K-Hunt++: Improved Dynamic Cryptographic Key Extraction

Thomas Faingnaert, Willem Van Iseghem, Bjorn De Sutter

CheckMATE 2024

Salt Lake City, 18 October 2024

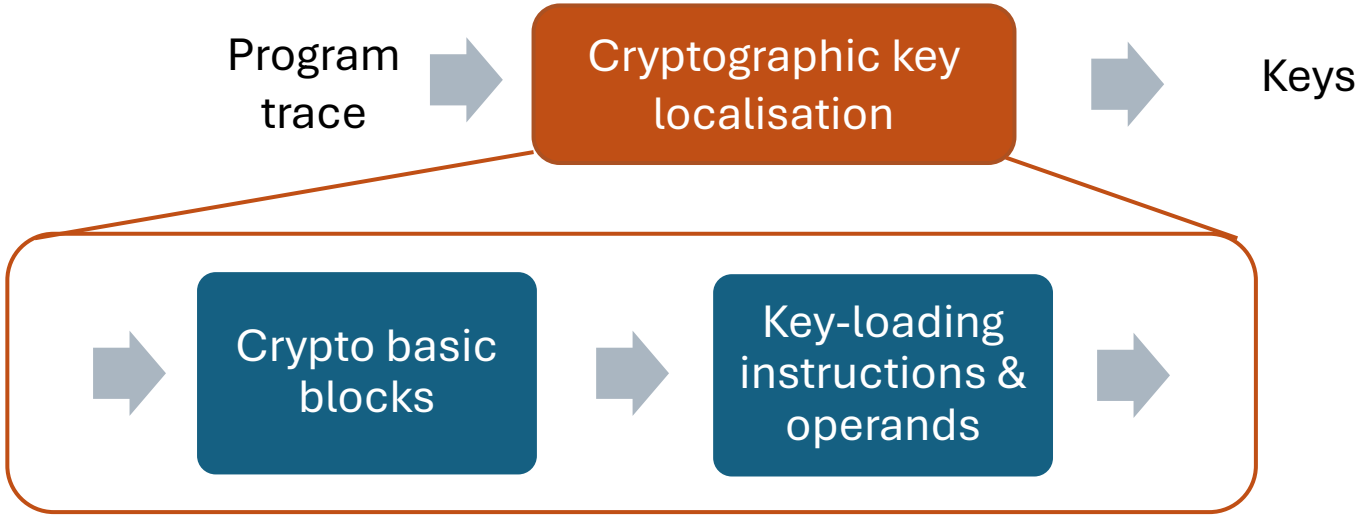
GHENT UNIVERSITY

State of the art in crypto localisation: K-Hunt

Goal: identify insecure cryptographic keys

State of the art in crypto localisation: K-Hunt

Goal: identify insecure cryptographic keys

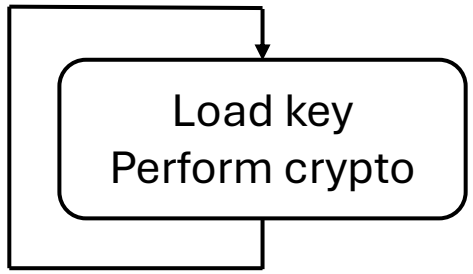


Problems:

1. K-Hunt is not (fully) available
2. K-Hunt has problems with certain features and block cipher modes (e.g. in 7-Zip and GPG)

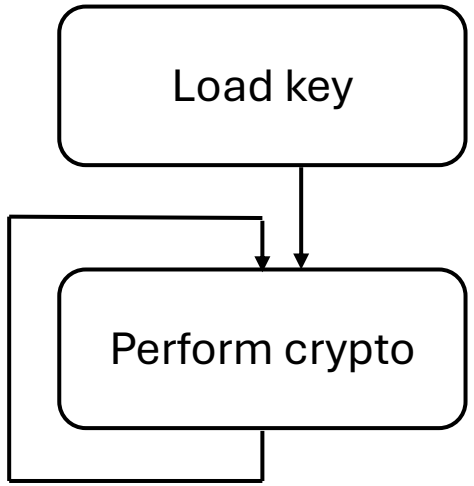
K-Hunt++: An extension of K-Hunt's pipeline

Program 1:



✓ K-Hunt

Program 2: (e.g. GPG)

