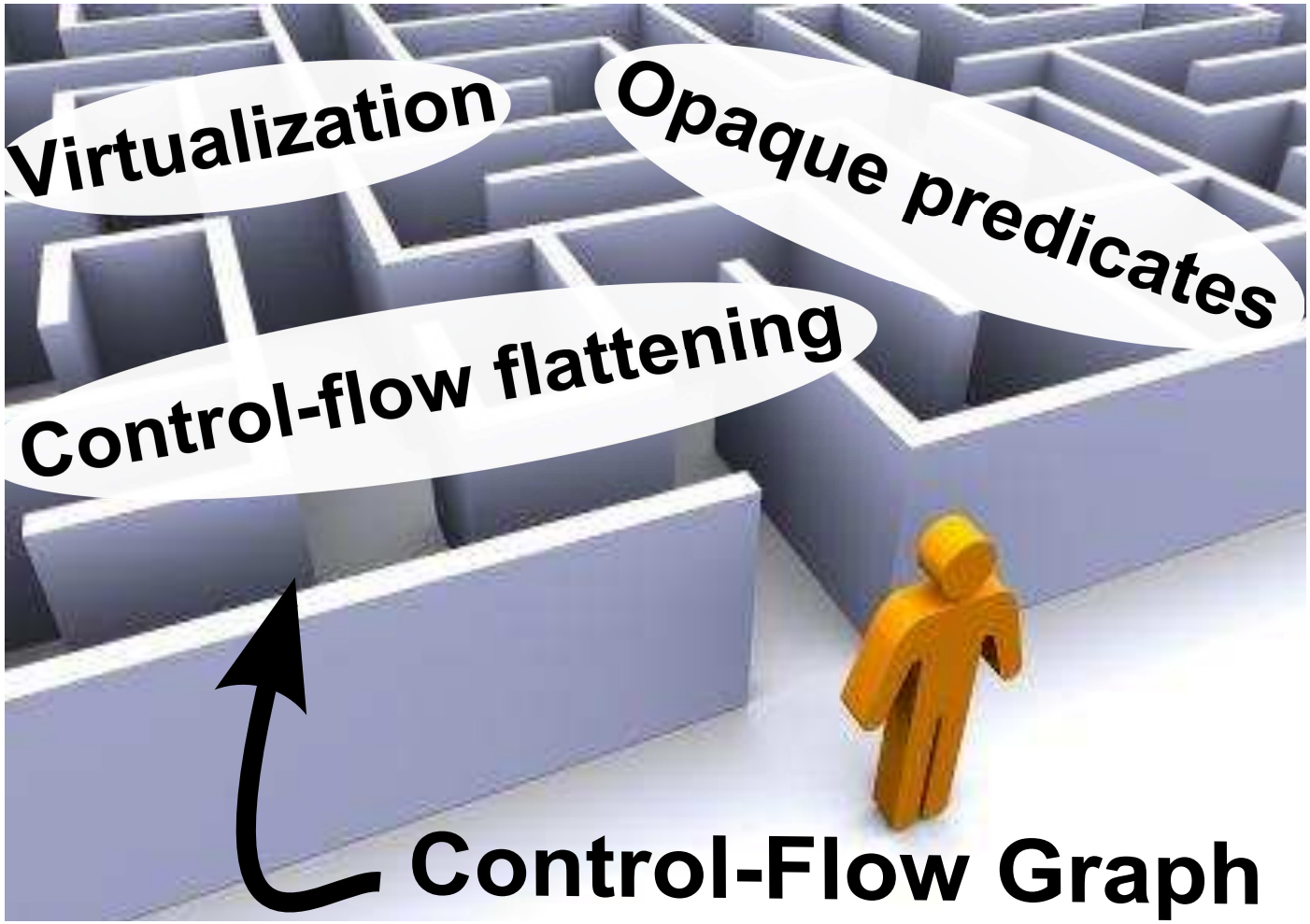
Data structure archaeology: scrape away the dirt and glue back the pieces!

