Subverting the security base of GSM

Karsten Nohl
Sascha Krißler

@ HAR 2009

Phone with end-to-end encryption-- soon needed?

GSM encryption has been broken over and over again

- Academic breaks of A5/1 cipher: EC1997, FSE2000, Crypto 2003, SAC2005, …
- A5/1 crackers widespread among intelligence agencies
- Cracking tables computed in 2008 but never released

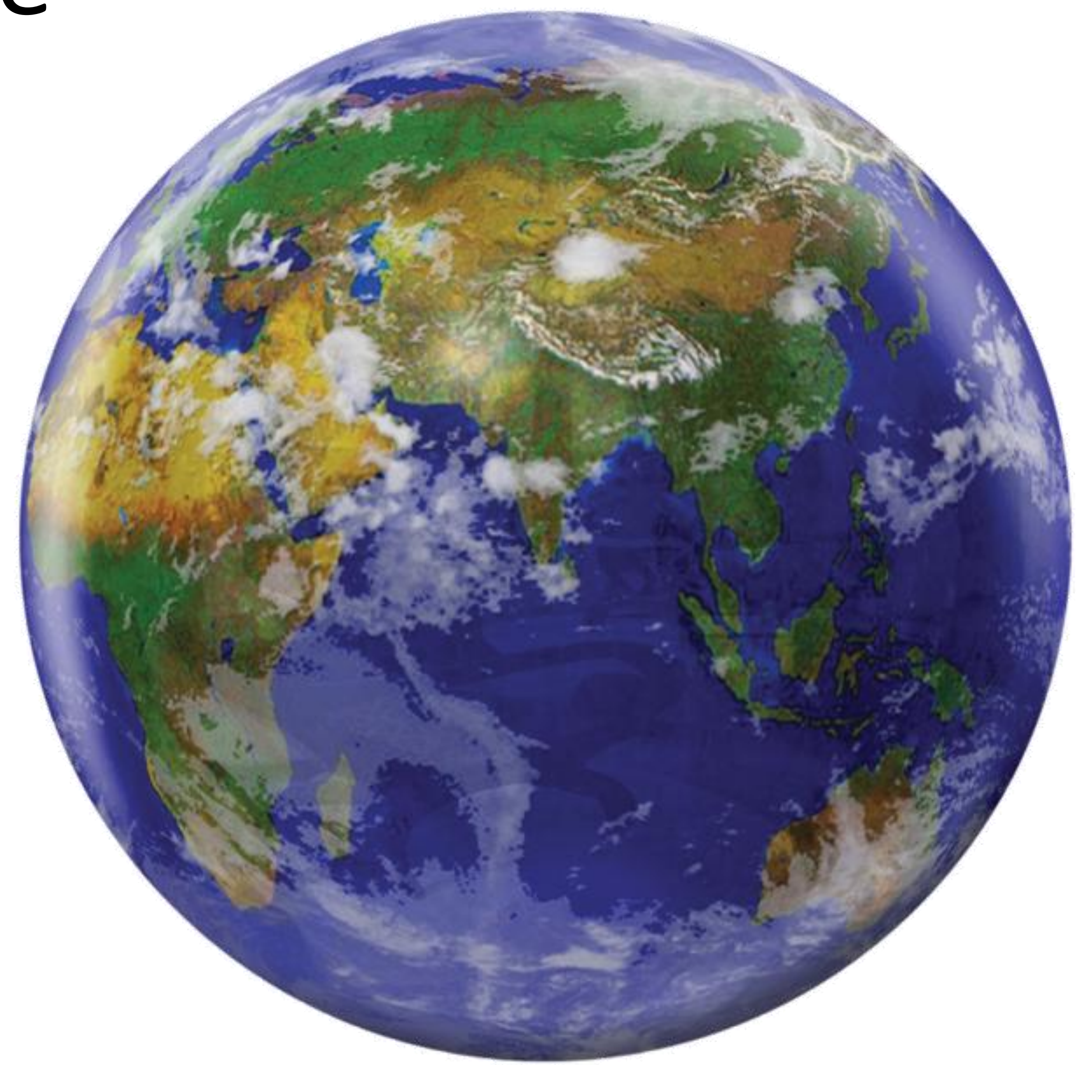
- After 15 years, still no public A5/1 exploit !!
- We'll change this over the next months

GSM is global, omnipresent and insecure

80% of mobile phone market

200+ countries

3 billion users!



