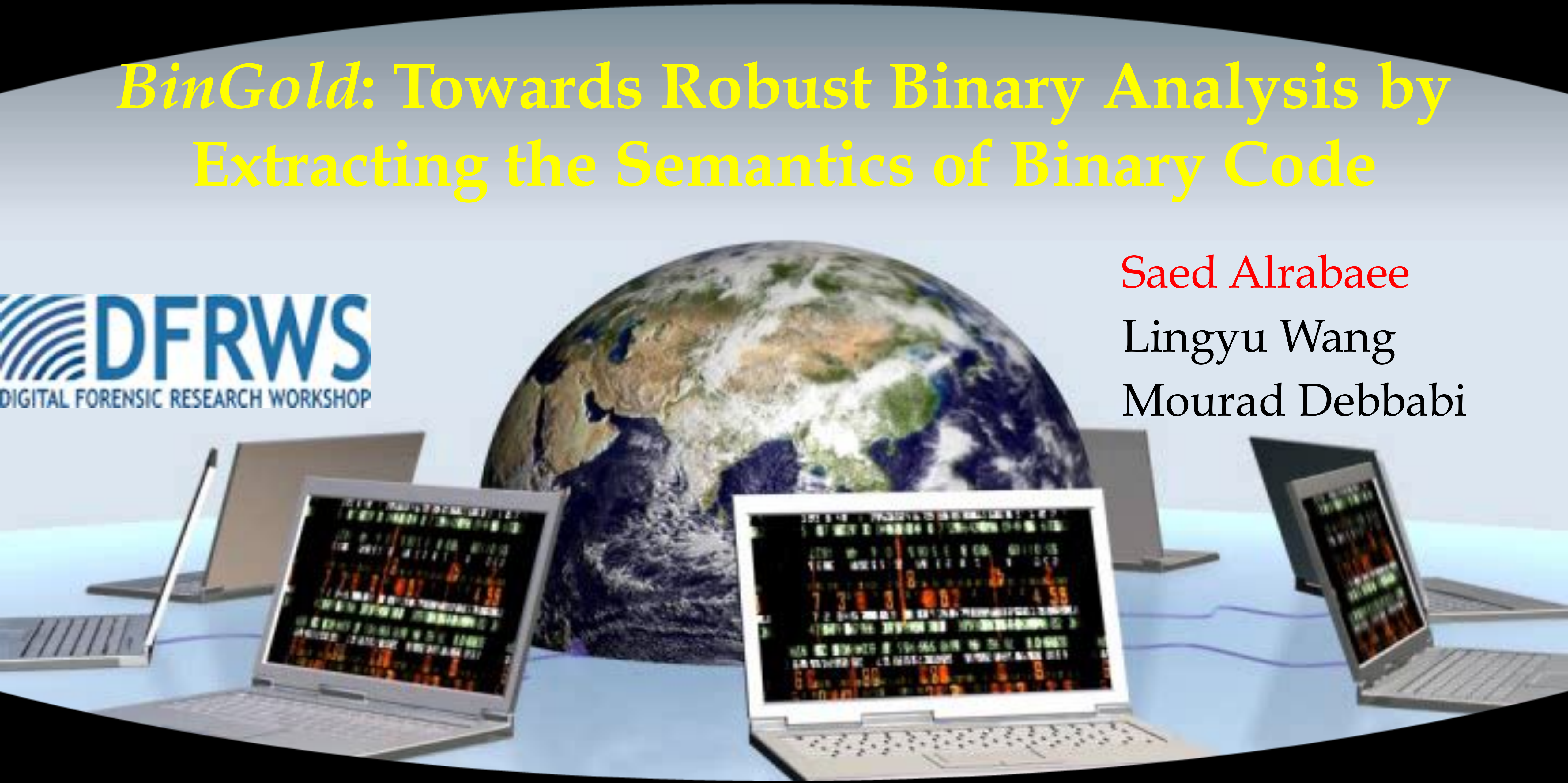
BinGold: Towards Robust Binary Analysis by Extracting the Semantics of Binary Code