Asia Slowinska, **István Haller**, Andrei Bacs, Silviu Baranga, Herbert Bos

DIMVA 2014
July 10, 2014

Significant research on control-flow obfuscation



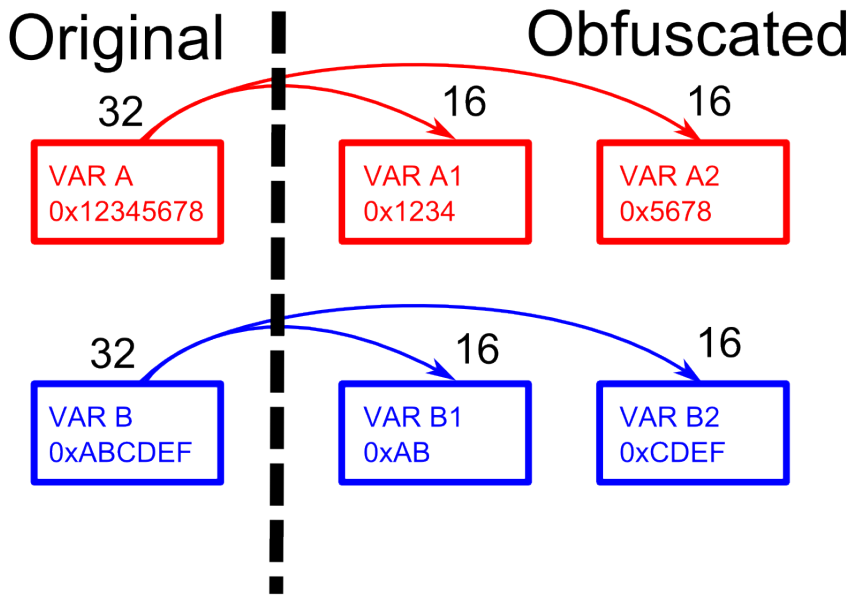
But what about the data?

Program data valuable for reverse engineering

- Recent work on data structure reversing
  - Data layout left intact by control obfuscation
  - Howard (NDSS'11), TIE (NDSS'11)
- Obfuscation resilient code extraction
  - The underlying data-flow is typically unchanged
  - Trace-oriented programming (CCS'13)
  - Compiler techniques could eliminate unnecessary code

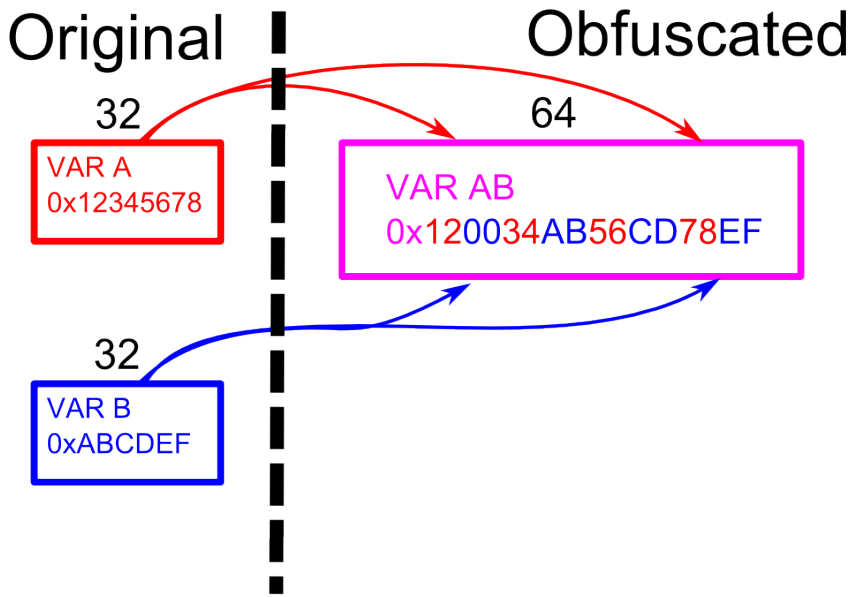
Available data-obfuscation strategies

- Variable Splitting
  - Split variable content across multiple locations
  - Locations may be reordered or interleaved
  - Memory dumps contain garbled data
  - Used in commercial obfuscation products



Available data-obfuscation strategies

- Variable Merging
  - Share memory location by multiple variables
  - Typically combined with splitting



Research questions

- Are the currently suggested data obfuscation techniques viable against a determined attacker?
- Are there fundamental properties of data-flows which make attempts at obfuscation futile?