GSM security introduced in 1987 …

… then disclosed and shown insecure in 1994

GSM must not be used for security systems, especially not for new ones

Recent adoptions of GSM despite weak security:

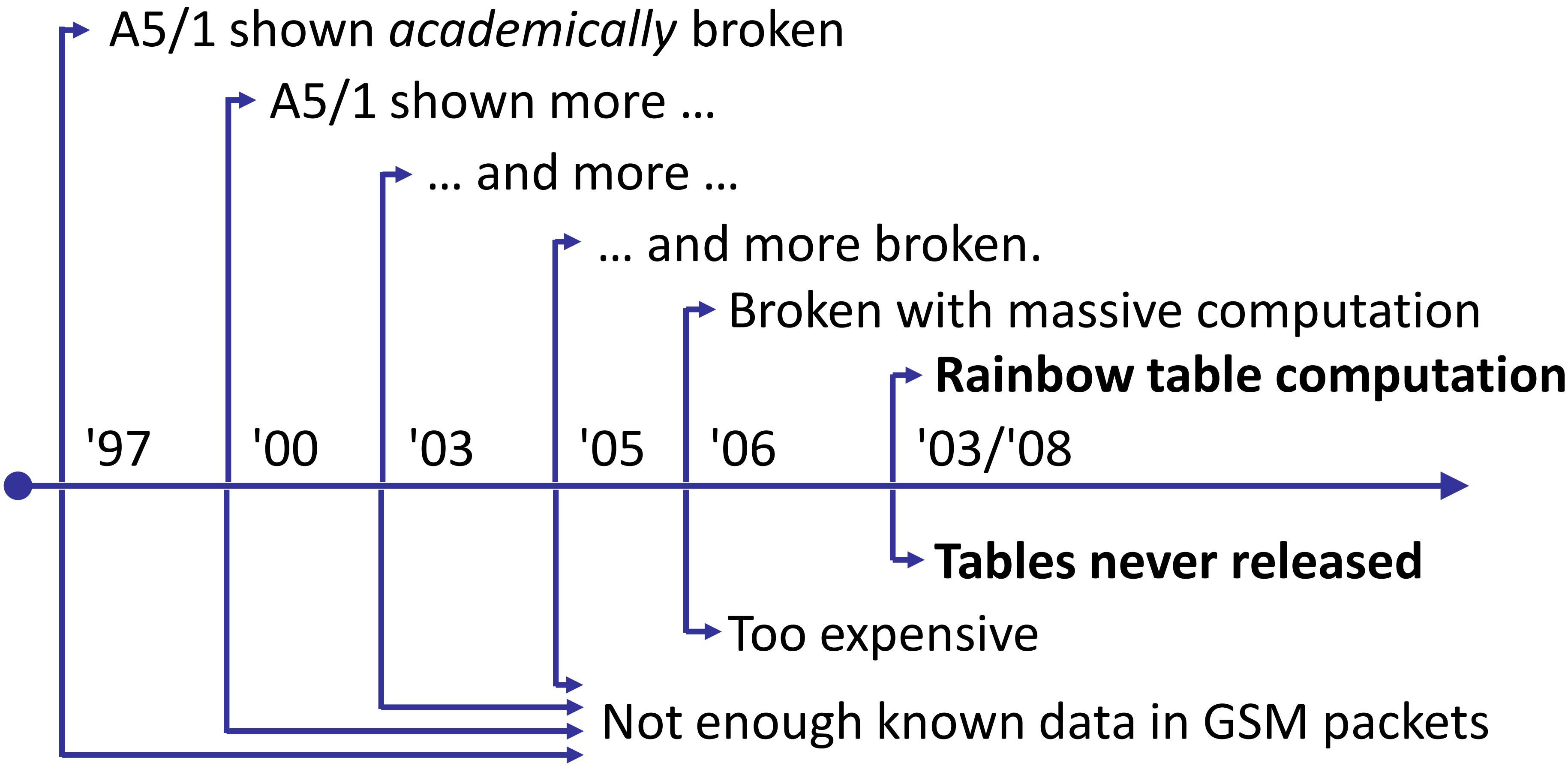
- Home banking
- Payment
- Authentication