Saed Alrabaee
Lingyu Wang
Mourad Debbabi

DFRWS
DIGITAL FORENSIC RESEARCH WORKSHOP



Introduction

- Binary analysis is useful in many practical applications
  - Detection of malware
  - Vulnerability analysis
  - Clone Detection
- Binary Code
  - Syntax Features
  - Semantic Features
  - Structural Features

Problem Overview

- Applying some techniques to evade existing works:
  - Light Obfuscation
  - Factoring Process
  - Source Compilers
  - Compilation Settings
- Applying such techniques:
  - Change the syntax of code
  - Change the structure of code
- As a result:
  - Leads to increase the rate of false positives
  - Affects the existing features

Background

- Function inlining:
  - The compiler may inline a small function into its caller code as an optimization

- Common Subexpression:
  - Remove redundant computations

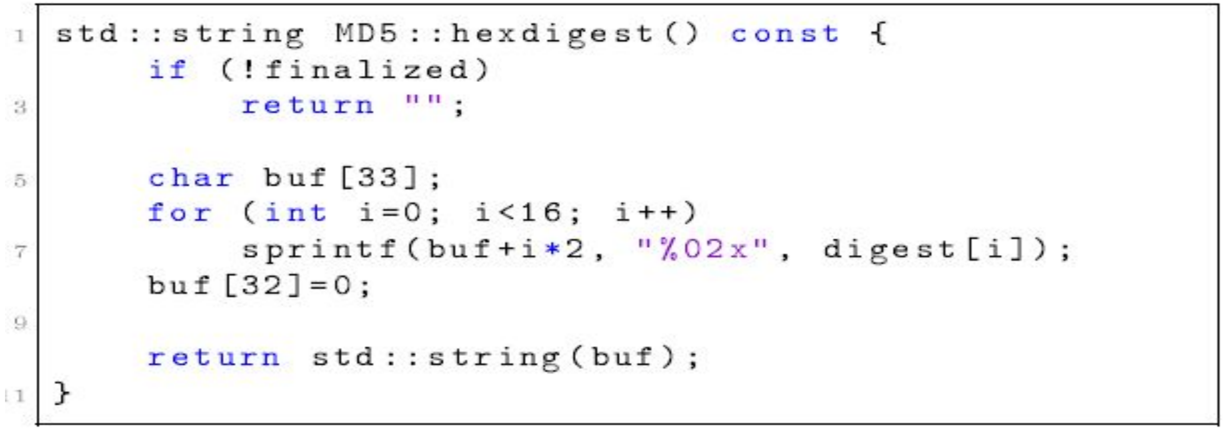
- Calling Conventions:
  - This specifies which registers are used for transferring parameters

Background (Cont'd)

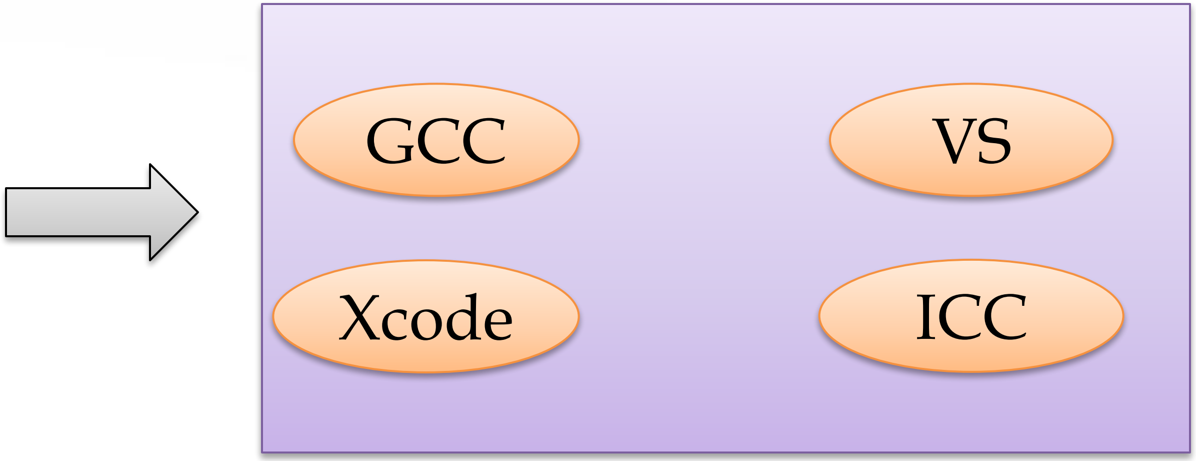
- Light Obfuscation:
  - Register renaming
  - Dead code
  - Instruction replacement
  - Instruction reordering