Research questions

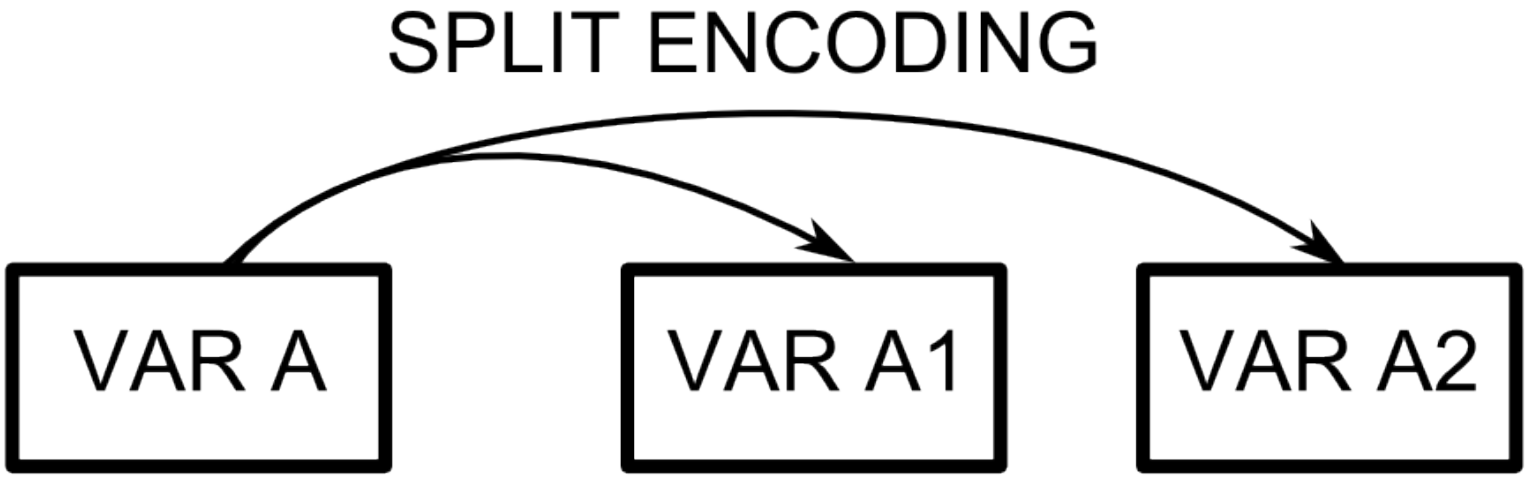
- Are the currently suggested data obfuscation techniques viable against a determined attacker? **NO!**☹
- Are there fundamental properties of data-flows which make attempts at obfuscation futile? **Does not seem like it!**☺