Google's SMS Payment Patent

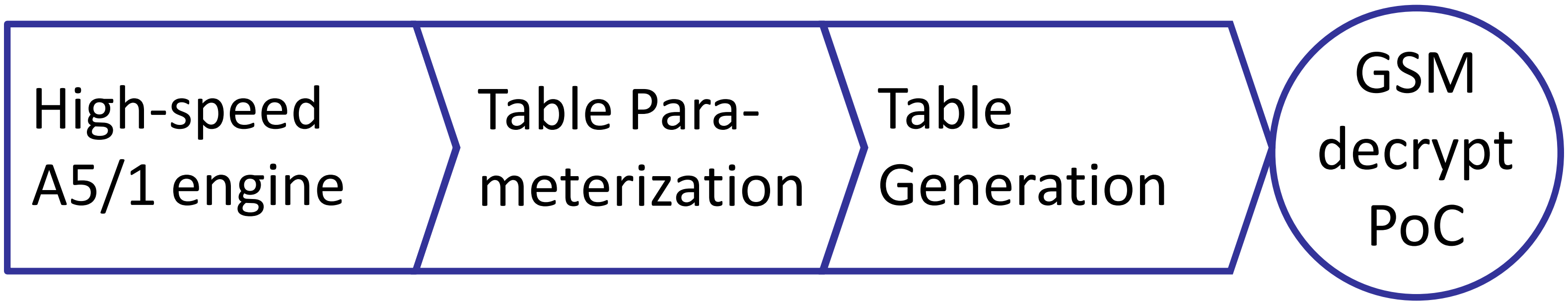
Google in February 2006 filed a patent with the

GSM apparently seen as *secure enough* for payment & access. Falsely so!

We need a public GSM decrypt PoC



Groundwork for table generation is complete and open sourced



Status ✓ ✓ Your help* needed!

Source and doc available:
reflextor.com/trac/a51

* CUDA graphic cards or Xilinx Virtex FPGAs needed

A5/1 is vulnerable to generic pre-computation attacks

- For ciphers with small keys, code books allow decryption
- Code book provides a mapping from known output to secret state

Secret state	Output
A52F8C02	52E91001
62B9320A	52E91002
C309ED0A	52E91003

- An A5/1 code book is 128 Petabyte and takes 100,000+ years to be computed on a PC

This talk revisits techniques for computing the code book faster and for storing it compressed

Key to code book generation is a fast A5/1 engine

Time on single threaded CPU:
100,000+ years

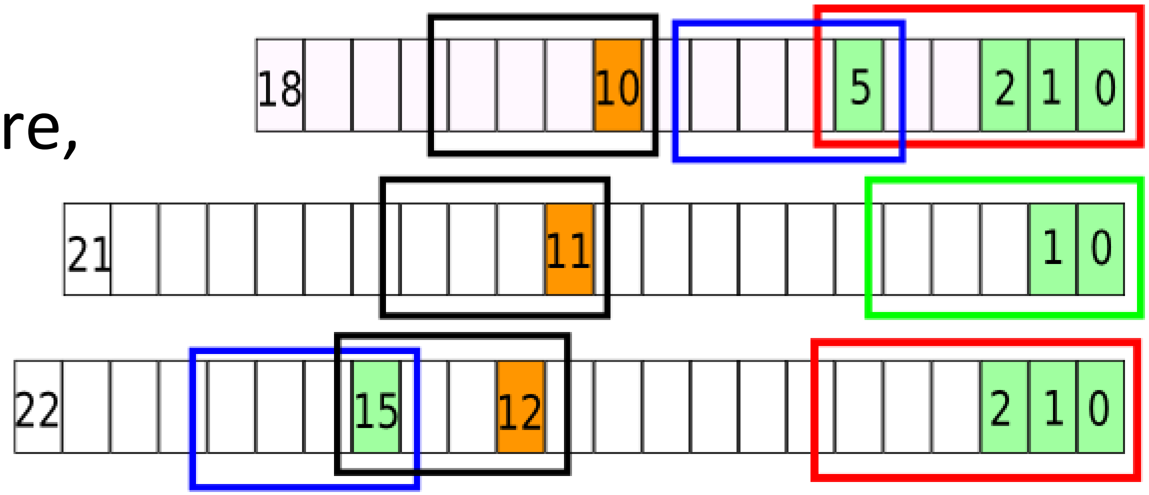
Parallelization
- CUDA: hundreds of threads
- FPGA: thousands of engines

