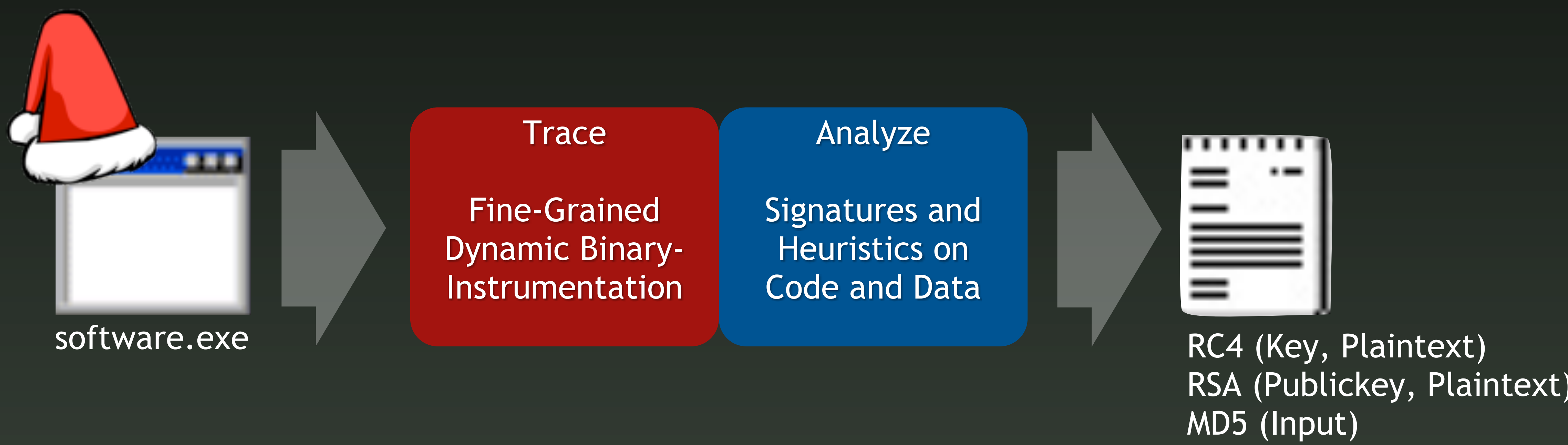
Automatic Identification of Cryptographic Primitives in Software

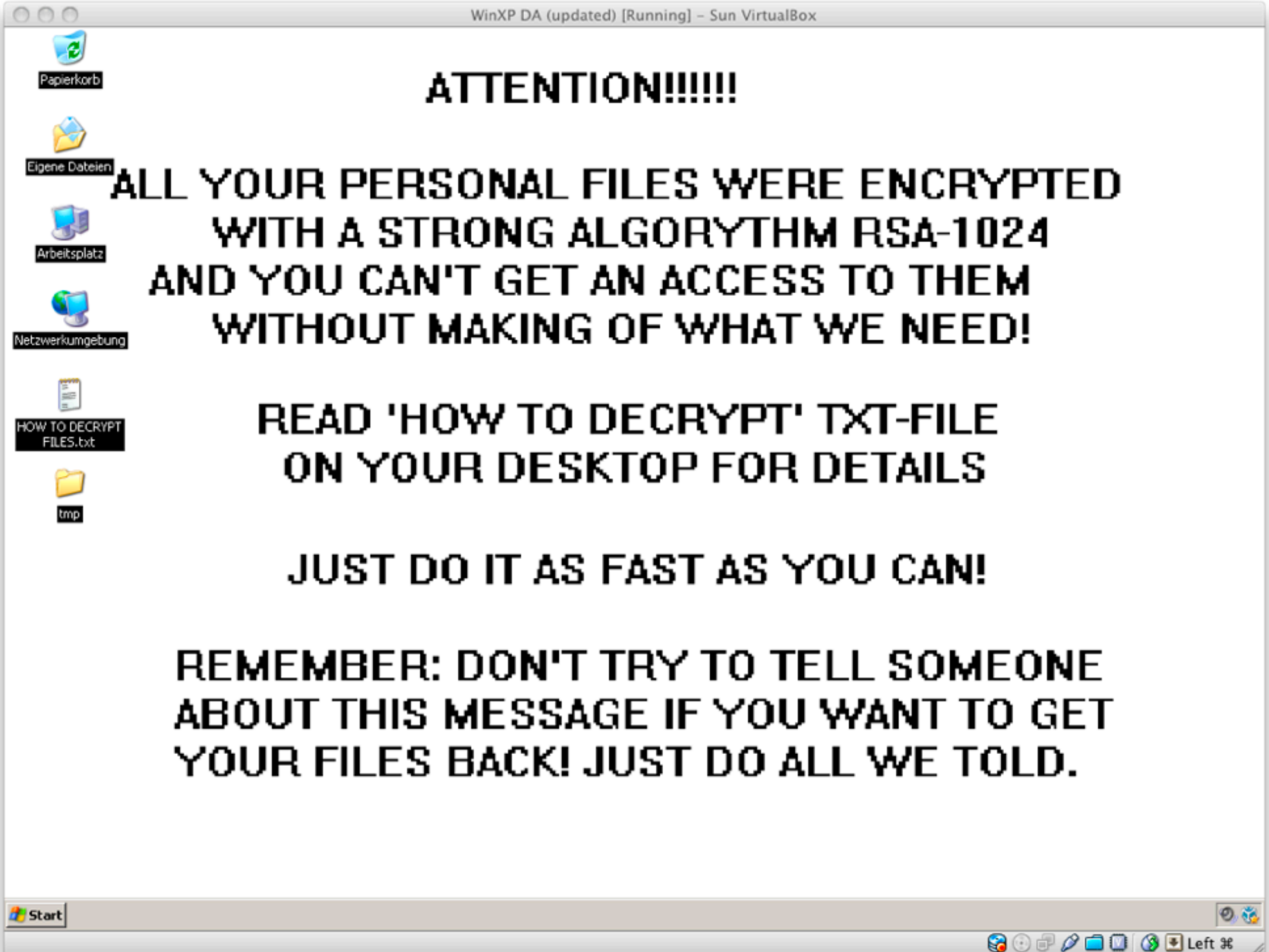
—or: cage fighting with Rice’s Theorem—



27th of December 2010

27c3 in Berlin, Germany

Felix Gröbert <felix@groebert.org>


WinXP DA (updated) [Running] - Sun VirtualBox
Papierkorb
Eigene Dateien
Arbeitsplatz
Netzwerkumgebung
HOW TO DECRYPT FILES.txt
tmp
ATTENTION!!!!!!
ALL YOUR PERSONAL FILES WERE ENCRYPTED
WITH A STRONG ALGORYTHM RSA-1024
AND YOU CAN'T GET AN ACCESS TO THEM
WITHOUT MAKING OF WHAT WE NEED!
READ 'HOW TO DECRYPT' TXT-FILE
ON YOUR DESKTOP FOR DETAILS
JUST DO IT AS FAST AS YOU CAN!
REMEMBER: DON'T TRY TO TELL SOMEONE
ABOUT THIS MESSAGE IF YOU WANT TO GET
YOUR FILES BACK! JUST DO ALL WE TOLD.
Start
Left ⌘

Motivation: Cryptography in Malware

- GpCode: AES-ECB-256 and “STRONG ALGORYTHM RSA-1024”
- ShadowBot: own implementation of MD5, obfuscation 8-bit-XOR
- Conficker: OpenSSL SHA1, reference implementation of MD6, RSA with 1024 bit, later 4096 bit, for signature verification
- Waledac: OpenSSL AES-CBC with zero IV, key exchange protocol with MITM vulnerability, JPEG obfuscation/steganography
- Mebroot / Torpig / Sinowal: BASE64 XOR obfuscation, symmetric cipher with self-designed 58-round Feistel network with 32 bit key, IV-modified SHA1
- Agobot: IRC over SSL
- Storm: P2P/FastFlux subnode authentication with 56 bit RSA, static XOR obfuscation
- Nugache: RSA key exchange, AES-256, RSA-4096 signed MD5 hashes of C&C

» How to help analyst finding cryptographic usage?

Motivation: System Verification

- *“A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.”*
  — Kerckhoffs, 1883

- *“The enemy knows the system.”*
  — Shannon, 1948

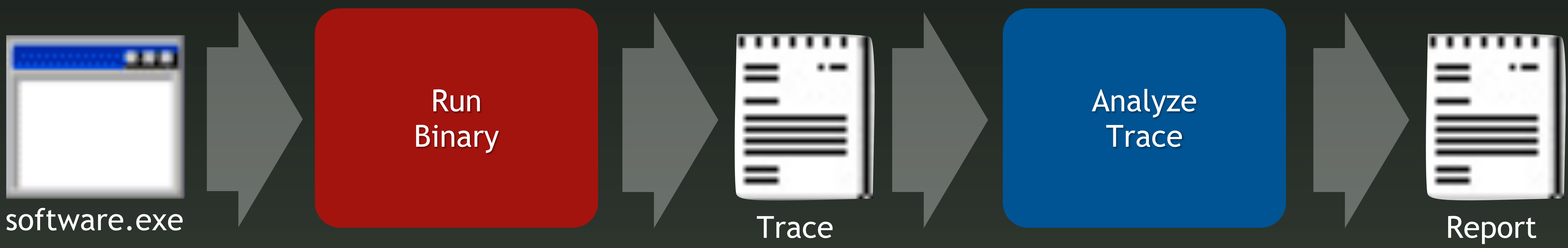
- *“Any security software design that doesn’t assume the enemy possesses the source code is already untrustworthy.”*
  — Raymond, 2004

  - Extended Version: *“Any security software design that doesn’t assume the enemy is able to reverse engineer the source code is already untrustworthy.”*

- **Security evaluation**: determine the used cryptographic primitives and their composition: what, where, how, when

  - Difficult if design or code is not public: Custom DRM and application protocols, malware protocols… is it a secure cryptographic design or just secure by obscurity?