Approach

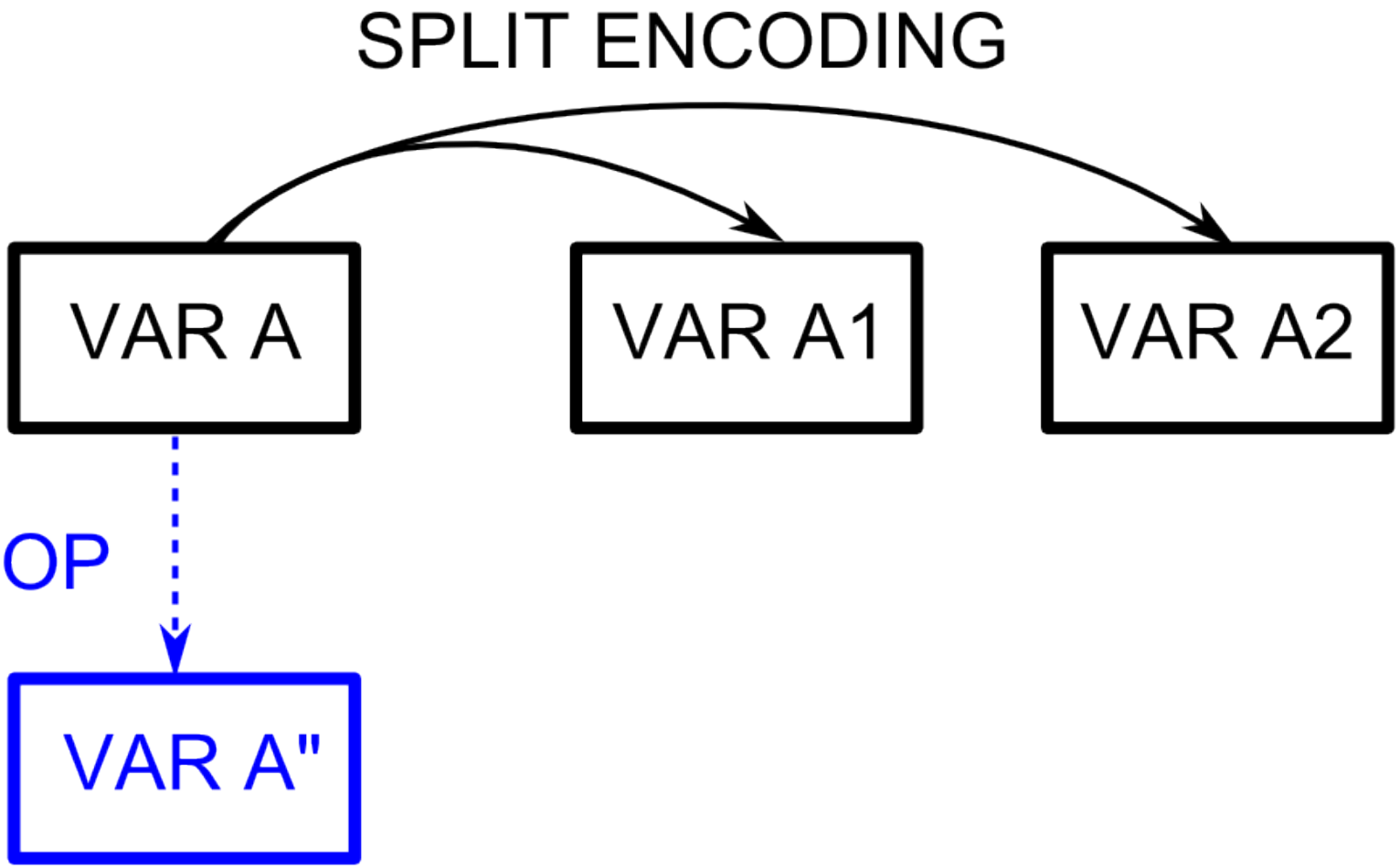
- Carter data deobfuscation tool against split/merge obfuscation
- Leverages inherent properties of the obfuscation
- Based on program access patterns and information flow
- **Focus**: split obfuscation