Algorithmic tweaks
- CUDA: compute 4 bits at once
- FPGA: minimize critical path

3 months on **80 CUDA nodes**

Algorithmic tweaks accelerate CUDA A5/1 engine significantly

- Shift registers are expensive in software, while memory is cheap
- Only a few state bits determine round function
- Trade table lookups for shifts; optimal for CUDA: 4 shifts at once



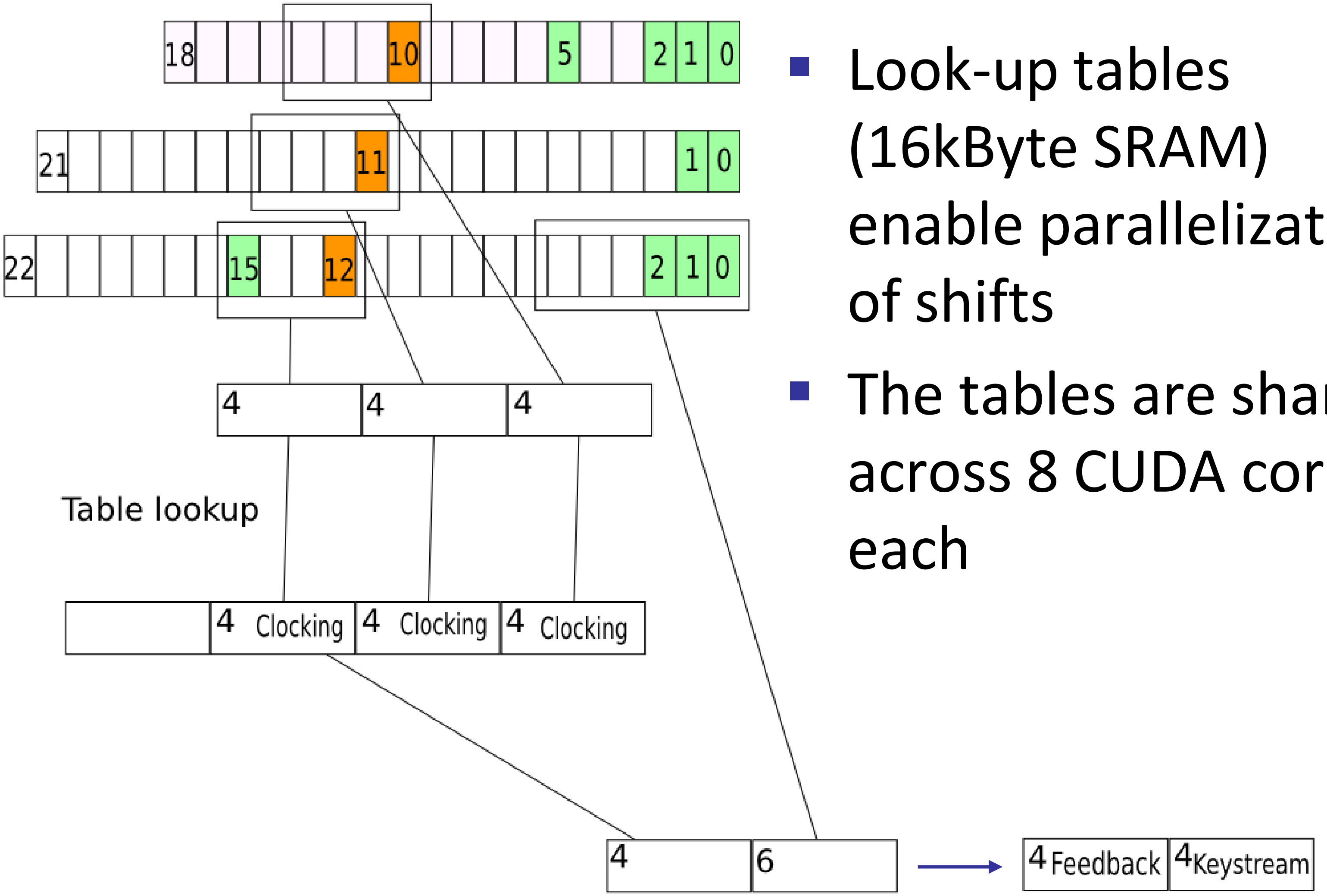
Clocking Table: 4096 x 16 bit

Table 1: 1024 x 8 bit

Table 2: 512 x 8 bit

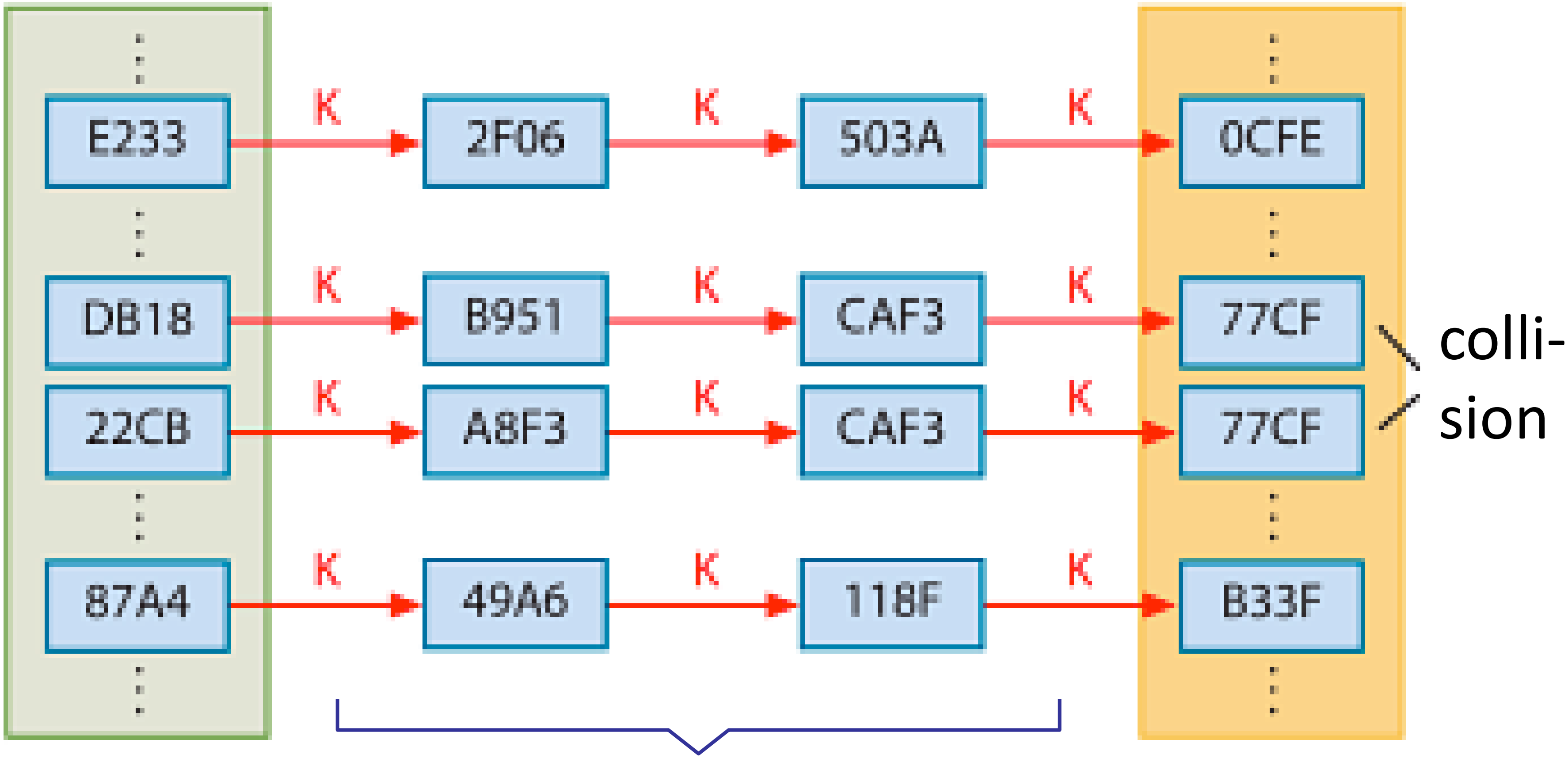