Proposed Solution



Thesis & Contributions

- *If a standardized cryptographic primitive with its input and output is present in an execution trace, an algorithm exists to identify and verify the instance of the primitive including its parameters.*
- Assumptions for the master thesis proof:
  - Not obfuscated or self-modifying code (except one-stage packers)
  - Not just-in-time-compiled or interpreted code
  - Limited to only the cryptographic primitive:
    - No mode-of-operation detection
    - No plaintext encoding or padding detection
    - No compression detection
  - Only Win32-based x86 code

The Moving Targets

original definition → errata → optimization → implementation → compiler optimization → compiled code

Reference	Version	Algo	Mode	Compiler	Key	Input	Output
Beecrypt	4.1.2	AES	ECB	VC dll	128 bit	128B	128B
Beecrypt	4.1.2	MD5	-	VC dll	-	4096B	16B
Crypto++	5.6.0	AES	CFB	VC static	128 bit	128B	128B
Crypto++	5.6.0	DES	CFB	VC static	64 bit	128B	128B
Crypto++	5.6.0	RC4	-	VC static	128B	128B	128B
Crypto++	5.6.0	MD5	-	VC static	-	4096B	16B
Crypto++	5.6.0	RSA	-	VC static	1024 bit	128B	128B
Gladman	07-10-08	AES	CBC	VC static	128 bit	128B	144B
custom	custom	XOR	-	VC static	128B	256B	256B
OpenSSL	0.9.8g	AES	CFB	MinGW, VC static	128 bit	128B	128B
OpenSSL	0.9.8g	DES	ECB	MinGW, VC static	64 bit	128B	128B
OpenSSL	0.9.8g	RC4	-	MinGW, VC static	128B	128B	128B
OpenSSL	0.9.8g	MD5	-	MinGW, VC static	-	4096B	16B
OpenSSL	1.0.0-beta3	RSA	-	VC dll	512 bit	128B	192B

Related Work: Static Approaches

Name	Author(s)	Platform	Version
Krypto Analyzer (KANAL)	Several	PEiD	2.92
Findcrypt plugin	Ilfak Guilfanov	IDA Pro	2
SnD Crypto Scanner	Loki	OllyDBG	0.5b
Crypto Searcher	x3chun	standalone	2004.05.19
Hash & Crypto Detector (HCD)	Mr Paradox, AT4RE	standalone	1.1
DRACA	Ilya O. Levin	standalone	0.5.7b

	KANAL		Findcrypt		SnD		x3chun		HCD		DRACA	
gladman aes	+		-		+		-		+		-	
cryptopp aes	+	2	-	2	+	2	+	1	+		-	
openssl aes	+	6	+	3	+	1	+	3	-	6	-	1
cryptopp des	+	3	+	2	+	3	+	2	-	1	+	
openssl des	+		-		+		+		-		-	
cryptopp rc4	-		-		+	3	-		-		-	
openssl rc4	-		-		-		-		-		-	
cryptopp md5	+		+	1	+	1	+		+		+	1
openssl md5	+		+	1	+		+		+		+	1
openssl rsa	-		-		-		-		-		-	
cryptopp rsa	-	4	-	3	-		-	3	-	4	-	1

+ = algorithm found
number = number of false-positives

	KANAL	Findcrypt	SnD	x3chun	HCD	DRACA
beecrypt.dll	11	18	7	5	7	4
libeay.dll	126	14	17	13	20	7

number = number of found algorithms

- Tools require unpacked binary
- Byte-orientated signatures
- Evaluation:
  - All detect MD5
  - No tool detects RSA
  - RC4 is only detected once by SnD
  - No tool detects dynamically linked cryptographic code

Related Work: Dynamic Approaches

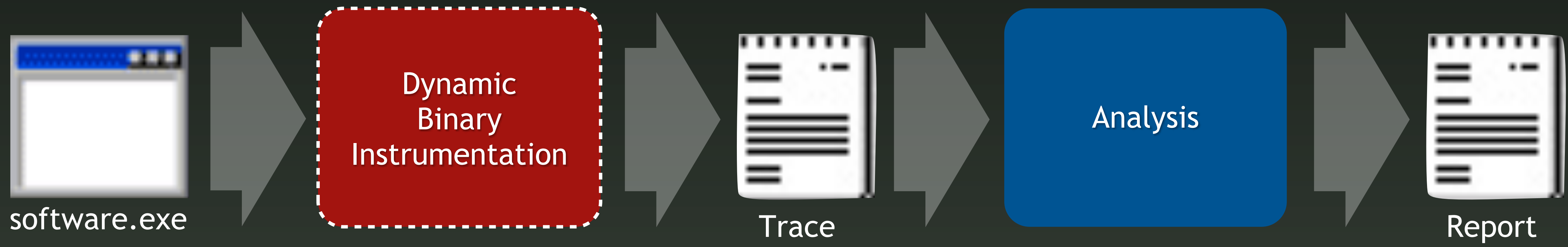
- Wang et al. (PoC with Agobot) 2008
  - Crypto operations on tainted data differ vastly from other types of modifications (high percentage of bitwise arithmetic instructions)
  - Cumulative bitwise instruction percentage is used to determine turning point between encryption phase and other processing phases
- Caballero et al. (PoC with MegaD) 2009
  - Encrypted block processing » rather use a function/block-wise bitwise instruction percentage instead of a cumulative
- Noé Lutz (PoC with Kraken) 2008
  - Determines whether the read/write set inside a loop decreases information entropy of tainted memory

Related Work: Adjacent Approaches

- Key search in data
  - Shamir, Van Someren, Janssens: RSA in bit strings
  - Halderman et al.: “Coldboot Attack”
  - Stevens: XORsearch
  - Boldewin: OfficeMalScanner

- Loosely related reverse code engineering approaches
  - BinCrowd: collaborative reverse engineering
  - REGoogle: IDA plugin to codesearch for imports and constants

Our Approach: Dynamic Instrumentation



Execution Tracing

- Dynamically instrument target binary code using PIN tool
  - PIN is a free-of-charge dynamic binary instrumentation framework by Intel
  - Can be extended by custom PIN tools (C++)
- Optionally filter by DLL or thread ID
- Start trace after a specific number of instructions
- Record compressed trace file
- Dynamic approach constraint: code must be executed
- Dynamic approach advantage: data can be examined

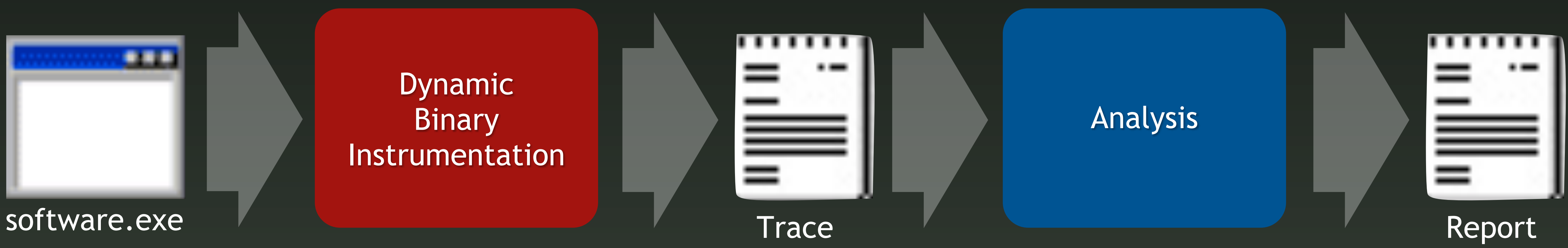
Execution Tracing Example

```
[...]
R|32|0022F948=0x22f9a4
0x7c9111f3|@1|@2|0x0016|0|mov esi, dword ptr [ebp+0x8]|esi=0x22f9a4
[...]
```

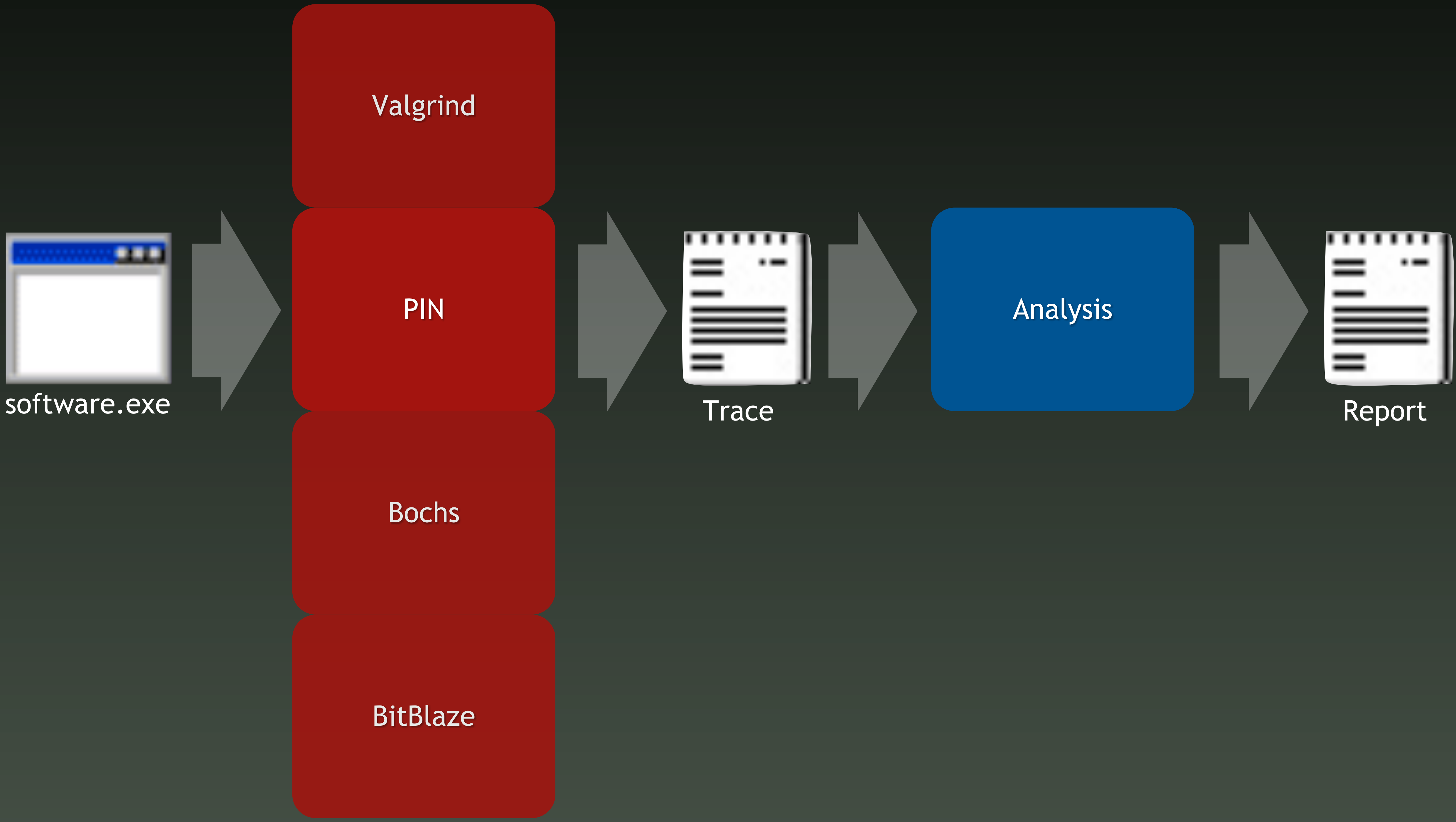
Execution Tracing Example

```
Memory Access Mode|Size|Address=0x22f9a4

[...]
R|32|0022F948=0x22f9a4
0x7c9111f3|@1|@2|0x0016|0|mov esi, dword ptr [ebp+0x8]|esi=0x22f9a4
[...]

EIP|Library|Function|Offset|Thread ID|Instruction Disassembly|Changed Registers
```

