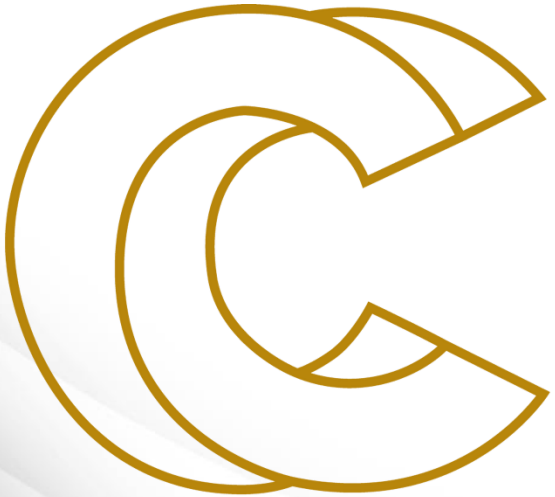

EURO 4SEE

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 9: CUDA Optimization of TEA3

CUDA Optimization of TEA3

- TEA3 is a keystream generator and it is not suitable for a table-based implementation due to its design.
- TEA3 can be seen as two parts in which one part applies the S-box on the key register and the other part applies F31, F32, and R3 functions, which consist of bit-level operations, on the state register.
- Initially we optimized TEA3 for GPUs using the straightforward implementation.
- However, this approach did not provide good results due to the bit level operations at the state registers.
- Our best optimization reached $2^{27.39}$ key trials per second on an RTX 4090 GPU.

Lesson 9: CUDA Optimization of TEA3

CUDA Optimization of TEA3

- TEA3 is also not suitable for a bitsliced implementation due to the 8-bit S-box operation on the key register.
- Thus, we combined the straightforward and bitsliced implementation techniques where we implemented the key register update part as a straightforward implementation and the state register update part as a bitsliced implementation.
- For the bitsliced part, we tried using 8-bit, 16-bit, and 32-bit registers.
- In other words, we obtained single instruction multiple data (SIMD)-level parallelism at variable-level by trying 8, 16, and 32 keys in a single encryption, respectively.
- Note that although increasing the word size of registers increases the number of parallel encryptions, it also increases the number of registers used in each thread.
- Although our straightforward implementation requires less than 65 registers, using 8, 16, and 32-bit values in the bitsliced part increases the required number of registers to 127, 166, and 171, respectively.
- Note that these numbers change depending on the compiled compute capability and used SDK so we compiled our code using many SDKs and compute capabilities but none of them reduced these numbers to 64 or to 128 in the case of 16 and 32-bit word sizes.
- Thus, our kernel blocks have to decrease to 512 threads when the word size is 8 bits and to 256 threads when the word size is 16 or 32 bits.

Lesson 9: CUDA Optimization of TEA3

CUDA Optimization of TEA3

- In each thread, we use 32 different key registers and store the S-box output results at *uint32_t bSboxOut[32]*.
- Since the S-box has a size of 8 bits, the result can be seen as a 32x8 matrix.
- We need the transpose of this bit matrix to move from our straightforward implementation to bitslice implementation.
- Note that limiting the maximum used register count to 64 (or to 128) as we did in our KASUMI implementation causes a performance loss in this case because the kernel really needs to keep more than 64 (or 128) registers.
- For instance, bitslicing the 80-bit state register already requires 80 registers per thread, exceeding the 64 limit. If we force CUDA SDK to use only 64 registers by using the command ***-maxrregcount=64*** as we did when implementing KASUMI, the required extra registers spill and are kept in the global memory.
- Reading and writing these values to and from the global memory or cache provides delays that significantly slow down our implementation.
- Although decreasing the thread count in a block to 512 or 256 decreases the GPU occupancy and therefore the performance, the number of parallel executions in the bitsliced implementation still provide better results.
- We obtained the best results when we used 32-bit registers and achieved $2^{34.71}$ key trials per second on an RTX 4090. This is around 160 times faster than our straightforward implementation.

CUDA Optimization of TEA3

GPU	Key Search
MX 250	$2^{28.95}$ keys/s
GTX 860M	$2^{29.23}$ keys/s
GTX 970	$2^{30.53}$ keys/s
RTX 2070 Super	$2^{32.54}$ keys/s
RTX 4090	$2^{34.71}$ keys/s

Next lecture

Summary of GPU Performance of Ciphers

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.