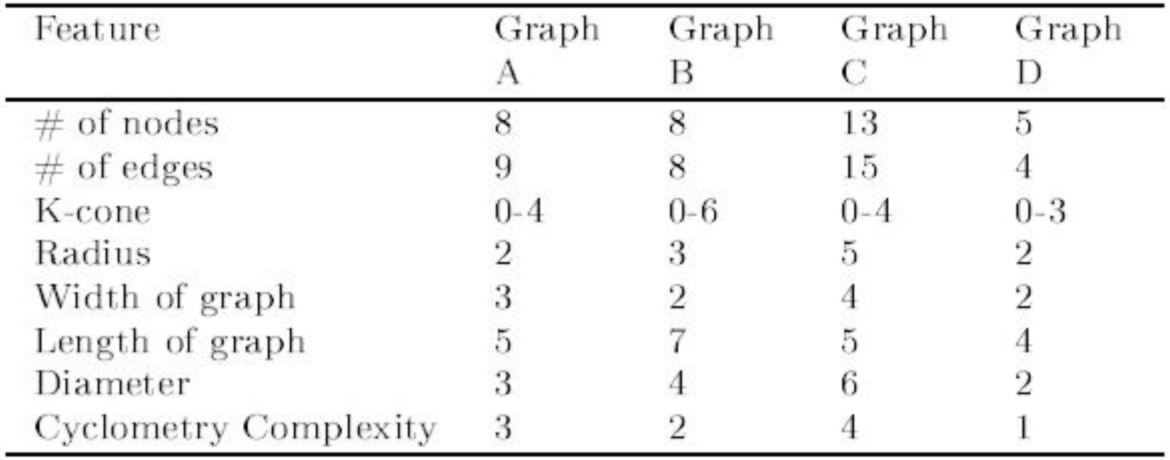
- Refactoring Process:
  - Variable renaming
  - Moving a method from a place to another place
  - Extracting a few statements and placing them into a new method

Motivation Example



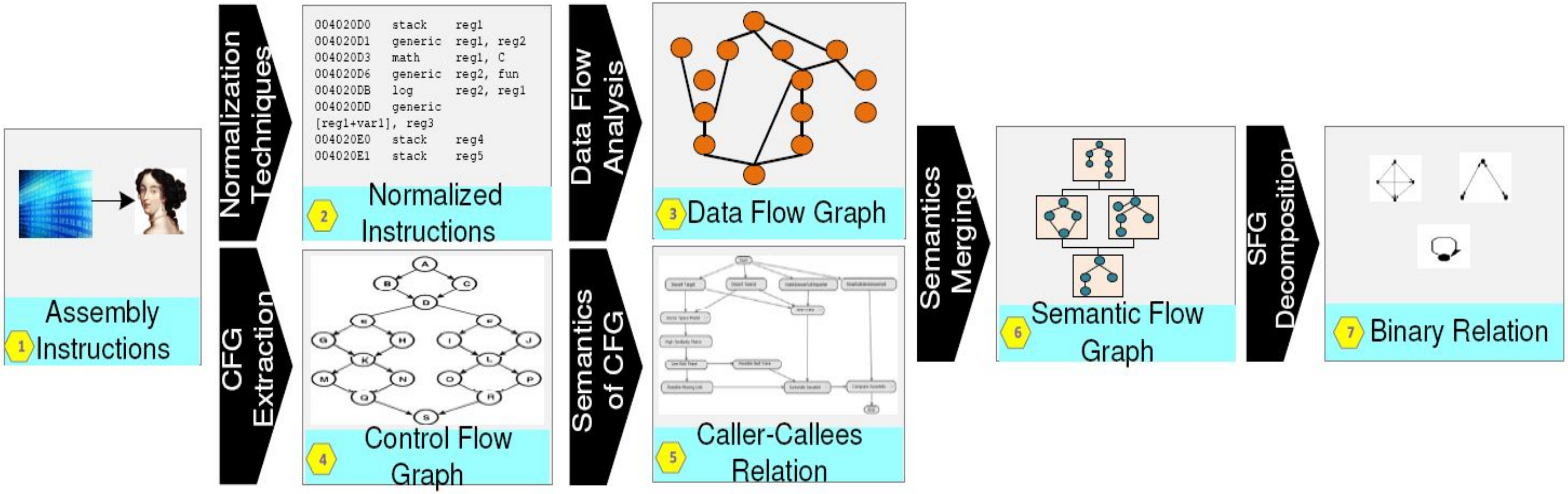
```
std::string MD5::hexdigest() const {
    if (!finalized)
        return "";

    char buf[33];
    for (int i=0; i<16; i++)
        sprintf(buf+i*2, "%02x", digest[i]);
    buf[32]=0;

    return std::string(buf);
}
```