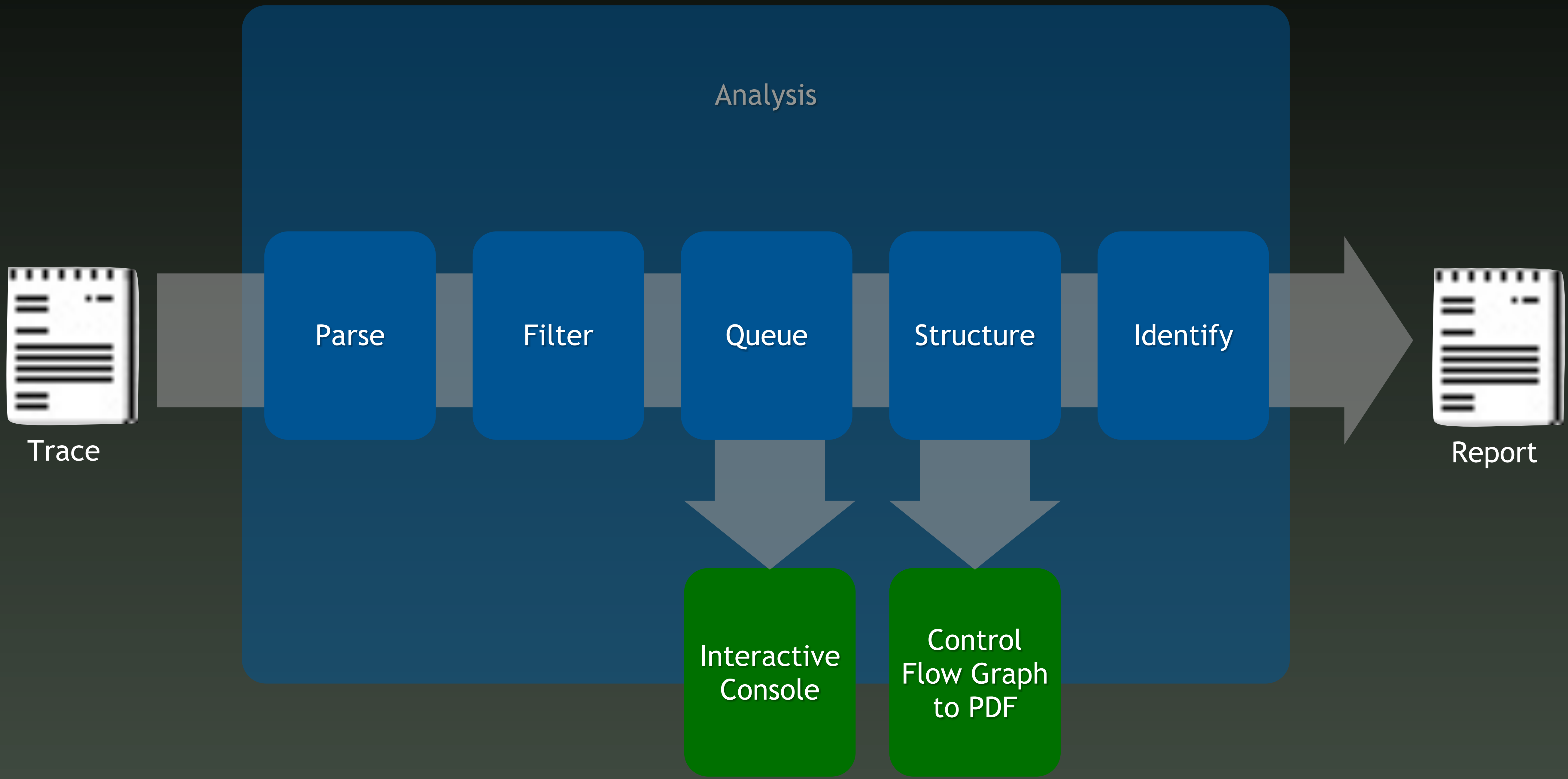
Tracing Framework = Exchangeable



Tracing Framework = Exchangeable

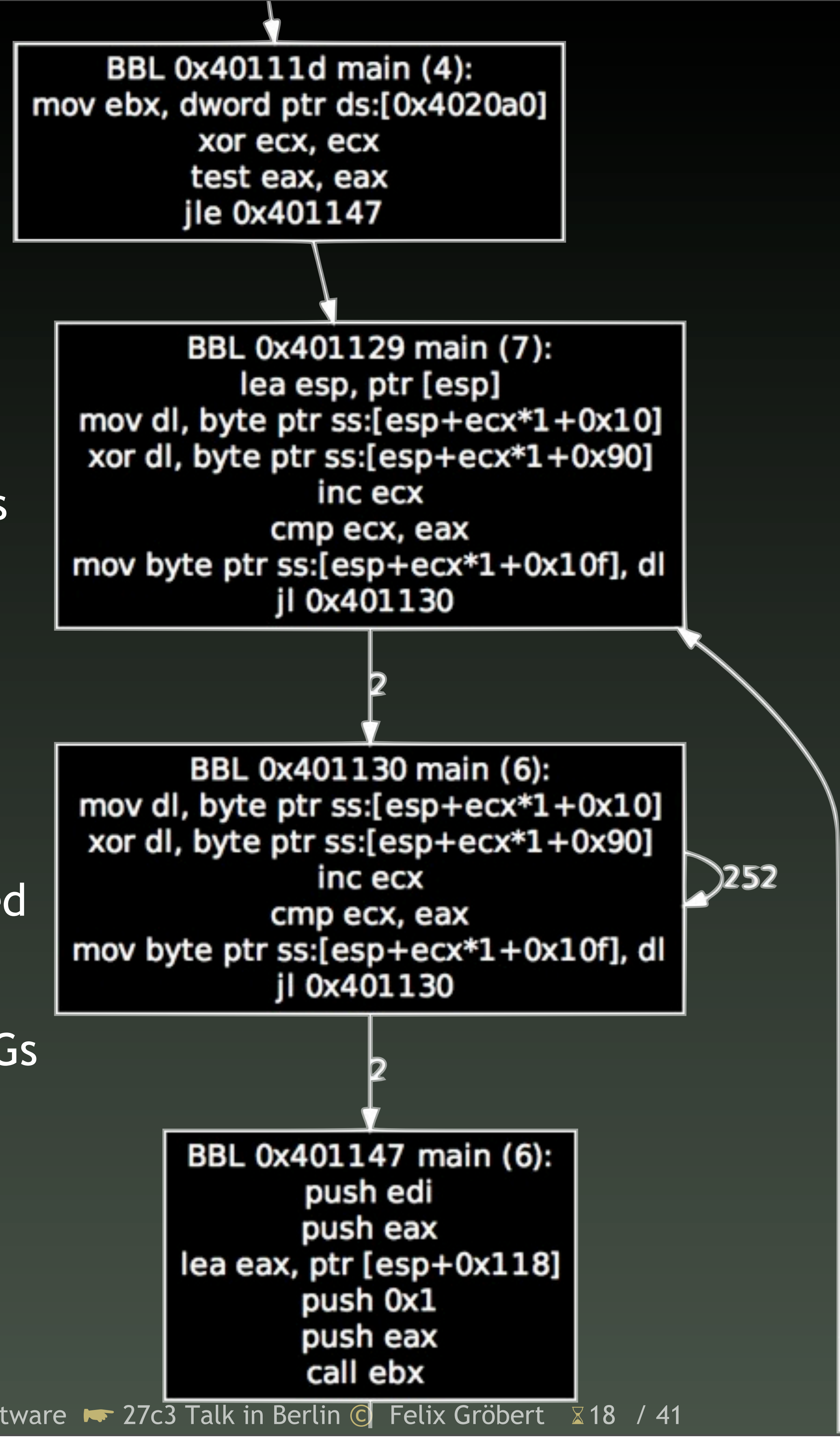


Implementation: Analysis



BBL & CFG

- Basic block (BBL) generation
  - Two pass over sequential instruction trace:
    (1) determine starts and ends of BBLs
    (2) populate data structure
  - Advantage of dynamic analysis: known targets of indirect branches
- Control flow graph (CFG) generation
  - Single pass over sequentially executed BBLs
  - Use Graphviz to export visualized CFGs