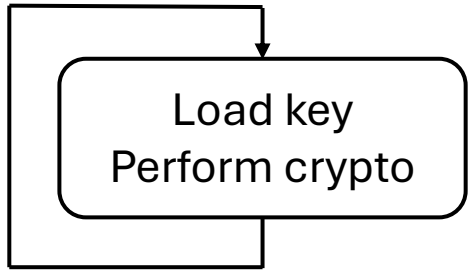


❌ K-Hunt

K-Hunt++: An extension of K-Hunt's pipeline

Extend basic block set using data dependencies

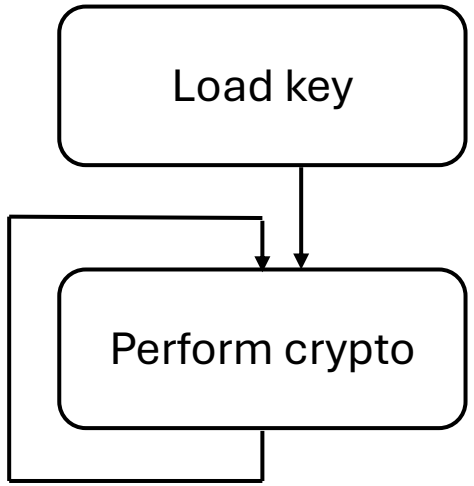
Program 1:



✓ K-Hunt

✓ K-Hunt++

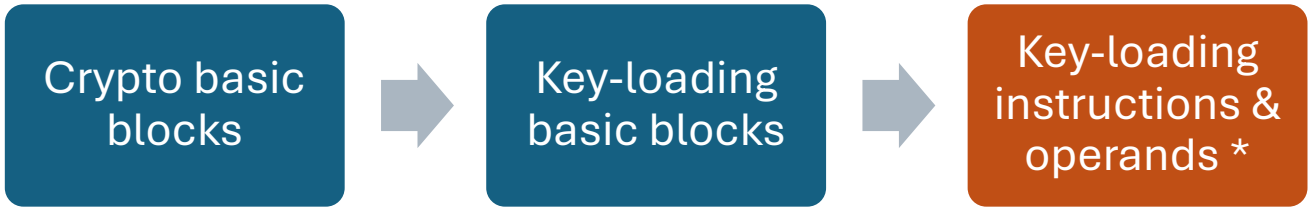
Program 2: (e.g. GPG)