Split obfuscation

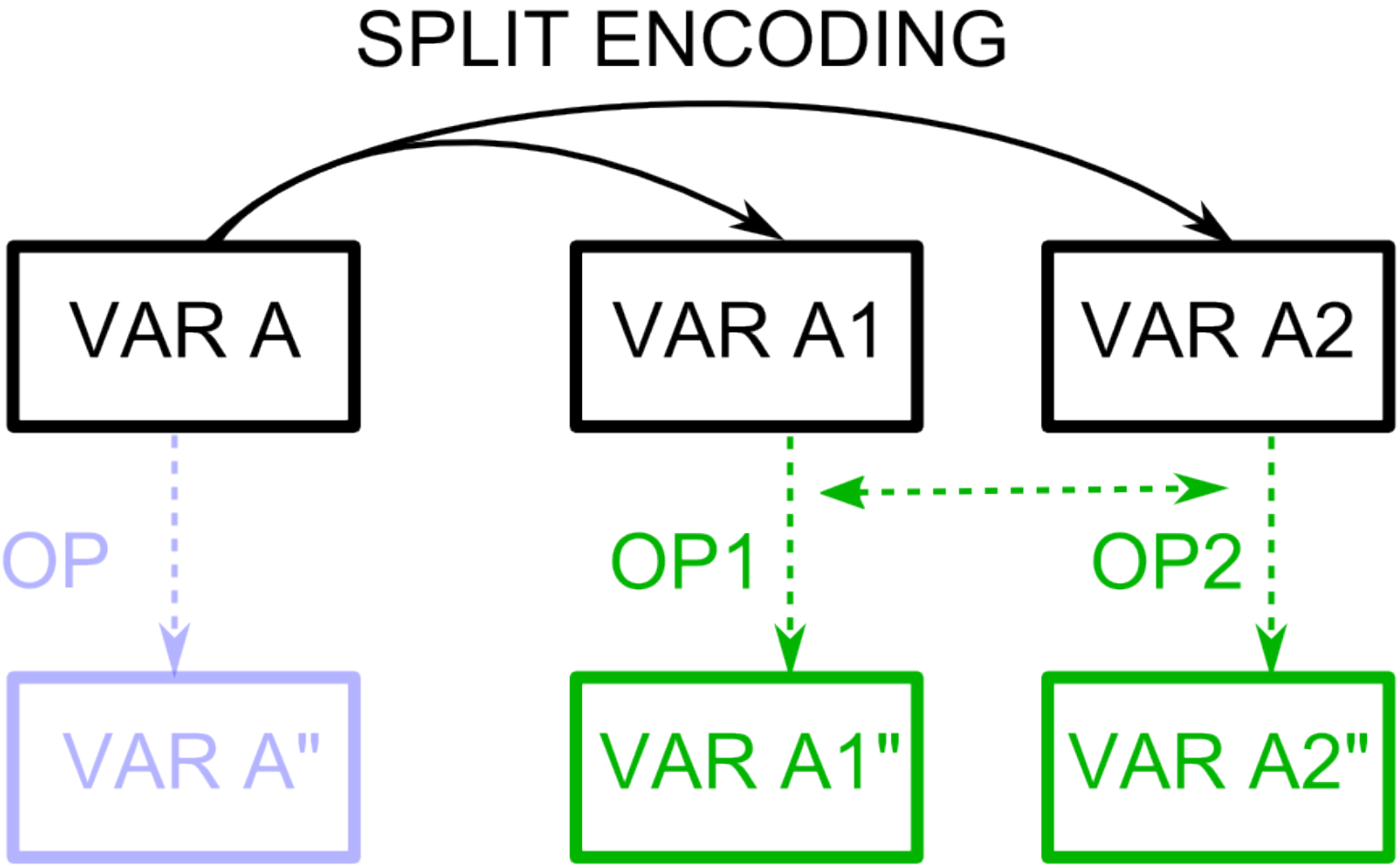
Split obfuscation



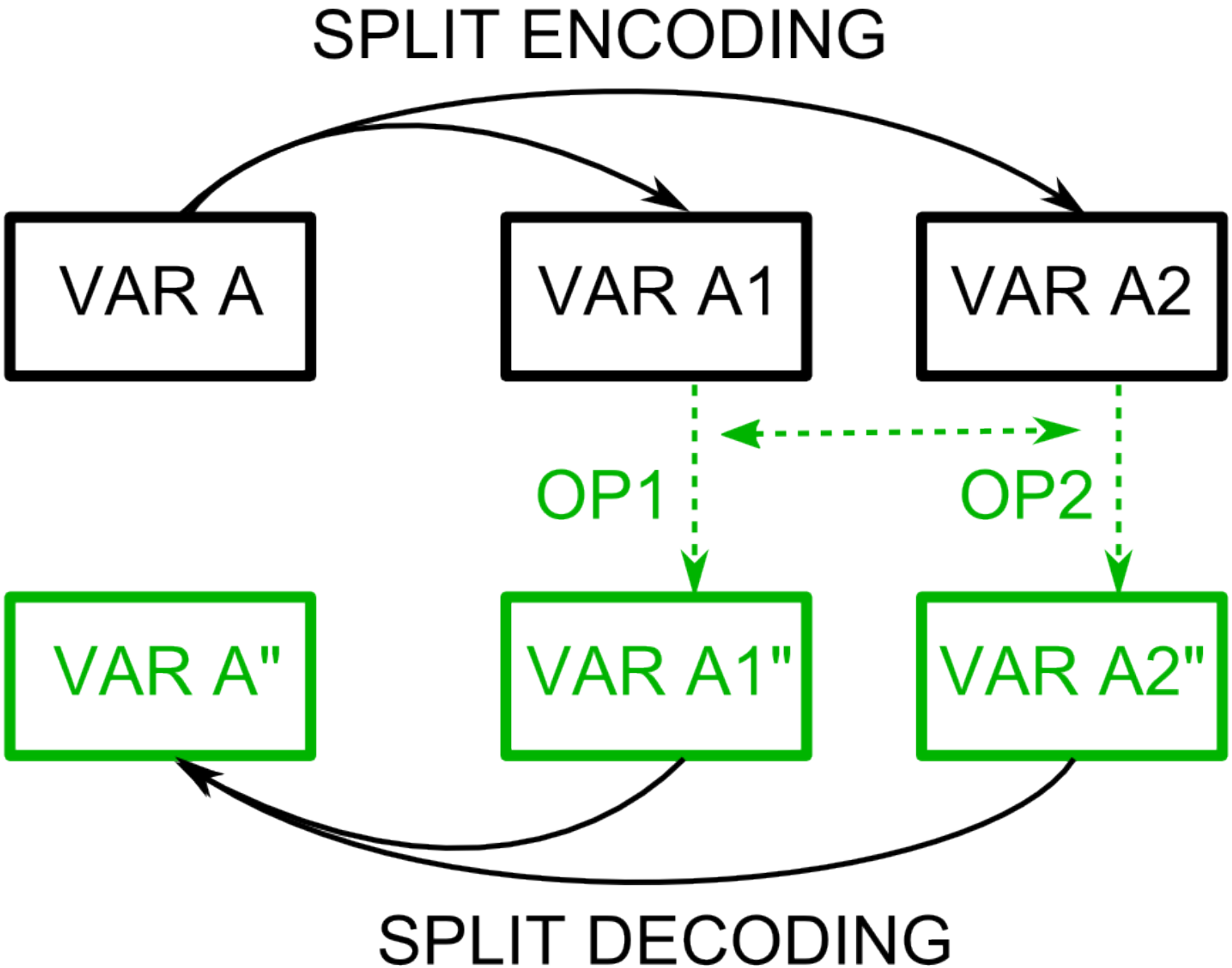
Split obfuscation



Split obfuscation



Split obfuscation

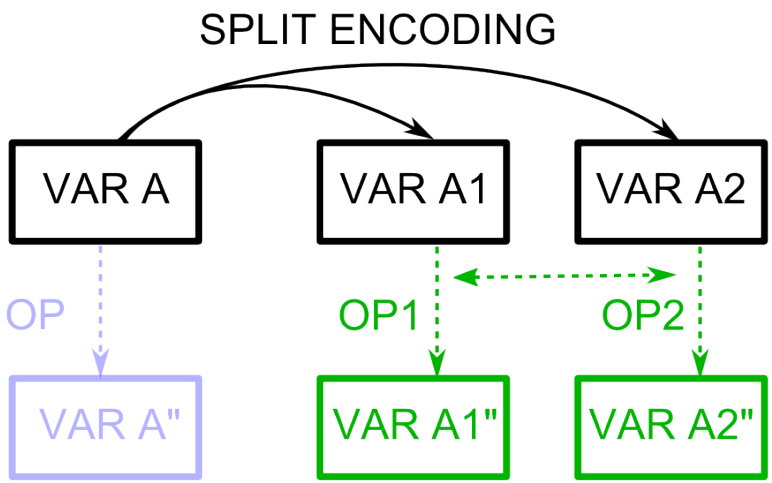


Example of split obfuscation

- Representation: Upper K-1 bits and Lowest bit
- Encoding: $X1 = X/2$ and $X2 = X \ mod \ 2$
- Decoding: $X = X1 * 2 + X2$
- Mapping for addition ($Z = X + Y$)
  - $Z1 = X1 + Y1 + (X2 + Y2)/2$
  - $Z2 = (X2 + Y2) \ mod \ 2$