Loop Detection

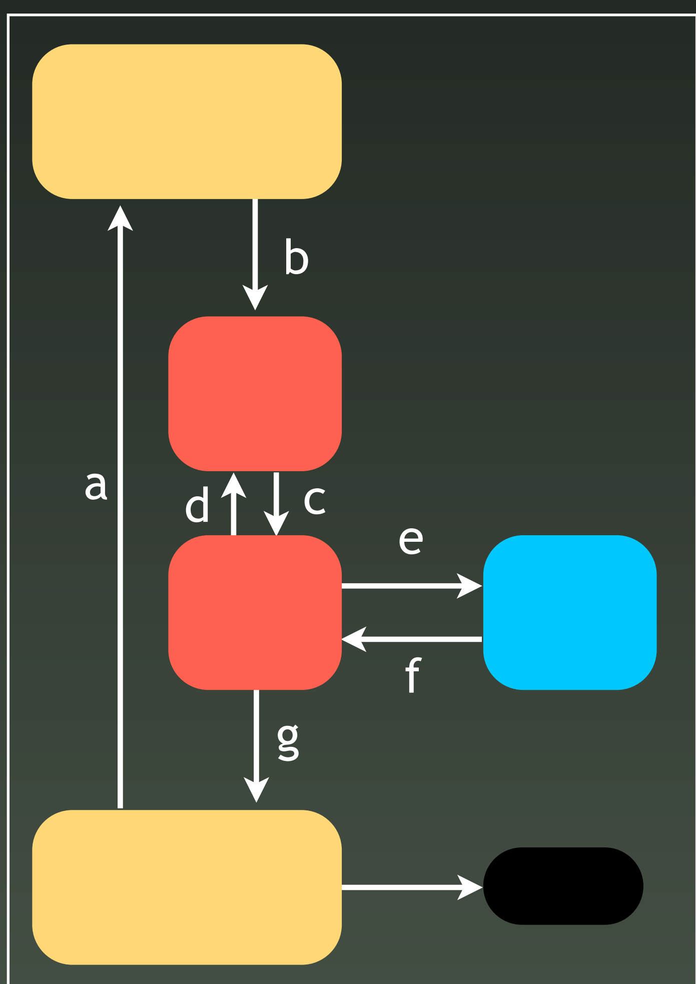
- Based on Tubella et al.: "Control Speculation in Multithreaded Processors through Dynamic Loop Detection"
- Detects a loop by multiple executions of the same addresses
- Exposes the following features (unlike CFG-based Lengauer-Tarjan algorithm):
  - Number of loop executions
  - Number of loop iterations per loop execution (min/avg/max/total)
  - Set of instructions belonging to a loop body
  - Hierarchy of nested loops

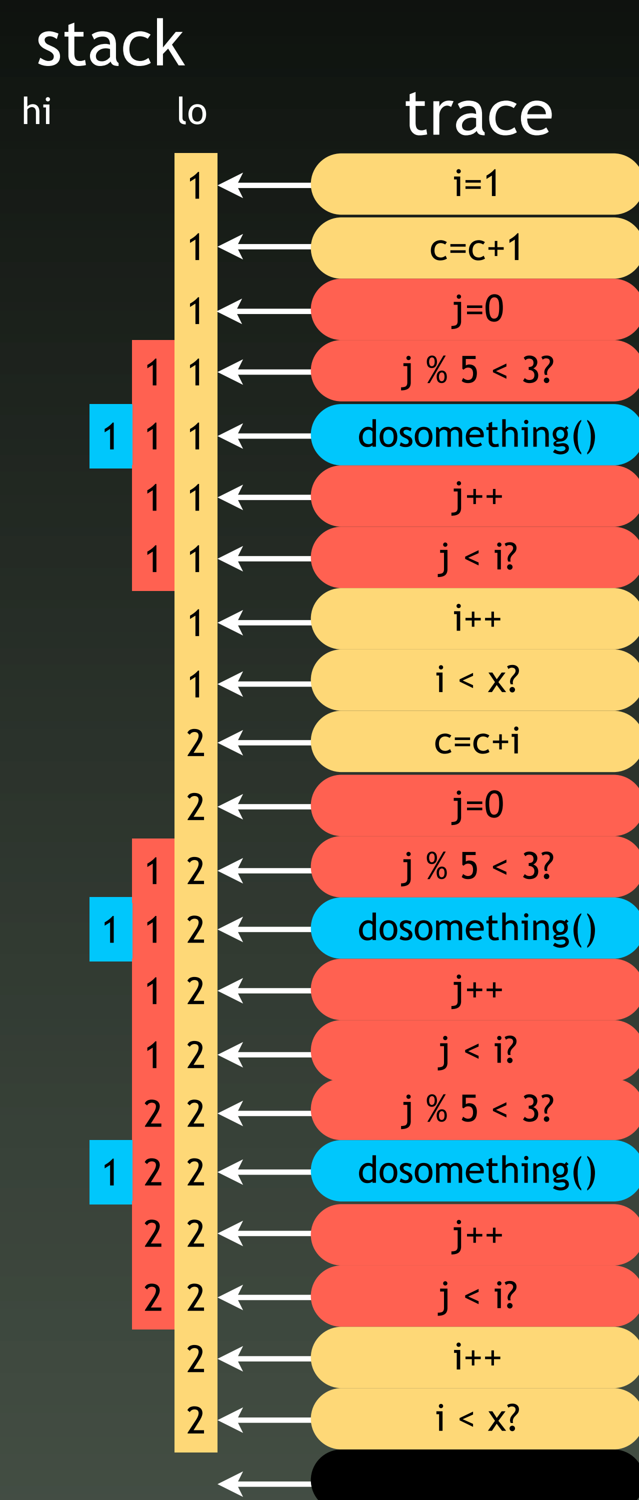
Loop Detection: Example

target code

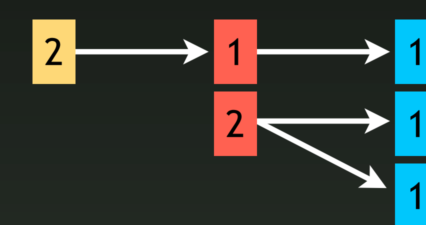
```
for(i = 1; i < x; i++) {
  c = c + i;
  for(j = 0; j < i; j++) {
    if(j % 5 < 3)
      dosomething(c,j);
  }
}
```

CFG





result

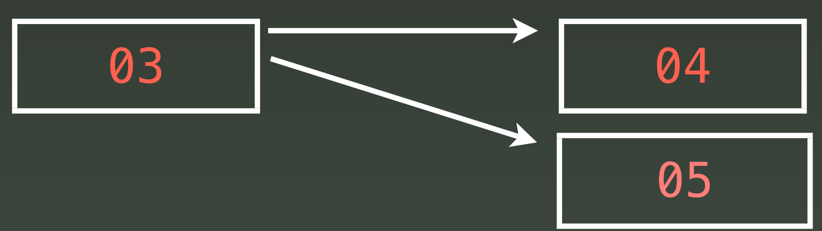


Outer yellow loop was iterated twice. Red loop was executed twice, first with one, then with two iterations. Inner blue loop was executed three times, each with one iteration.

Memory Reconstruction from Trace

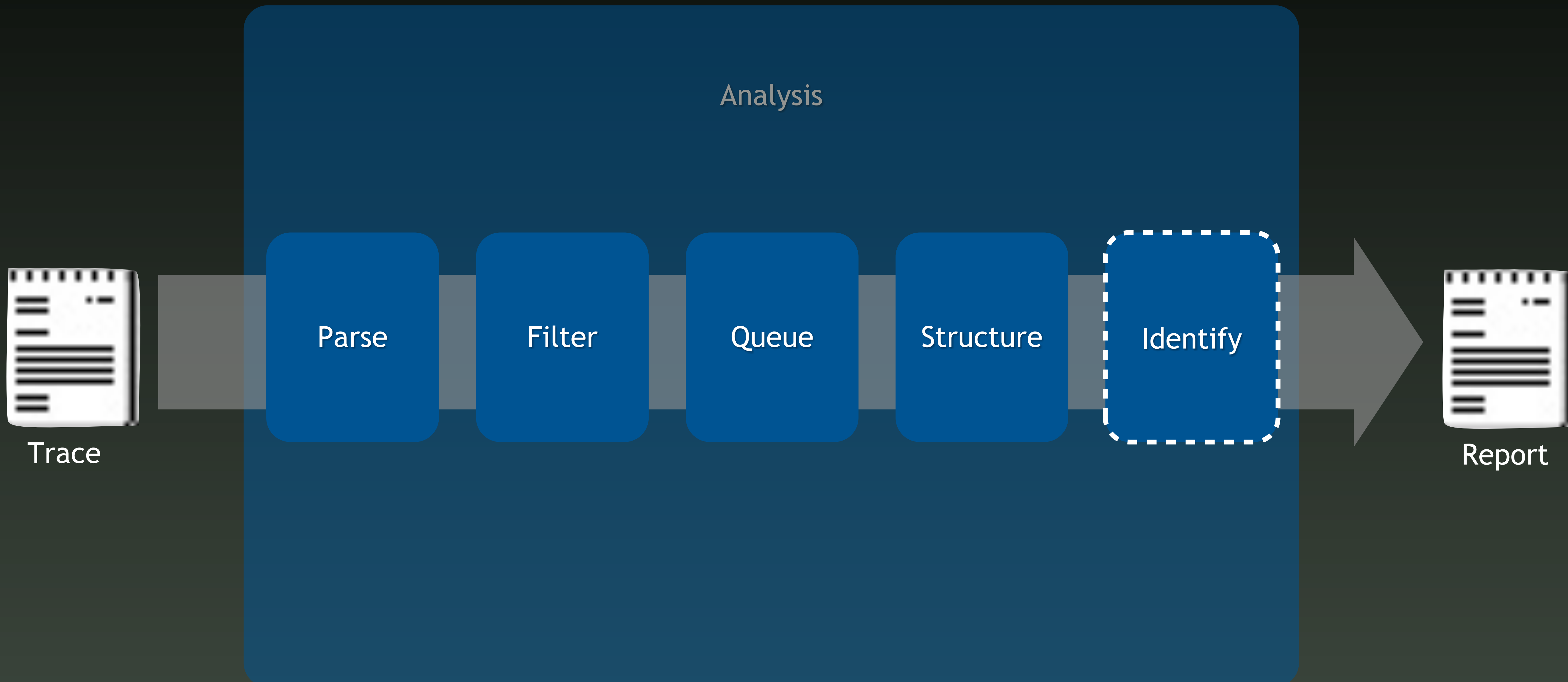
- Cryptographic data is generally larger than CPU supported register size. Thus the reconstruction of blocks of relating data is beneficial to key reconstruction
- For an instruction+memory trace:
  - A recursive search checks whether at the next address (last address +length) a similar memory value has been accessed
  - The search continues if the structure is continued (eg two byte access) at the next address. A search may be split if there are multiple matching next values
- The search also checks whether the access occurred nearby the last access in the current block in the instruction trace

