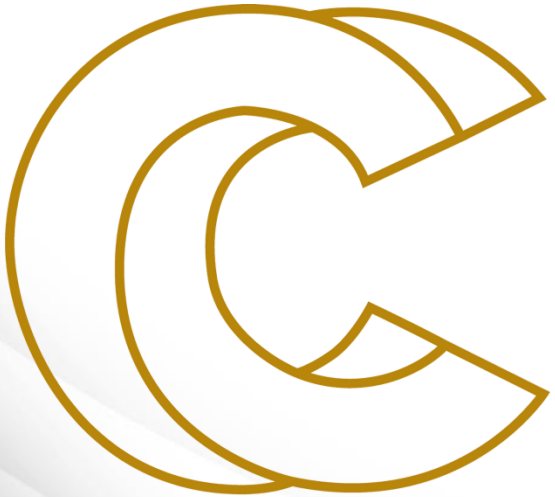

EURO 4SEE

GPU Assisted Brute Force Cryptanalysis of GPRS, GSM, RFID, and TETRA

Cihangir Tezcan, PhD
Graduate School of Informatics, METU, Ankara

ncc@ulakbim.gov.tr

Lesson 8: CUDA Optimization of SPECK

CUDA Optimization of SPECK (SDK Version)

- We observed that compiling our SPECK optimizations with compute capability 5.2 provides around $2^{0.06}$ better performance compared to a code compiled for compute capability 8.9.
- For example, trying 2^{43} keys for SPECK-96-26 using an RTX 4090 takes 80.63 seconds and 77.80 seconds when compiled for compute capabilities 8.9 and 5.2, respectively.
- Current CUDA SDKs do not support compute capability less than 5.0 but using older CUDA SDKs allow us to use deprecated compute capabilities.
- Thus, we used CUDA SDK 11.1 and compiled our implementations for compute capability 3.5.
- However, using compute capability 3.5 did not provide any observable speed up compared to 5.2.

Lesson 8: CUDA Optimization of SPECK

CUDA Optimization of SPECK

- Although SPECK-64-22 and SPECK-72-22 have smaller number of rounds compared to SPECK-96-26, we obtained better speeds for SPECK-96-26.
- We observed that this is because the key and the data are stored in 32-bit registers in SPECK-96-26 due to its block size and on CUDA devices the rotation operation on 32-bit values is faster than 16 or 24-bit values.
- Thus, 64 and 72-bit versions might be broken faster in the future if new GPU instructions provide speed-ups for rotations on values smaller than 32 bits.
- Note that using 32-bit unsigned integers for storing 16 or 24-bit values requires additional AND operations after the rotation with 0xFFFF or 0xFFFFFF, respectively.
- In software implementations, it is a common practice to use a *for* loop to perform ***r*** rounds of encryption inside the loop. Using the ***#pragma unroll*** macro unrolls these loops and in the case of SPECK, this provided marginal speed-up in our implementations.
- Our optimizations use small numbers of registers so that we can call the GPU kernels with 1024 threads as follows to try $2^{20+trial}$ keys:
  - speck64_exhaustive <<< 1024, 1024 >>> (ct_d, pt_d, K_d, trial);
  - speck72_exhaustive <<< 1024, 1024 >>> (ct_d, pt_d, K_d, trial);
  - speck96_exhaustive <<< 1024, 1024 >>> (ct_d, pt_d, K_d, trial);
  - speck128_exhaustive <<< 1024, 1024 >>> (ct_d, pt_d, K_d, trial);

Lesson 8: CUDA Optimization of SPECK

GPU	SPECK-64-22	SPECK-72-22	SPECK-96-26	SPECK-128-32
MX 250	$2^{30.56}$ keys/s	$2^{30.72}$ keys/s	$2^{31.43}$ keys/s	$2^{29.93}$ keys/s
GTX 860M	$2^{30.53}$ keys/s	$2^{30.72}$ keys/s	$2^{31.30}$ keys/s	$2^{29.86}$ keys/s
GTX 970	$2^{32.12}$ keys/s	$2^{32.43}$ keys/s	$2^{32.99}$ keys/s	$2^{31.38}$ keys/s
RTX 2070 Super	$2^{33.99}$ keys/s	$2^{34.27}$ keys/s	$2^{34.49}$ keys/s	$2^{32.97}$ keys/s
RTX 4090	$2^{36.20}$ keys/s	$2^{36.47}$ keys/s	$2^{36.72}$ keys/s	$2^{35.30}$ keys/s

Next lecture

CUDA Optimization of TEA3

EURO 4SEE

Thanks!

Co-funded by
the European Union

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101191697. The JU receives support from the Digital Europe Programme and Germany, Türkiye, Republic of North Macedonia, Montenegro, Serbia, Bosnia and Herzegovina.