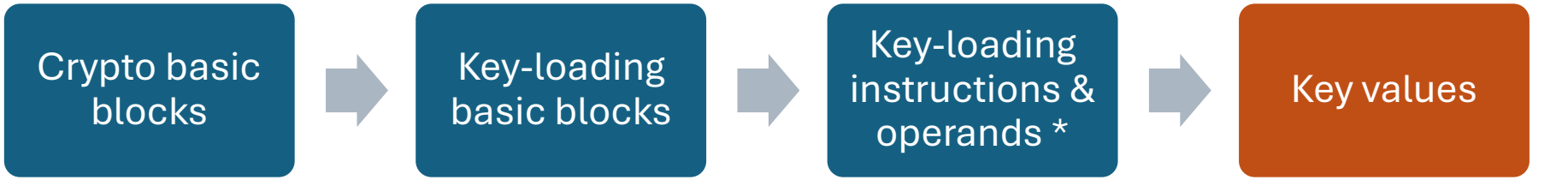
❌ K-Hunt

✓ K-Hunt++

K-Hunt++: An extension of K-Hunt's pipeline

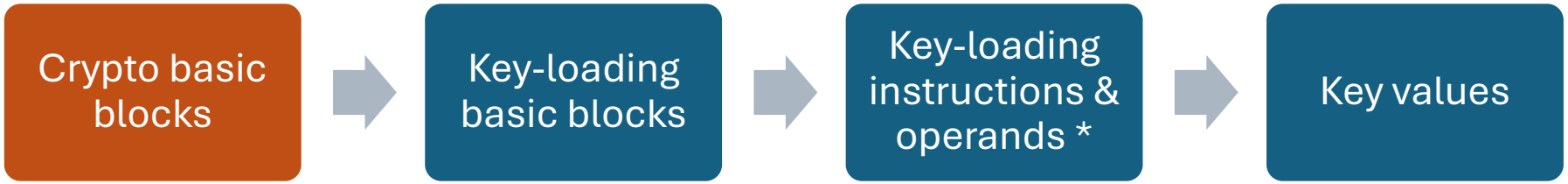


K-Hunt++: An extension of K-Hunt's pipeline



Use debugger to print values of key

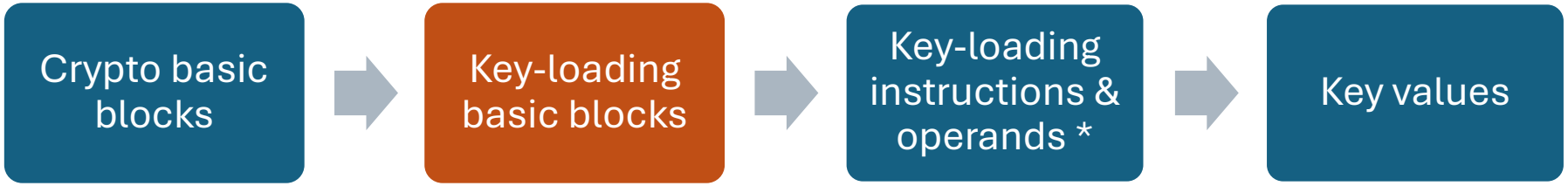
Phase 1: Localise cryptographic basic blocks



Sort basic blocks using information from traces:

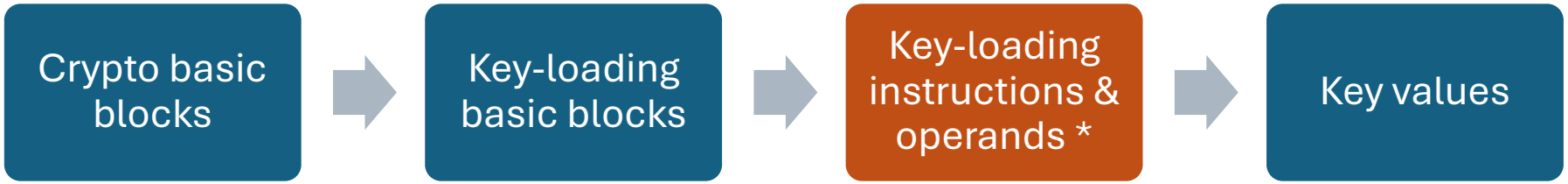
a) *Instruction mix*: arithmetic, bitwise, …
b) *Execution count linear scaling*
c) *Functionality coverage*
d) *Randomness of produced/consumed data*