Address	Values (eg only reads)		
001a0000	31313131		
001a0004	32323232		
001a0008	33333333		
001a000c	34343434		
001a0010	35353535	01	
001a0011	02		
001a0012	03		
001a0013	04	05	60
001a0014	36363636	05	08
001a0015	06	0d	
001a0016	07	15	



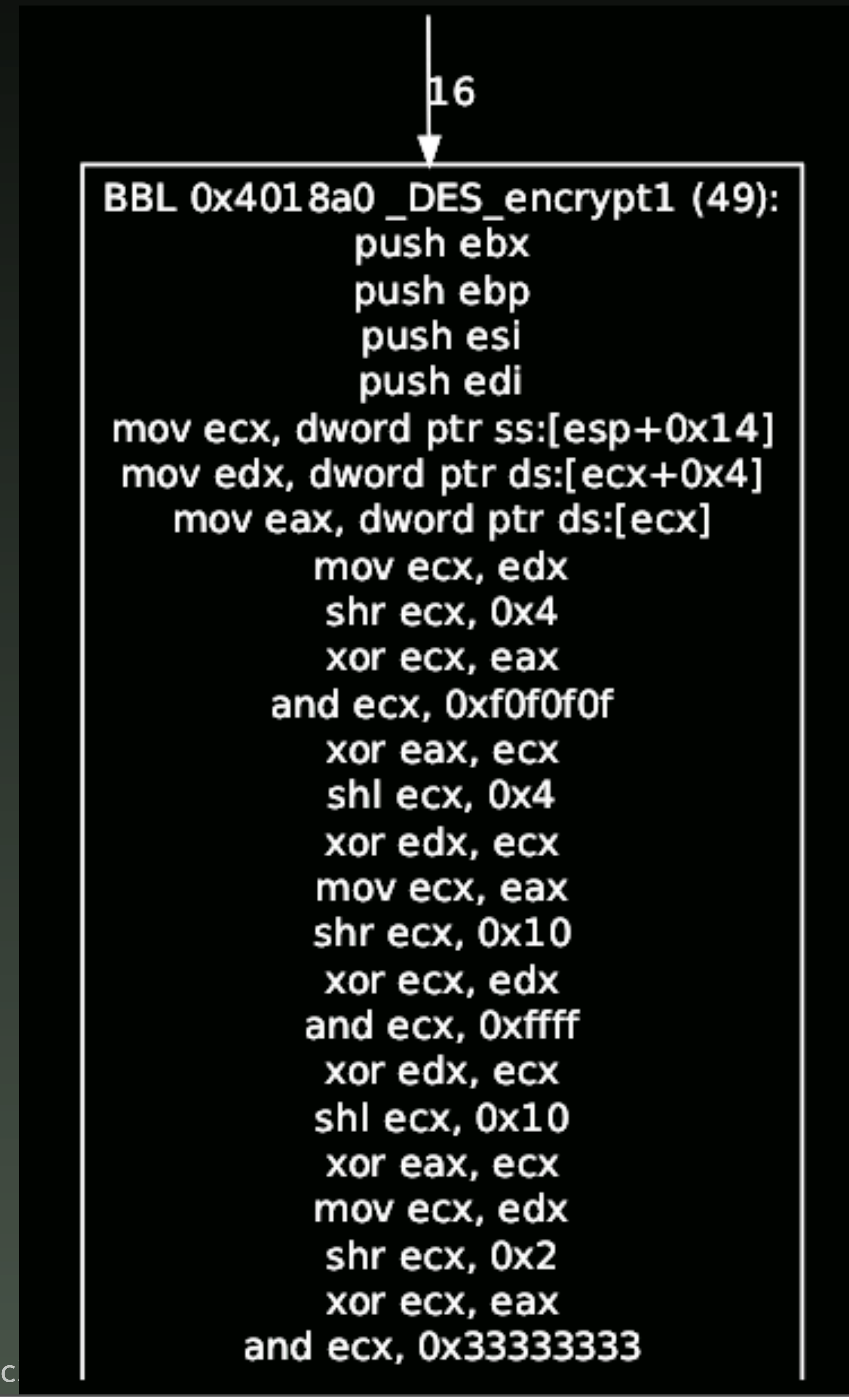
Nearby (in the trace) the access to 03, the access to 04 and 05 occurred, but not the access to 60

Implementation: Analysis



Identification Observations

Cryptographic code makes excessive use of bitwise arithmetic instructions



Identification Observations

Constants and sequences of mnemonics indicate the type of cryptographic algorithm

beecrypt

```
rol ecx, 0x7
add ecx, edi
mov ebp, esi
xor ebp, edi
and ebp, ecx
xor ebp, esi
add ebp, ebx
lea edx, ptr [edx+ebp*1-0x173848aa]
mov ebx, dword ptr ds:[eax+0x18]
rol edx, 0xc
add edx, ecx
mov ebp, edi
xor ebp, ecx
and ebp, edx
xor ebp, edi
add ebp, ebx
```

cryptopp

```
rol ecx, 0x7
add ecx, edi
mov ebp, esi
xor ebp, edi
and ebp, ecx
xor ebp, esi
mov dword ptr ss:[esp+0x54], ebx
mov ebx, dword ptr ds:[eax+0x4]
add ebp, ebx
lea edx, ptr [edx+ebp*1-0x173848aa]
rol edx, 0xc
add edx, ecx
mov ebp, edi
xor ebp, ecx
and ebp, edx
xor ebp, edi
```

openssl

```
rol edx, 0x7
add edx, ebp
mov edi, ebx
xor edi, ebp
and edi, edx
xor edi, ebx
add edi, esi
mov esi, dword ptr ds:[ecx+0xc]
lea esi, ptr [edi+esi*1-0x173848aa]
mov edi, ebp
xor edi, edx
rol esi, 0xc
add esi, edx
and edi, esi
xor edi, ebp
add edi, dword ptr ds:[eax-0x30]
```

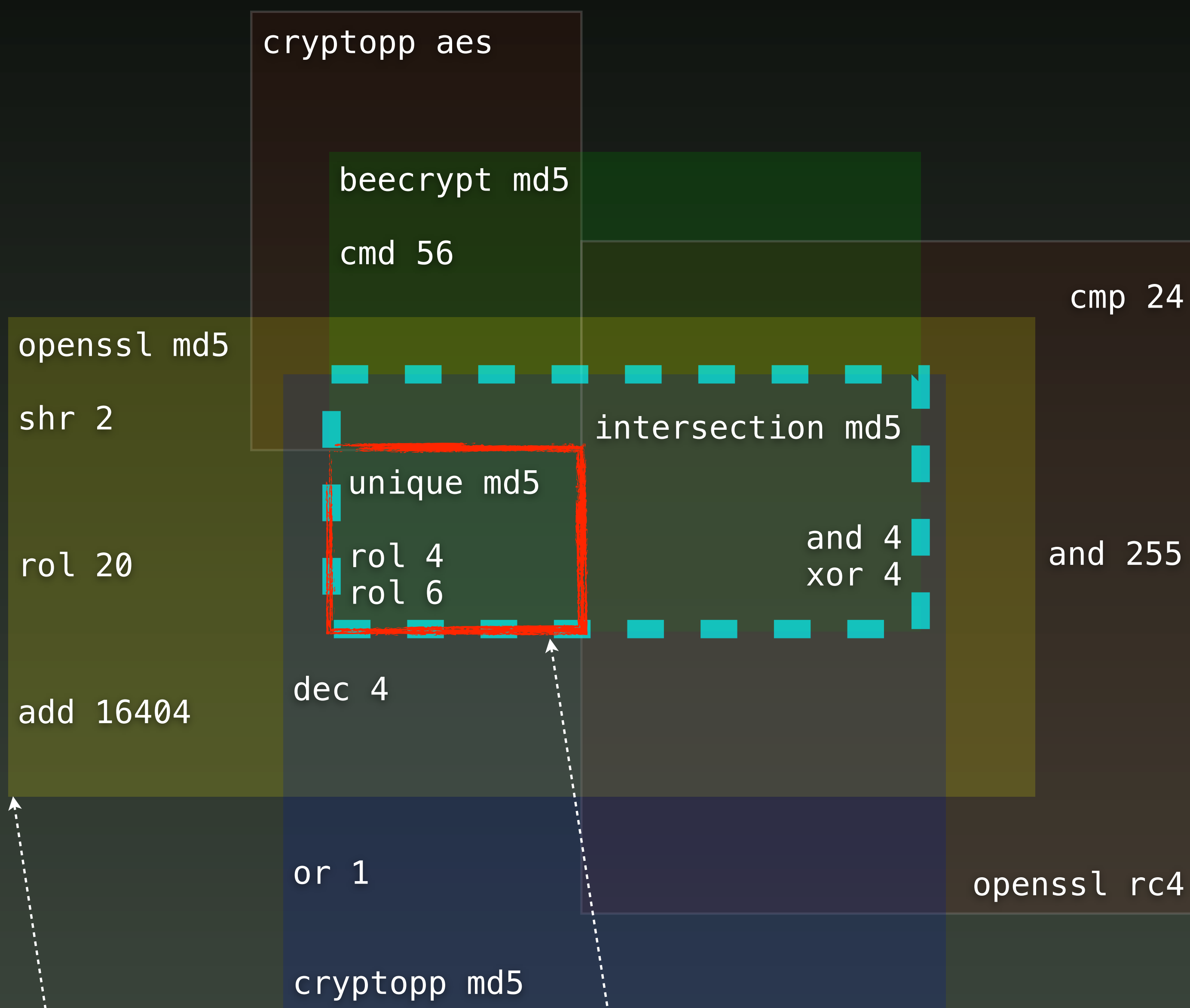
Identification Observations

- *Cryptographic code contains loops*
  - Similar set of operations is commonly applied to the state with a different round key
  - Unrolling of loops is used to optimize algorithms
- *Input and output to cryptographic code have a predefined, verifiable relation*
  - If algorithm is known, the relation can be verified
  - Blinded RSA has (somewhat) non-deterministic relation

Implemented Analysis Modules

- Signature identification methods
  - sigAPI
  - constants in memory
  - mnemonic sequences
  - <u>(mnemonic, constant) tuples</u>
- Related work
  - Caballero (bitwise-instruction-percentage for functions and BBLs)
  - Wang (turning-point in cumulative bitwise-instruction-percentage)
  - Lutz (entropy-based)
- Generic identification methods
  - xor detection
  - <u>loop differ</u>
  - <u>data verifier</u>