Feature	Graph A	Graph B	Graph C	Graph D
# of nodes	8	8	13	5
# of edges	9	8	15	4
K-cone	0-4	0-6	0-4	0-3
Radius	2	3	5	2
Width of graph	3	2	4	2
Length of graph	5	7	5	4
Diameter	3	4	6	2
Cyclometry Complexity	3	2	4	1

Architecture Overview



```
004020D0   stack    reg1
004020D1   generic  reg1, reg2
004020D3   math     reg1, C
004020D6   generic  reg2, fun
004020DB   log      reg2, reg1
004020DD   generic
[reg1+var1], reg3
004020E0   stack    reg4
004020E1   stack    reg5
```

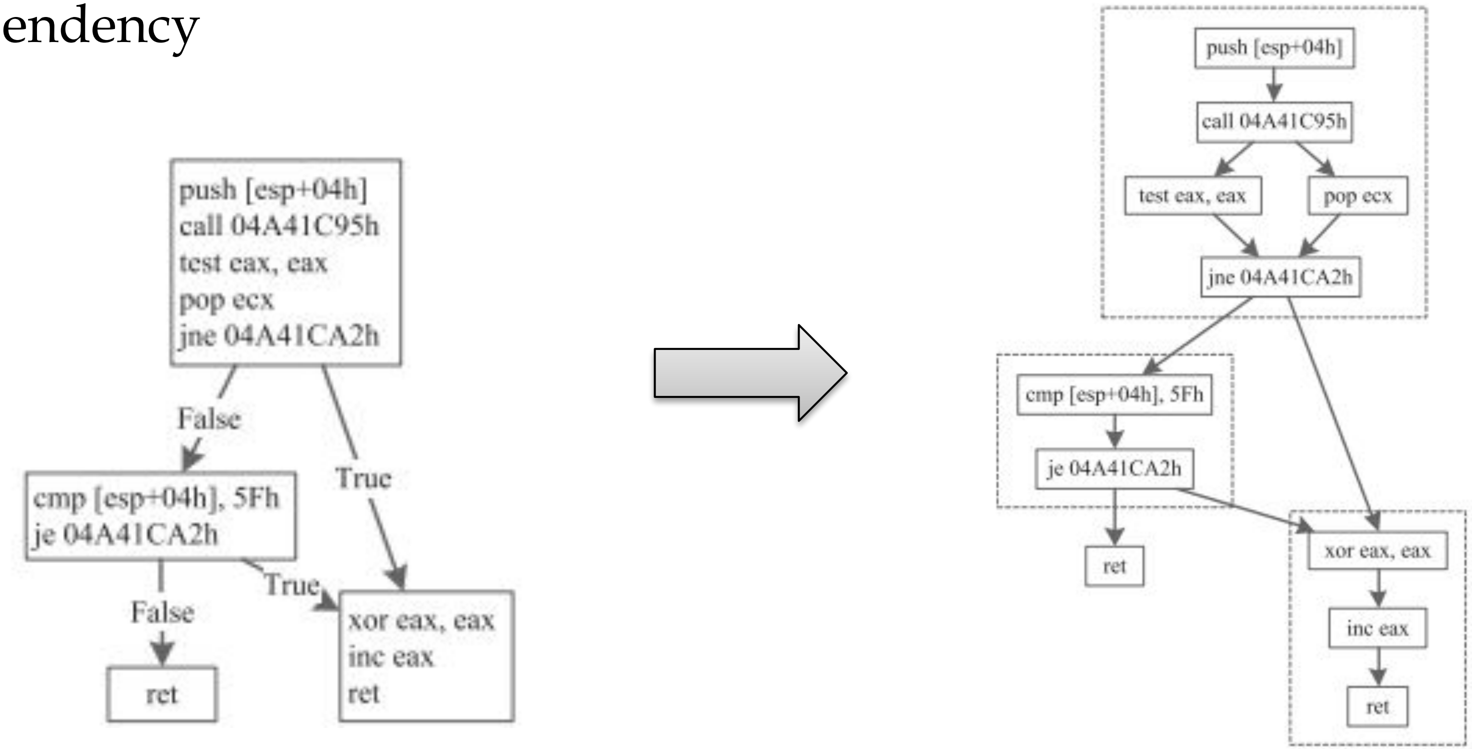
Methodology

- Normalization:
  - Generalize memory references
  - Registers