Phase 2: Key-loading basic blocks



Extend the set of basic blocks from phase 1 using data dependencies

Phase 3: Key-loading instructions & operands

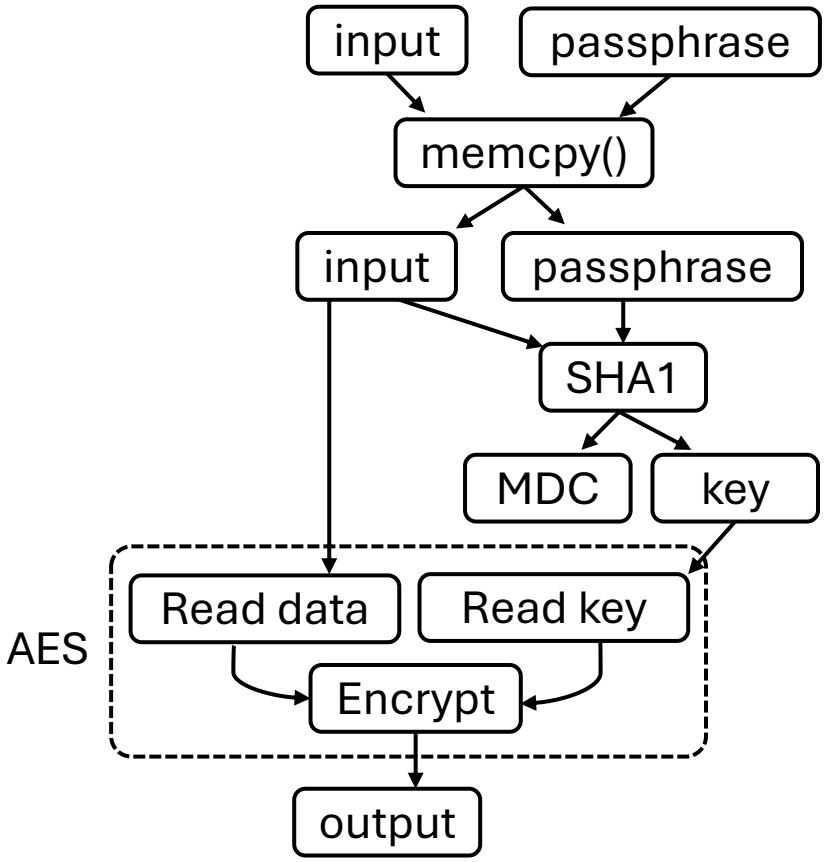


Differentiate key-loading and data-loading instructions using:

e) *Data source*: KDF/RNG vs. file/network (taint analysis)
f) *Buffer size*: small, constant-size buffer

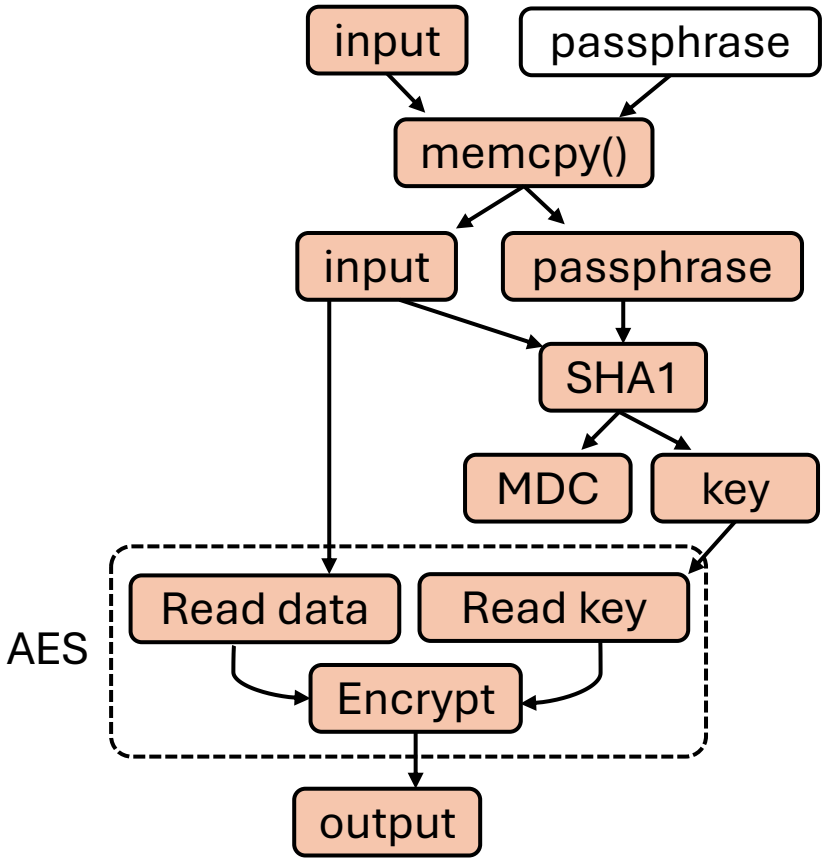
Improved Data Source heuristic in K-Hunt++