(Mnemonic,Constant)-Tuples



- Foreach implementation, build a set of bitwise instructions with static constants, eg (rol, 0x14)
- Foreach algorithm, build a intersection and a unique set
- For an unknown set of instructions from a trace, the match degree is the percentage of found signature tuples in the unknown set

small set, strong relation to algorithm

broad set, loose relation to algorithm
high relation to type of implementation

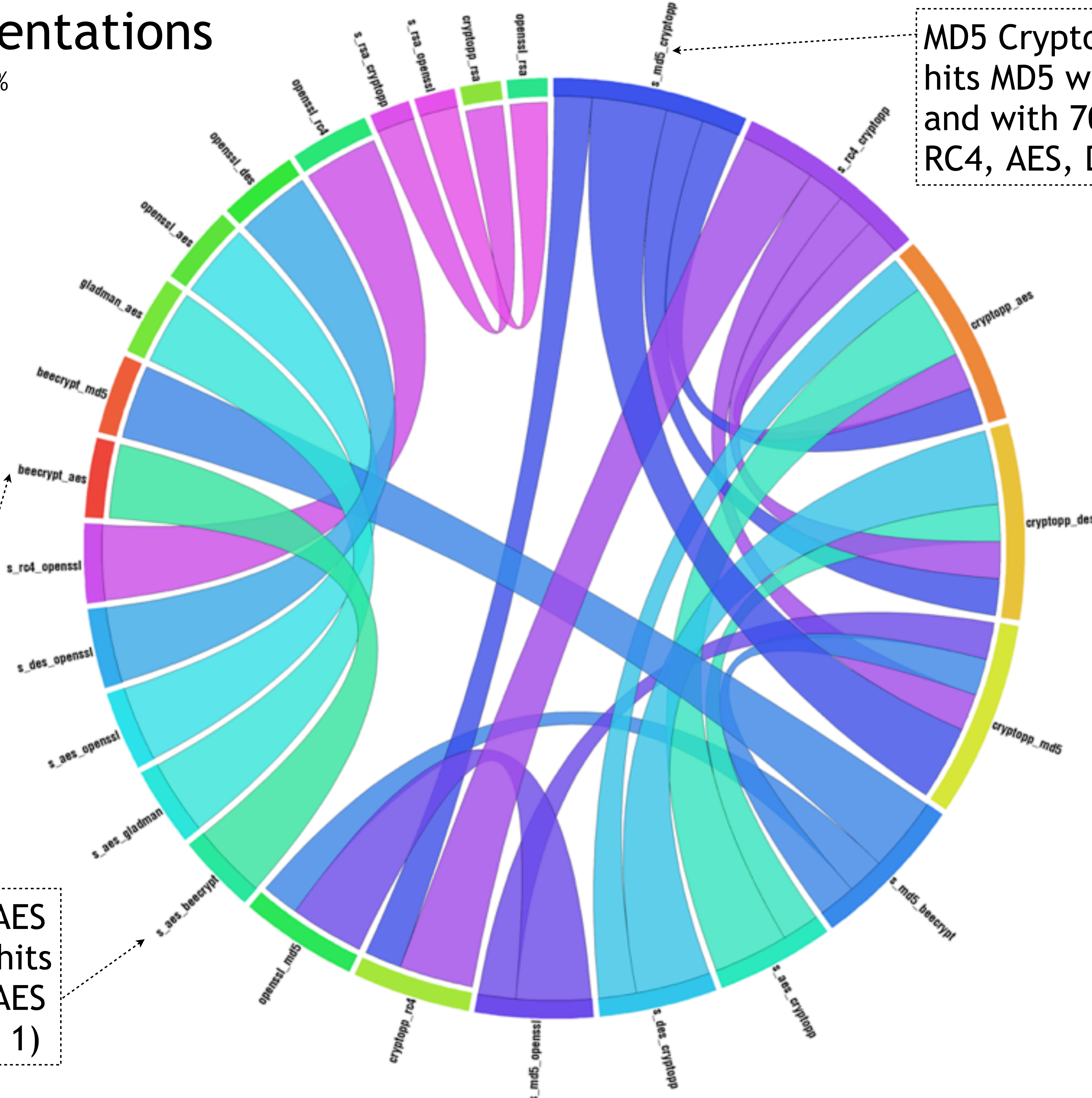
Performance

	beecrypt aes	beecrypt md5	cryptopp aes	cryptopp des	cryptopp md5	cryptopp rc4	cryptopp rsa	gladman aes	openssl aes	openssl des	openssl md5	openssl rc4	openssl rsa	xor256	xor4096
rc4 unique	0 %	0 %	100 %	100 %	100 %	100 %	100 %	0 %	50 %	0 %	0 %	100 %	0 %	0 %	0 %
des unique	0 %	0 %	44 %	100 %	44 %	44 %	44 %	22 %	33 %	100 %	11 %	0 %	0 %	0 %	0 %
rsa unique	22 %	8 %	58 %	61 %	50 %	46 %	89 %	34 %	18 %	1 %	7 %	1 %	89 %	0 %	0 %
md5 unique	0 %	100 %	6 %	29 %	100 %	6 %	12 %	0 %	0 %	0 %	100 %	0 %	0 %	0 %	0 %
rc4 intersect	68 %	68 %	100 %	100 %	100 %	100 %	95 %	64 %	77 %	77 %	68 %	100 %	68 %	59 %	59 %
aes intersect	100 %	82 %	100 %	100 %	82 %	82 %	94 %	100 %	100 %	88 %	88 %	71 %	88 %	59 %	59 %
des intersect	56 %	51 %	87 %	100 %	77 %	77 %	82 %	51 %	74 %	100 %	64 %	46 %	64 %	38 %	38 %
rsa intersect	34 %	28 %	71 %	71 %	63 %	57 %	93 %	41 %	35 %	24 %	29 %	16 %	92 %	12 %	12 %
md5 intersect	40 %	100 %	60 %	67 %	100 %	52 %	62 %	26 %	45 %	43 %	100 %	38 %	52 %	36 %	36 %
rc4 cryptopp	13 %	14 %	83 %	82 %	82 %	100 %	57 %	16 %	17 %	16 %	15 %	11 %	31 %	8 %	8 %
rc4 openssl	60 %	58 %	68 %	63 %	58 %	55 %	65 %	38 %	55 %	53 %	50 %	100 %	45 %	53 %	53 %
aes beecrypt	100 %	33 %	35 %	34 %	27 %	27 %	58 %	62 %	41 %	29 %	27 %	26 %	40 %	24 %	24 %
aes gladman	41 %	12 %	27 %	28 %	23 %	22 %	45 %	100 %	21 %	17 %	13 %	11 %	32 %	8 %	8 %
aes cryptopp	12 %	13 %	100 %	73 %	64 %	62 %	59 %	15 %	16 %	14 %	14 %	10 %	29 %	6 %	6 %
aes openssl	52 %	34 %	56 %	55 %	47 %	47 %	62 %	40 %	100 %	45 %	37 %	30 %	52 %	26 %	26 %
des cryptopp	12 %	14 %	74 %	100 %	65 %	62 %	53 %	15 %	15 %	15 %	15 %	10 %	29 %	6 %	6 %
des openssl	26 %	22 %	36 %	38 %	29 %	30 %	36 %	22 %	32 %	100 %	29 %	20 %	27 %	17 %	17 %
rsa cryptopp	12 %	9 %	48 %	43 %	39 %	36 %	72 %	14 %	11 %	9 %	9 %	6 %	23 %	4 %	4 %
rsa openssl	22 %	19 %	47 %	47 %	42 %	38 %	62 %	28 %	23 %	17 %	20 %	11 %	91 %	8 %	8 %
md5 beecrypt	45 %	100 %	50 %	56 %	74 %	41 %	58 %	26 %	38 %	35 %	73 %	35 %	47 %	33 %	33 %
md5 cryptopp	11 %	22 %	74 %	76 %	100 %	72 %	57 %	14 %	15 %	13 %	23 %	10 %	30 %	7 %	7 %
md5 openssl	34 %	66 %	49 %	53 %	71 %	41 %	55 %	25 %	37 %	41 %	100 %	27 %	45 %	26 %	26 %