    - General registers: e.g., eax
    - Segment registers: e.g., cs
    - Index and pointer registers: e.g., esi
  - Constants

- Convert each instruction into three-tuple (g, c, d) :
  - g indicates the group that instruction belongs to
  - d represents the instruction opcode.
  - c represents the types of operands

Methodology (Cont’d)

- Data Flow Graph:
  - Internal dependency
  - Control dependency



```
push [esp+04h]
call 04A41C95h
test eax, eax
pop ecx
jne 04A41CA2h
```

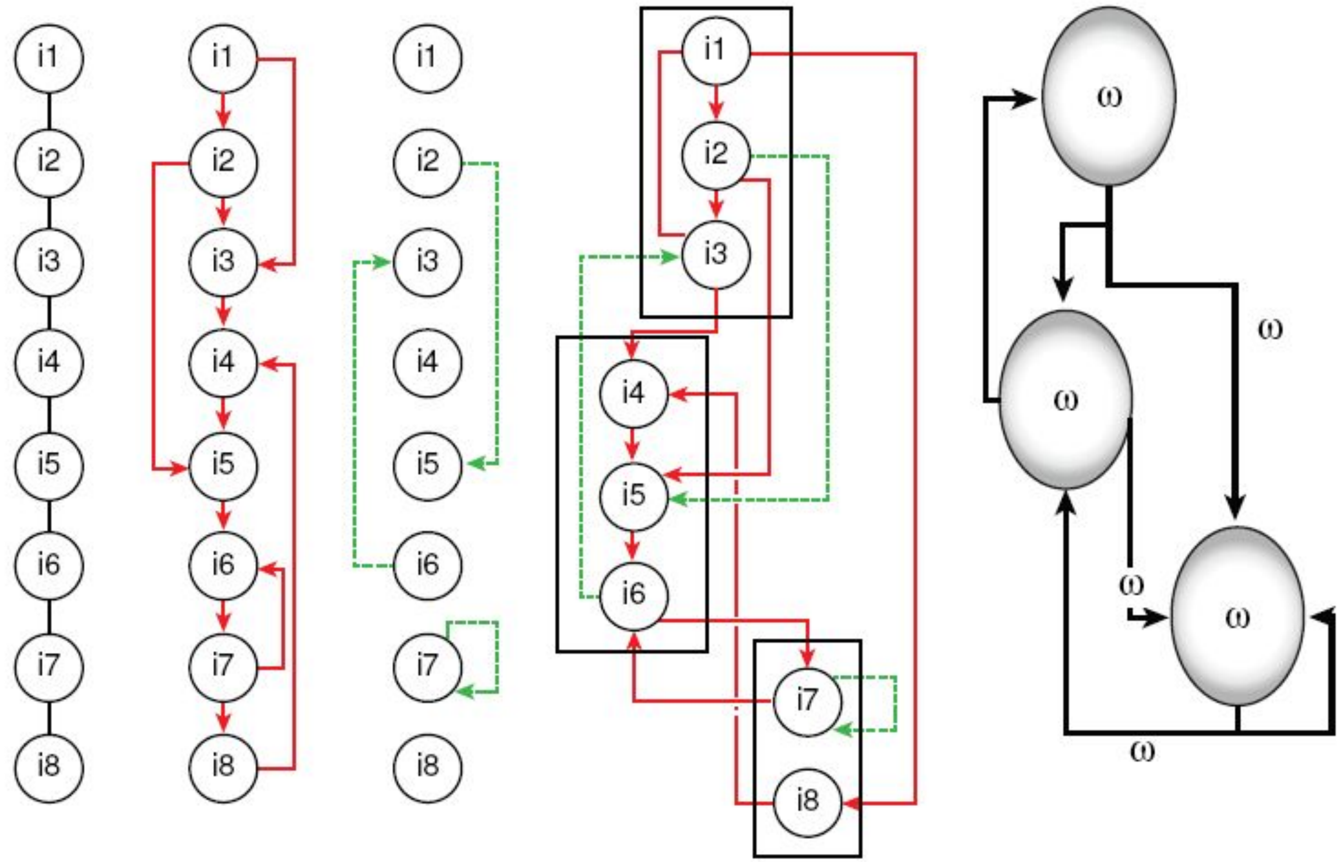
```
cmp [esp+04h], 5Fh
je 04A41CA2h
```

```
ret
```

```
xor eax, eax
inc eax
ret
```