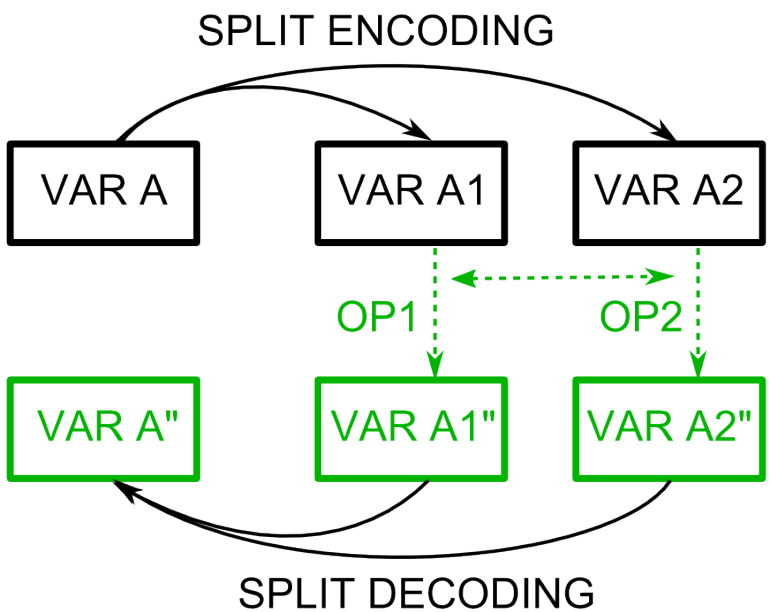
Properties of the obfuscation

- P1: Access to the variable is synchronized
  - The sub-components are always accessed together
  - The accesses are grouped together in time



Properties of the obfuscation

- P2: The variable must be decoded when interacting externally
  - External library calls and pointer dereferences cannot use obfuscated value (system unaware of obfuscation)
  - Decoding involves a merger of the individual data-flows



Potential corner-cases

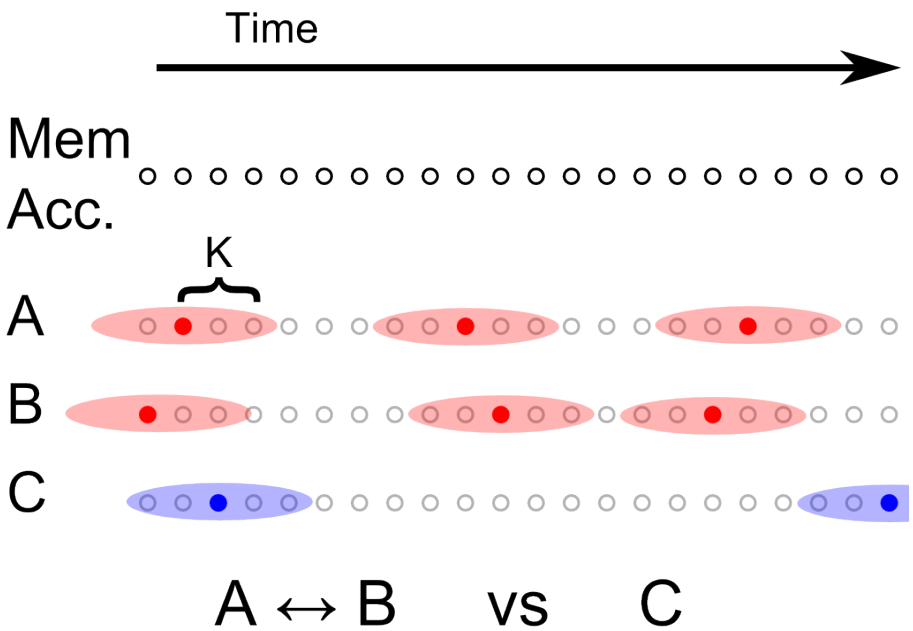
- Compilers encoding 64-bit values in 32-bit binaries
  - The two 32-bit components act as split components
  - The components are accessed together and share data-flows
  - Unavoidable false-positives (small percentage in practice)
- Some variable pairs may be used synchronously
  - For example: array+length, elements of a struct
  - The lack of decoding can filter false positives
  - In most cases the data-flows never merge, thus the candidate is not confirmed to be a split variable