Table 3: 256 x 8 bit

Balancing memory lookups and computation maximizes throughput



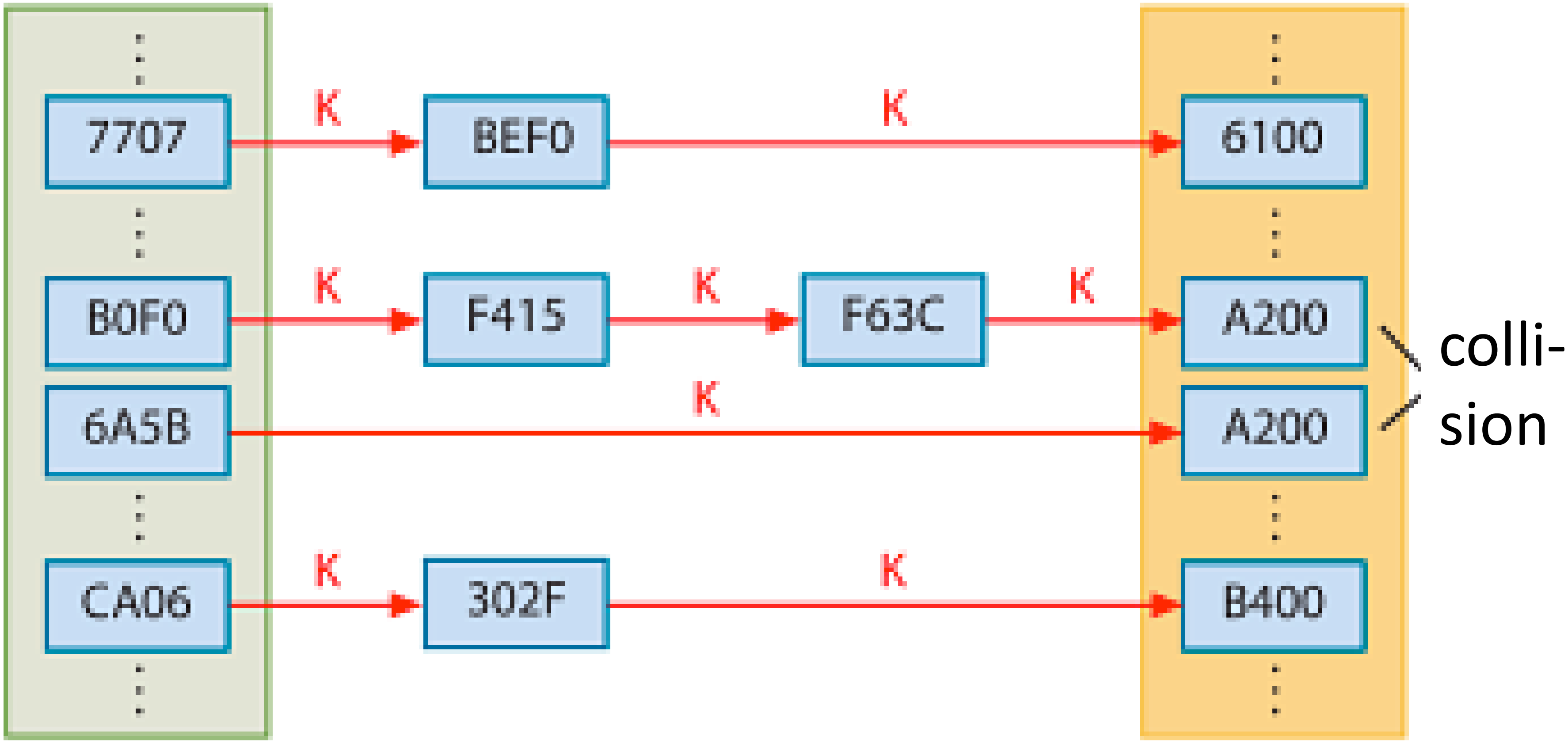
- Look-up tables (16kByte SRAM) enable parallelization of shifts
- The tables are shared across 8 CUDA cores each