- AES does not have a unique set
- RC4 unique set only has two tuples

Implementations

Cell value > 70%

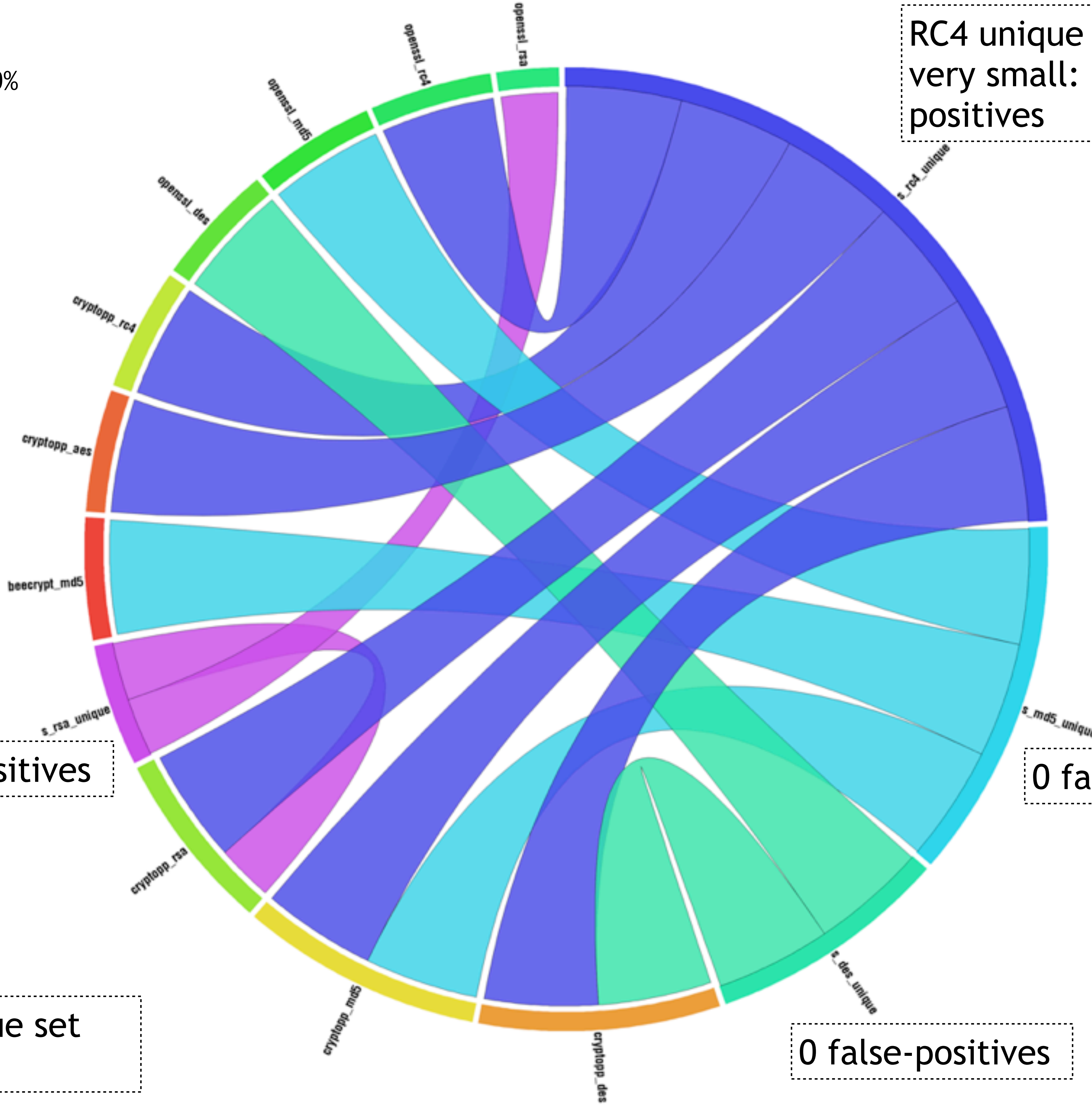


MD5 CryptoPP signature hits MD5 with 100% and with 70%<x<100% RC4, AES, DES

Beecrypt AES signature hits Beecrypt AES code (1 to 1)

Unique

Cell value > 70%



RC4 unique signature very small: three false-positives

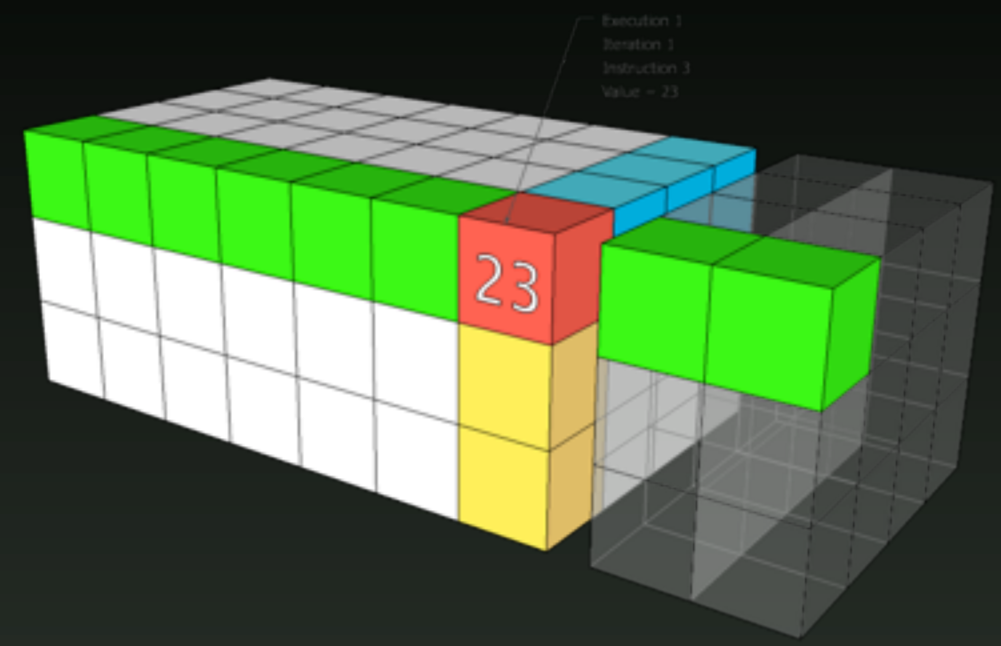
0 false-positives

0 false-positives

0 false-positives

No unique set for AES

Generic Method: Loop Differ

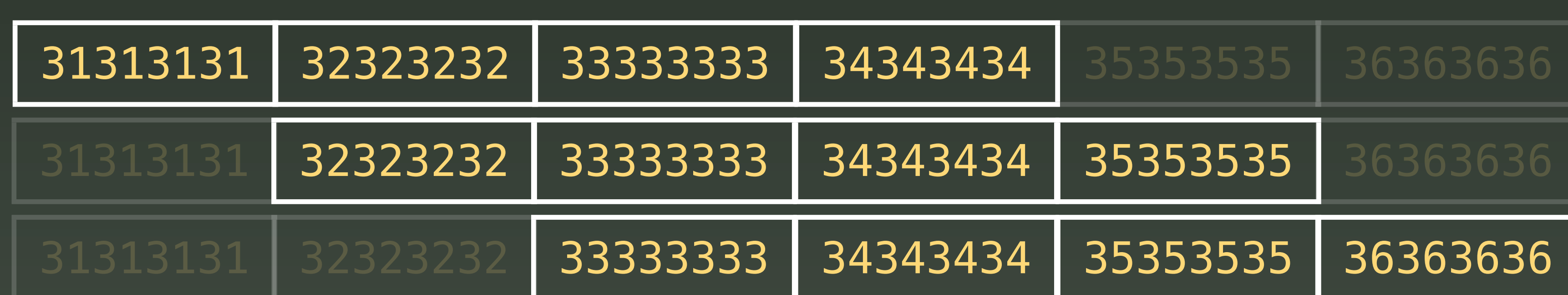


- Foreach Loop:
  - List of values for executions, iterations, instructions
  - Testing values for
    - XOR relation
    - Counter heuristic
    - S/P-box relation
    - Entropy heuristic
    - ...
- Evaluation:
  - Finds counters for almost all implementations
  - Finds XOR relation for most of the CFB/CBC mode symmetric ciphers