Example: GPG Data Dependency Graph

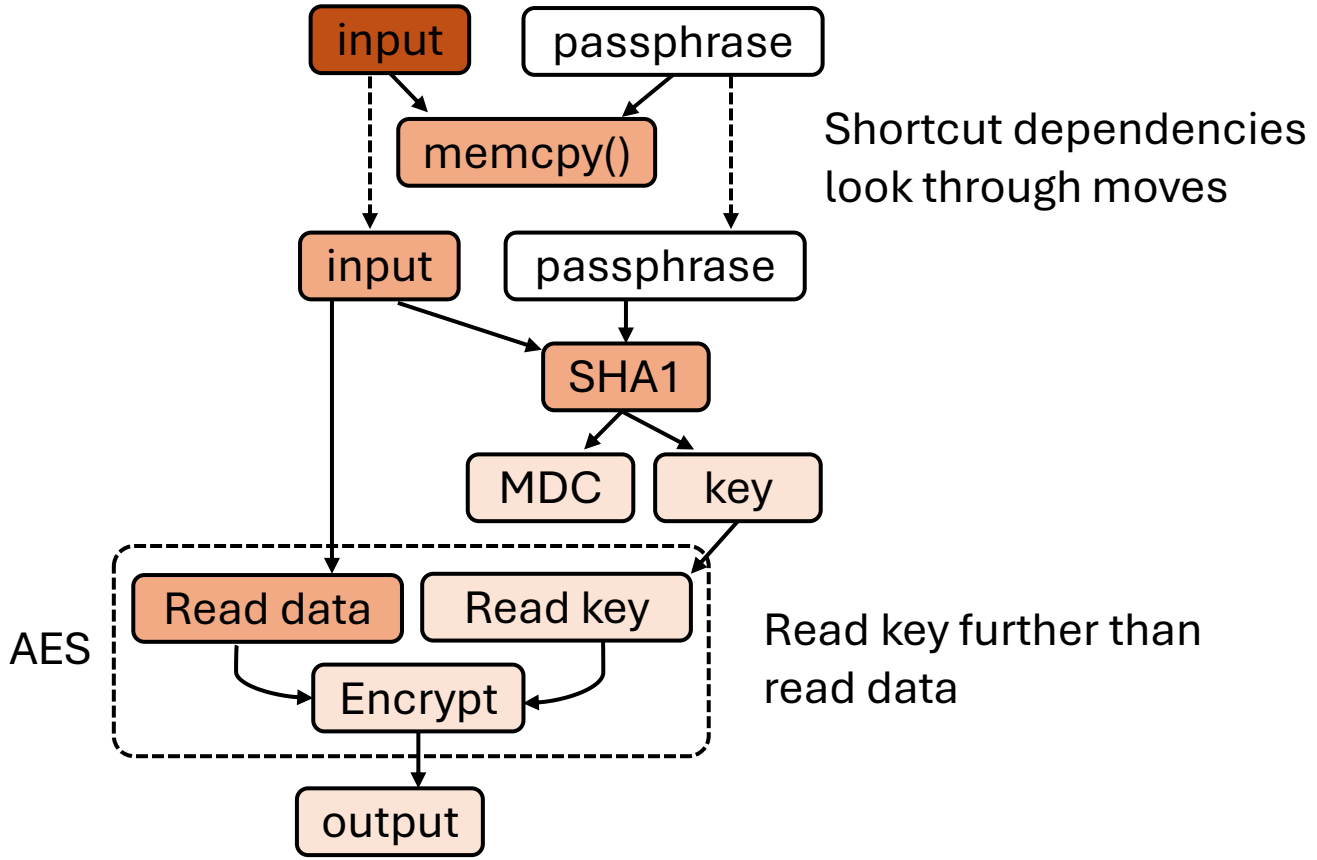


Improved Data Source heuristic in K-Hunt++

Example: GPG Data Dependency Graph



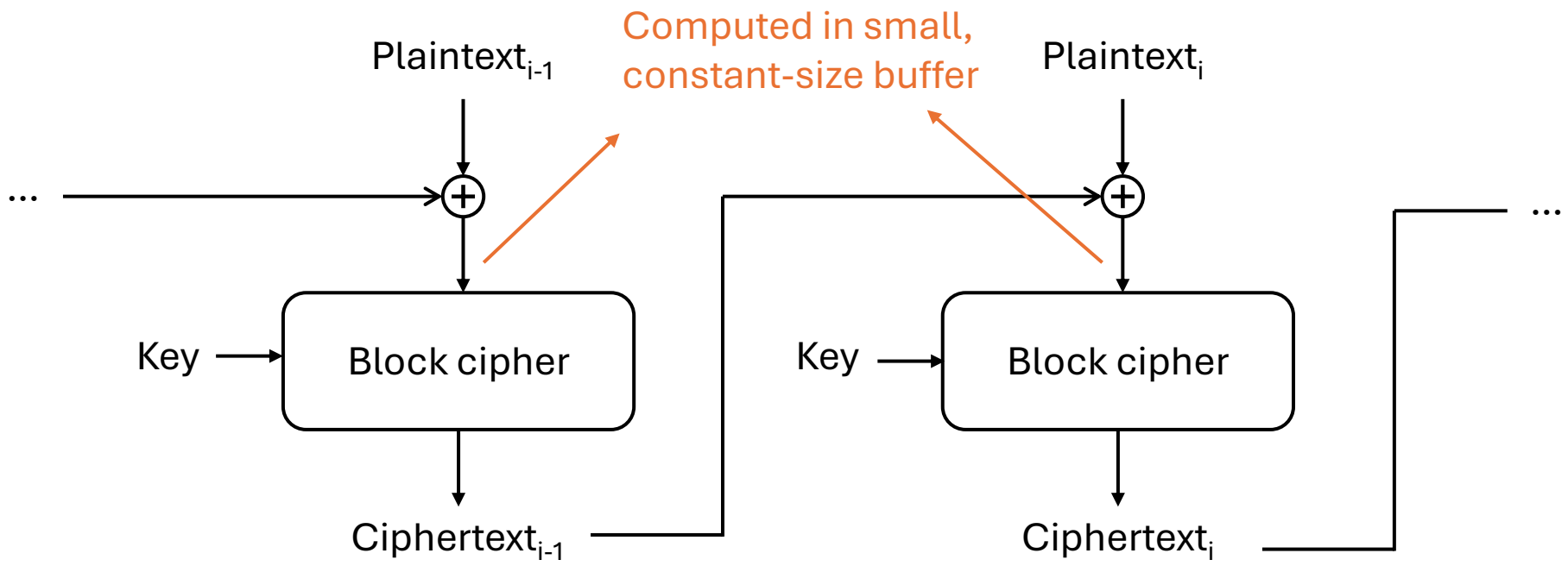
Using K-Hunt's function-level taint analysis



Using K-Hunt++'s distance in the shortcut data dependency graph

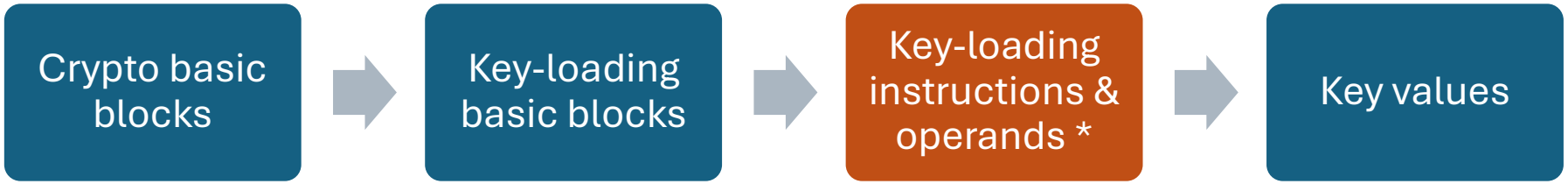
Buffer size heuristic can be broken easily

Example: Cipher Block Chaining mode in 7-Zip



Phase 3: K-Hunt++'s extra heuristics

K-Hunt++ has four additional heuristics in phase 3



Differentiate key-loading and data-loading instructions using:

e) *Data source*: KDF/RNG vs. file/network (taint analysis)
f) *Buffer size*: small, constant-size buffer
g) *Constant keys*: over different runs of the program
h) *Quasi constant keys*: in a single run of the program
i) *Likely key values*: ignore addresses, known constants, …
j) *Instruction types*: ignore e.g. control flow

Evaluation

Ablation study of Phase 1 and Phase 3

- For all configurations: no false negatives
- False positives increase as fewer heuristics are used
- Easy to filter out false positives: O(10) false positives

Performance

- 47 minutes total for 7-Zip, 18 minutes for GPG
- Run time dominated by tracing → similar performance to K-Hunt?

Robustness to obfuscation

K-Hunt++ is an extension of K-Hunt with improved robustness

- Source available
- Additionally extend BB set with data dependencies
  Deals with GPG's spread out loading and use of the key
- Improved "Data source" heuristic in phase 3 using distances in shortcut DDG
  Deals with GPG's memcpy and modification codes
- More heuristics in phase 3 as fallback
  Deals with 7-Zip's CBC

K-Hunt++: Improved Dynamic Cryptographic Key Extraction

Thomas Faingnaert, Willem Van Iseghem, Bjorn De Sutter

CheckMATE 2024

Salt Lake City, 18 October 2024

GHENT UNIVERSITY