Pre-computation tables store the code book condensed



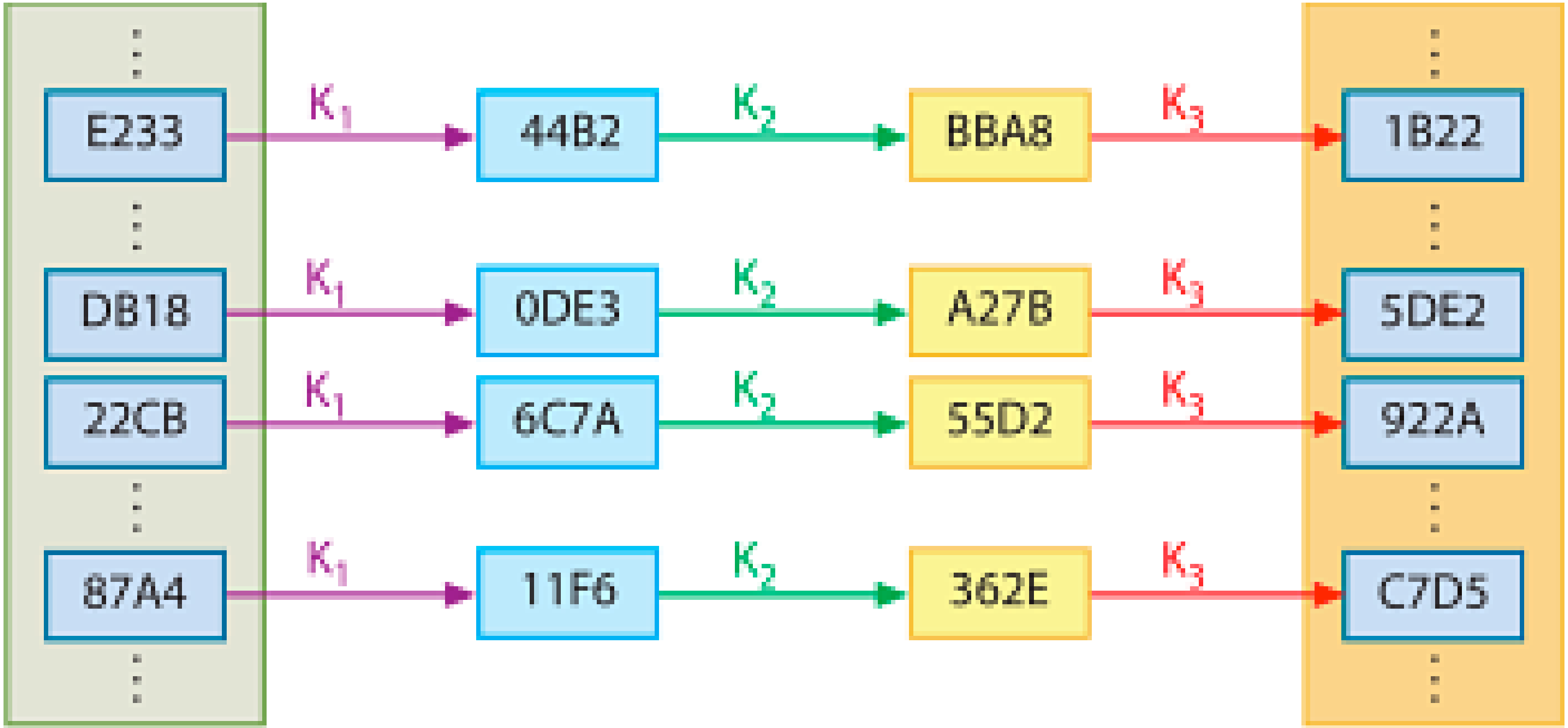
Longer chains := less storage := longer attack time