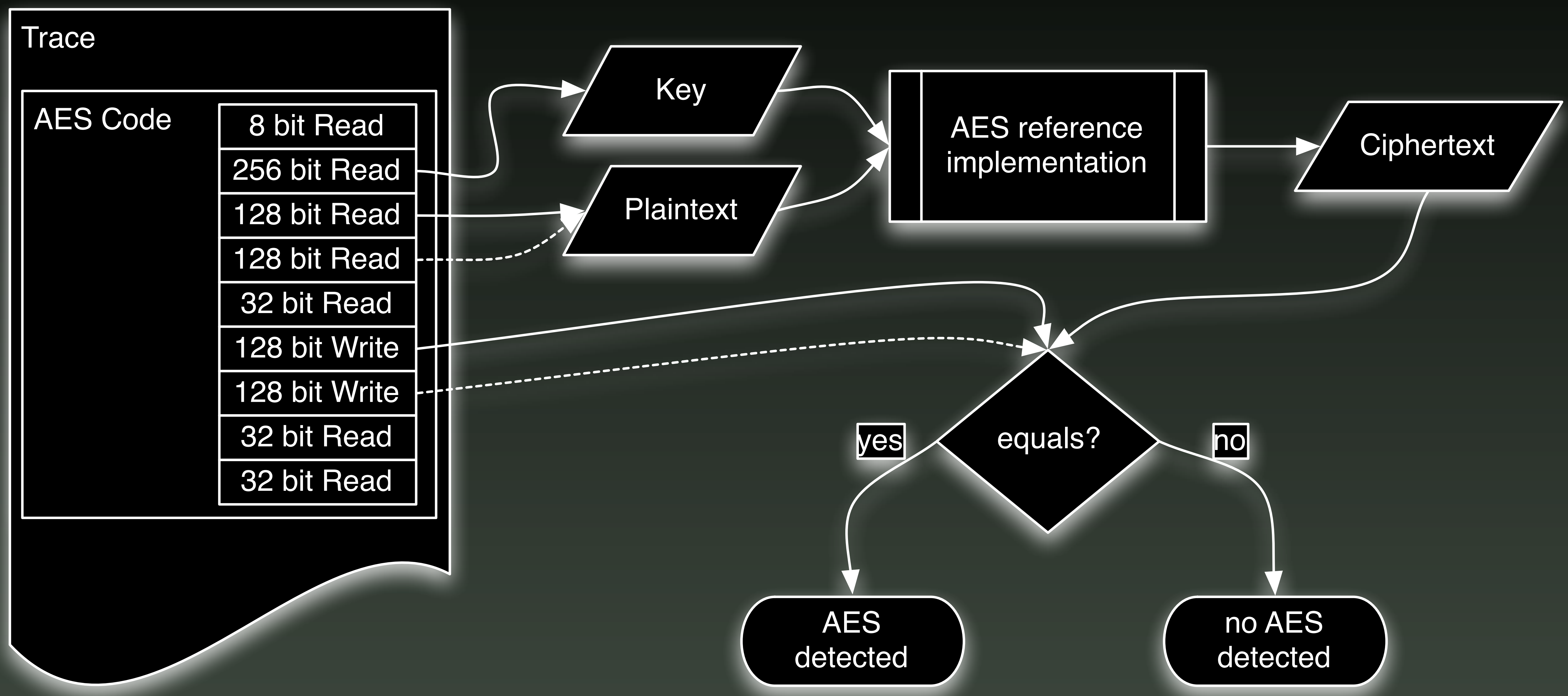
Generic Method: Data Verifier

- Use memory reconstruction to filter blocks above a specific size

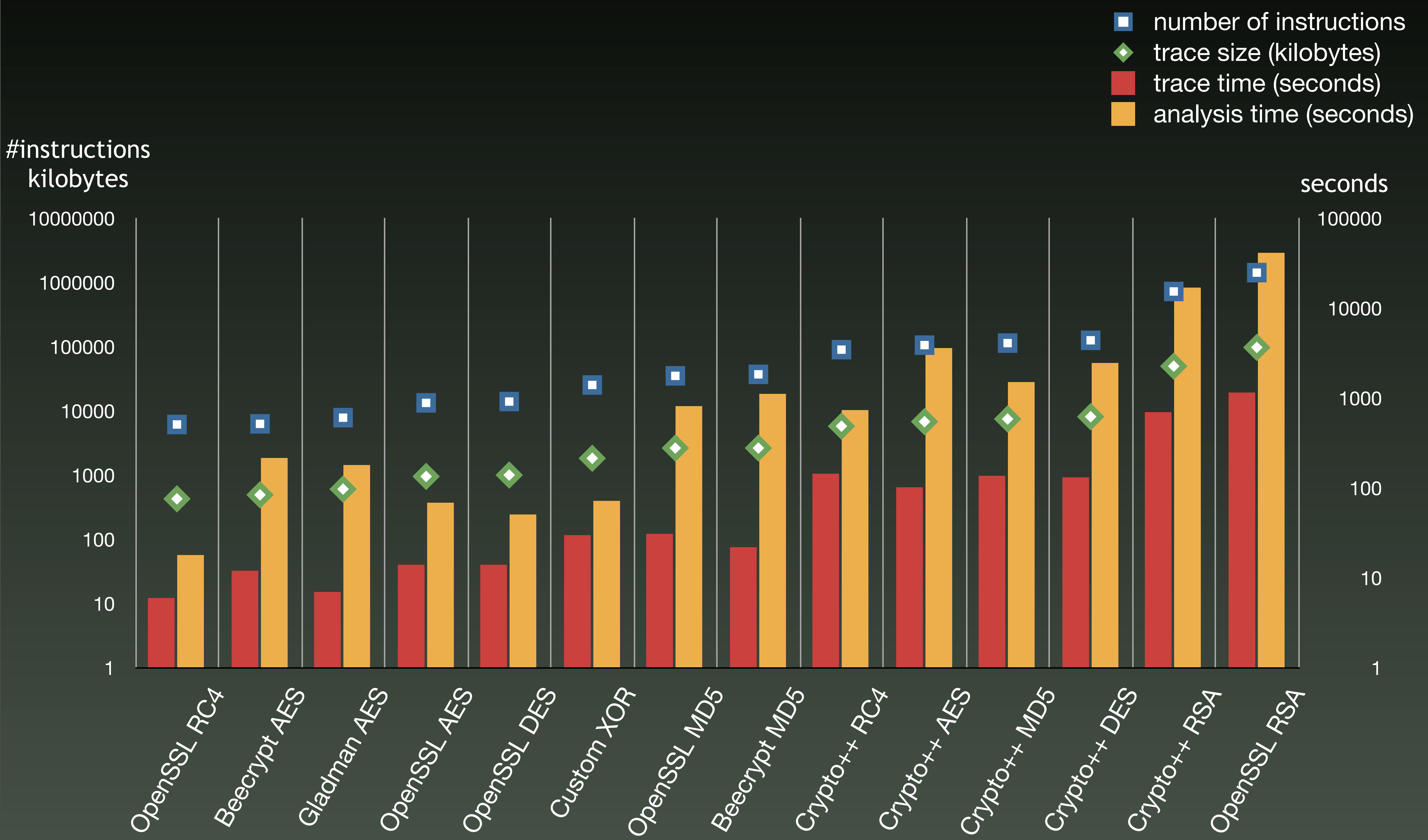
- Generate key, plaintext, ciphertext candidates of a specific size



Generic Method: Data Verifier



General Performance



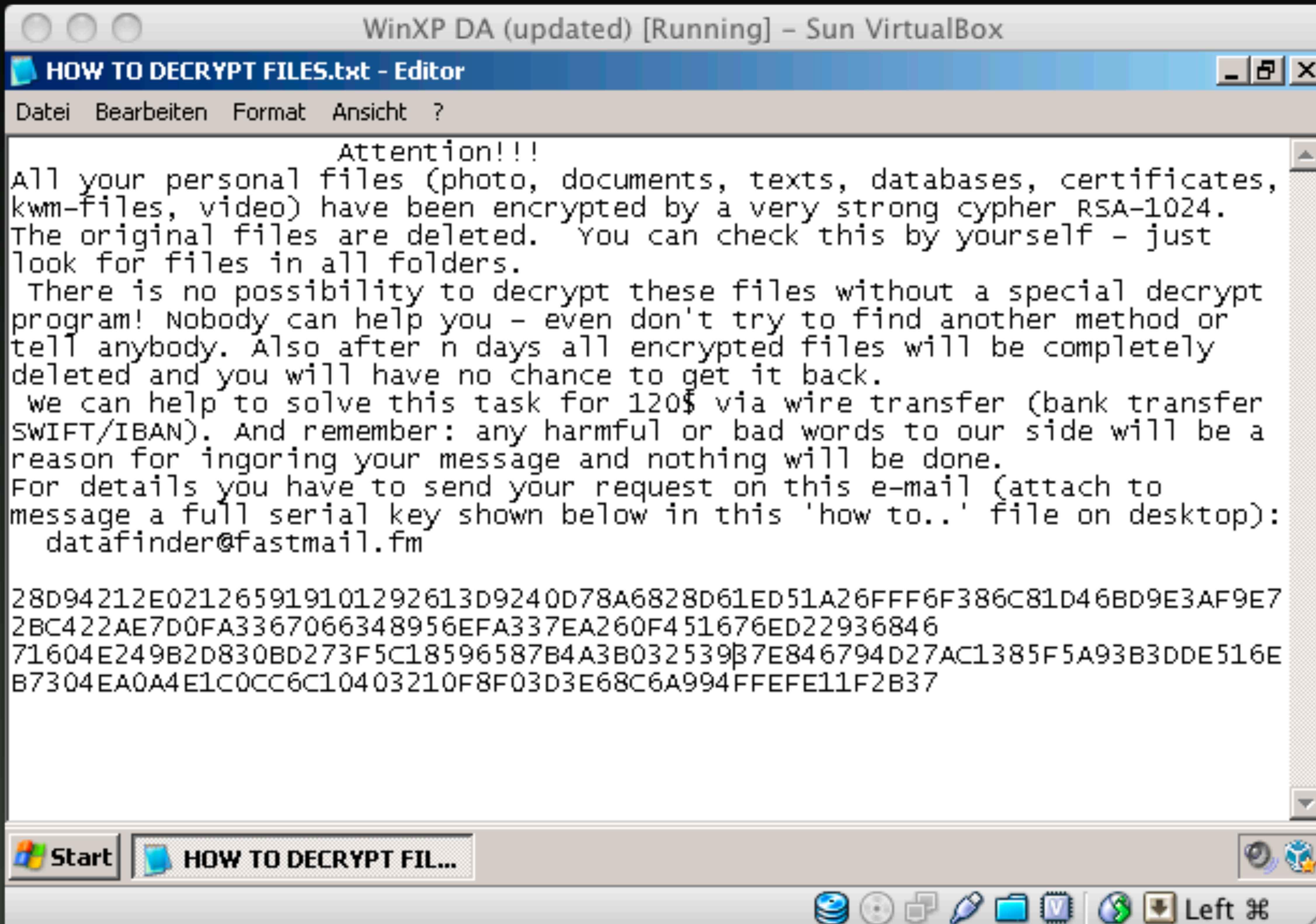
Real World Experiments

- Packed XOR testing application with ASPack 2.12
  - Trace size increased by factor 17; Analysis still found loops and xor
- curl HTTPS session: AES-256-CBC with OpenSSL 0.9.8l
  - Trace: 7 minutes, 45 MB; Analysis: 9 minutes
  - Identification of plain-, ciphertext and key in 94% of all blocks

Method	Results
`xorNotNullAndMov()`	only false-positives / unknown results
`symmetricCipherDataTester()`	detected 94% of AES instances including parameters
`loopDiffer()`	detected AES counters, some false-positives
`sigAPI()`	detected cryptographic functions
`constmemory()`	detected AES, one false-positive
`chains()`	detected AES and RSA, including implementation
`constmnemonic()`	detected AES implementation, one false-positive
`wang()`	no results
`caballero()`	detected core AES basic blocks
`lutz()`	detected core AES loops

GpCode Malware 2010

GpCode Malware 2010



GpCode Malware 2010

```
0000000: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000010: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000020: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000030: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000040: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000050: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000060: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000070: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333

0000000: ea0b e81c 2068 e7c4 89ac a21d 0298 7591  .... h........u.
0000010: ea0b e81c 2068 e7c4 89ac a21d 0298 7591  .... h........u.
0000020: ea0b e81c 2068 e7c4 89ac a21d 0298 7591  .... h........u.
0000030: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000040: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000050: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000060: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000070: 4444 4444 3333 3333 4444 4444 3333 3333  DDDD3333DDDD3333
0000080: 0300 0000                                ....

2010-12-05 20:48:17,080 Analysis.py:symmetricCipherDataTester@623 [DEBUG] found
aes 'DDDD3333DDDD3333'
'\xa8\xc2HM\xa21K\x96\x89\xce\x7fG\x13Q\xe1\xa4\x80\xf7:\xc1\x18\xe4u\x1f%\x85X
\x8c\xaa\xb6m\xda'
'\xea\x0b\xe8\x1c h\xe7\xc4\x89\xac\xa2\x1d\x02\x98u\x91'
```

Further Work

- Implement data relation checker in dedicated PIN tool or ptrace tool
  - ❖ Proof-of-Concept with Skype
- Reduce trace/analysis time and space requirements: Switch from PIN to BitBlaze
- Adopt machine-learning methods to signatures
- Research on detection of padding, compression, encoding and eventually be able to detect complete cryptographic composition

» http://code.google.com/p/kerckhoffs

Summary

- If you use a insecure cryptographic composition you fail (static key)
- Automatic identification of crypto code is feasible
- Applications of the proposed methods will find interesting results in malware, DRM systems and closed-source/obfuscated software
- Machine learning, dynamic binary trace systems and formats will help to further advance the described methods

» http://code.google.com/p/kerckhoffs
✆ felix@groebert.org

Thanks!

Questions?

special thanks: carsten willems & thorsten holz
greets: joernchen, aimster, buzze, d0mber, cw aka iceman, opti, sp, l1tt, foolabs, fluxfingers crew, gynvael, dfa, hawkes, jln, robert, taviso, liquid-k, asirap, novocainated, sirdarckcat, headhntr, greg, ths, max