Carter - Setup

- Use memory access trace to detect access groups (P1)
  - Groups of variables accessed together within a short time-frame
  - Based on reference affinity grouping (cache optimization)
- Uses information-flow tracking to confirm candidates (P2)
  - Each entry in a group receives its own tag
  - Tags are propagated along data-flow
  - Carter checks if tags are ever combined

Reference affinity grouping

- Traditionally used to maximize cache-line reuse
- Generates variable partitions where accesses are optimal
- Carter is searching variable groups always accessed together



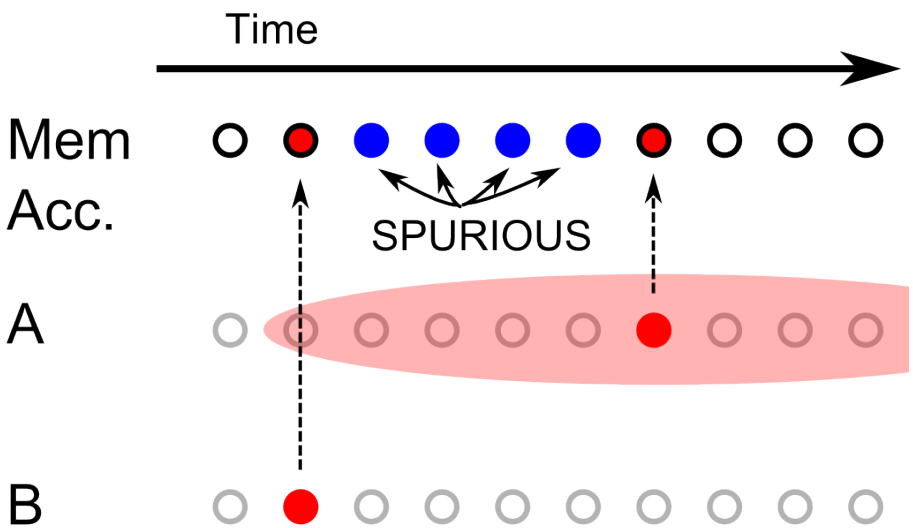
Evaluation - Basic split detection

	TPs	Partially correct	FPs	FNs
base64	79%	21%	0%	0%
expr	100%	0%	0%	0%
factor	58%	42%	1.84%	0%
ls	94%	6%	0.41%	0%
grep	88%	11%	0.82%	1%
gzip	93%	0%	0%	7%
lighttpd	97%	3%	0%	0%
wget	84%	12%	0.76%	4%

Table: Results for deobfuscation of split variables.

Impact of control obfuscation

- Dynamic analysis ensures proper data-flow tracking
- Extra instructions may affect memory trace
- Can be simulated by injecting spurious memory accesses



Evaluation - Combined with control obfuscation

	TPs	Partially correct	FPs	FNs
base64	72%	20%	0%	8%
expr	82%	0%	0%	18%
factor	56%	39%	1.84%	5%
ls	79%	12%	0.83%	9%
grep	72%	16%	0.68%	12%
gzip	100%	0%	0%	0%
lighttpd	94%	2%	0%	4%
wget	78%	10%	0.57%	12%

Table: Results for deobfuscation with 4 spurious accesses.

Impact on future data obfuscation approaches

- Data obfuscation still in its infancy
- More sophisticated approaches necessary in the future
- Static placement of variables is subject to temporal analysis
- Suggestion: aggressive memory reuse
- Additionally: disrupt data-flow tracking

Conclusions

- Carter is a new deobfuscation tool, against the split and split+merge data obfuscation techniques
- Existing techniques vulnerable against determined attackers
- New research avenues to break the assumptions of Carter