Distinguished point tables save hard disk lookups



Hard disk access only needed at distinguished points

Rainbow tables mitigate collisions

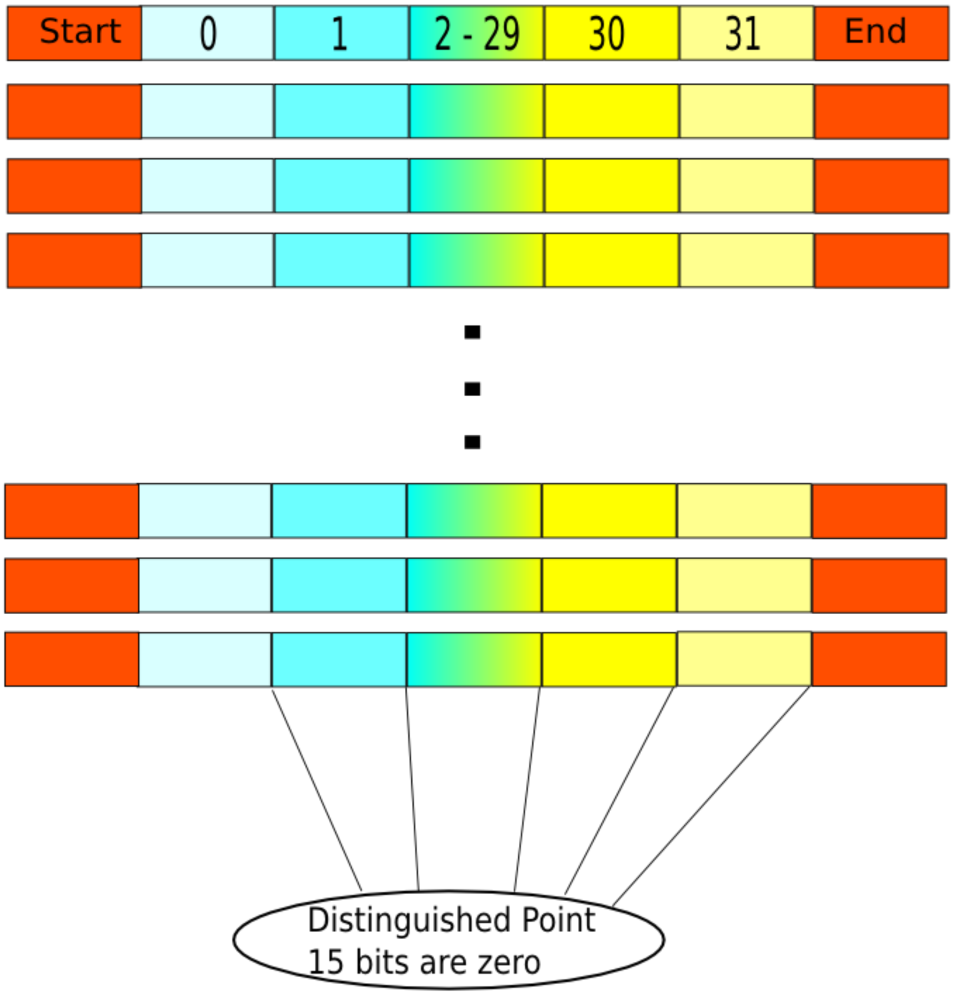


Rainbow tables have no mergers, but an exponentially higher attack time

The combination of both table optimizations is optimal

The most resource efficient table for A5/1 is:

- 32 DP segments of length 2^{15}
- Merged into one rainbow
- 725 such tables with height $2^{28.5}$ needed



Tables must be computed and stored distributed

- For efficiency, tables distributed over many nodes are preferred
- More importantly, no single point of failure should exist on the critical path to the GSM decode PoC

Suggested table generation process:

1. Download tools today
2. Generate table*
3. Publish table on Bittorrent*

Coordinate through mailing list if necessary (use TOR!)

*use random ID and advance parameter; publish as A51_<ID>_<parameter>.table

A5/1 cracking is just the first step …

- Pre-computation framework build to be generic
  – Any cipher (key size up to 64 bits)
  – Various backends: CPU, CUDA, FPGA
- Open source

- Please get involved
  – Compute tables and provide feedback
  – Extend the table generator to your projects

Questions?

Slides, source, documentation	reflextor.com/trac/a51
Mailing list	tinyurl.com/a51list
c’t article	tinyurl.com/ct-rainbows

Karsten Nohl <nohl@virginia.edu>

Many thanks to Sascha Krißler and Sébastien Bourdeauducq!