Methodology (Cont’d)

- Semantic Flow Graph (SFG):

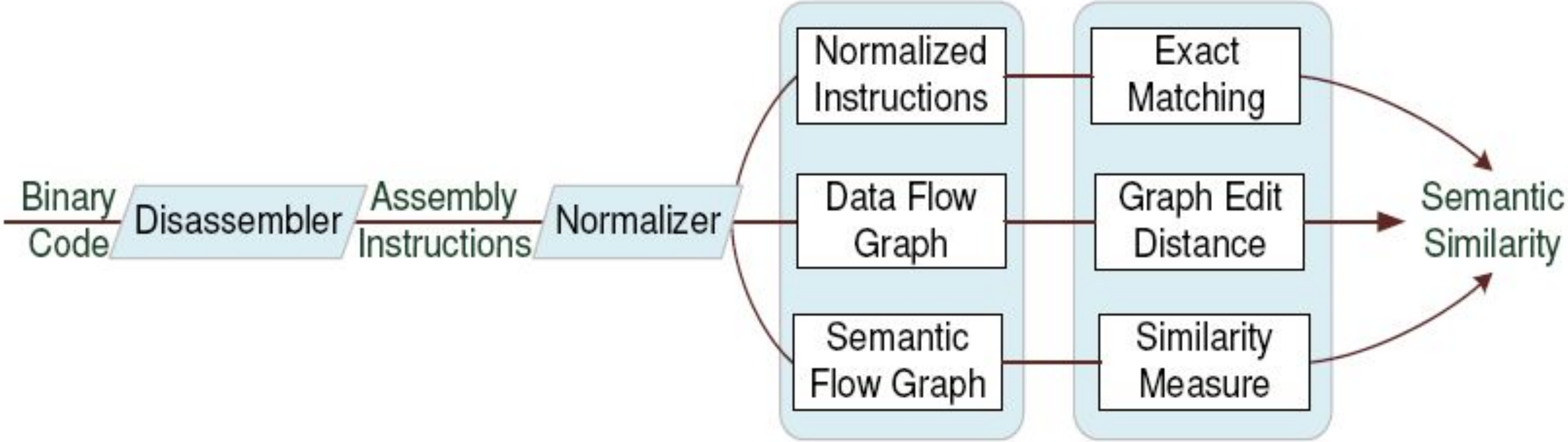


a) Normalized Instructions b) Data Flow c) Caller-Callee d) Data-Control Flow e) SFG

Methodology (Cont'd)

- SFG:
  - Reflexive
  - Symmetric
  - Antisymmetric
  - Transitive

Detection System



Evaluation

- 30 programs:
  - C++
  - C

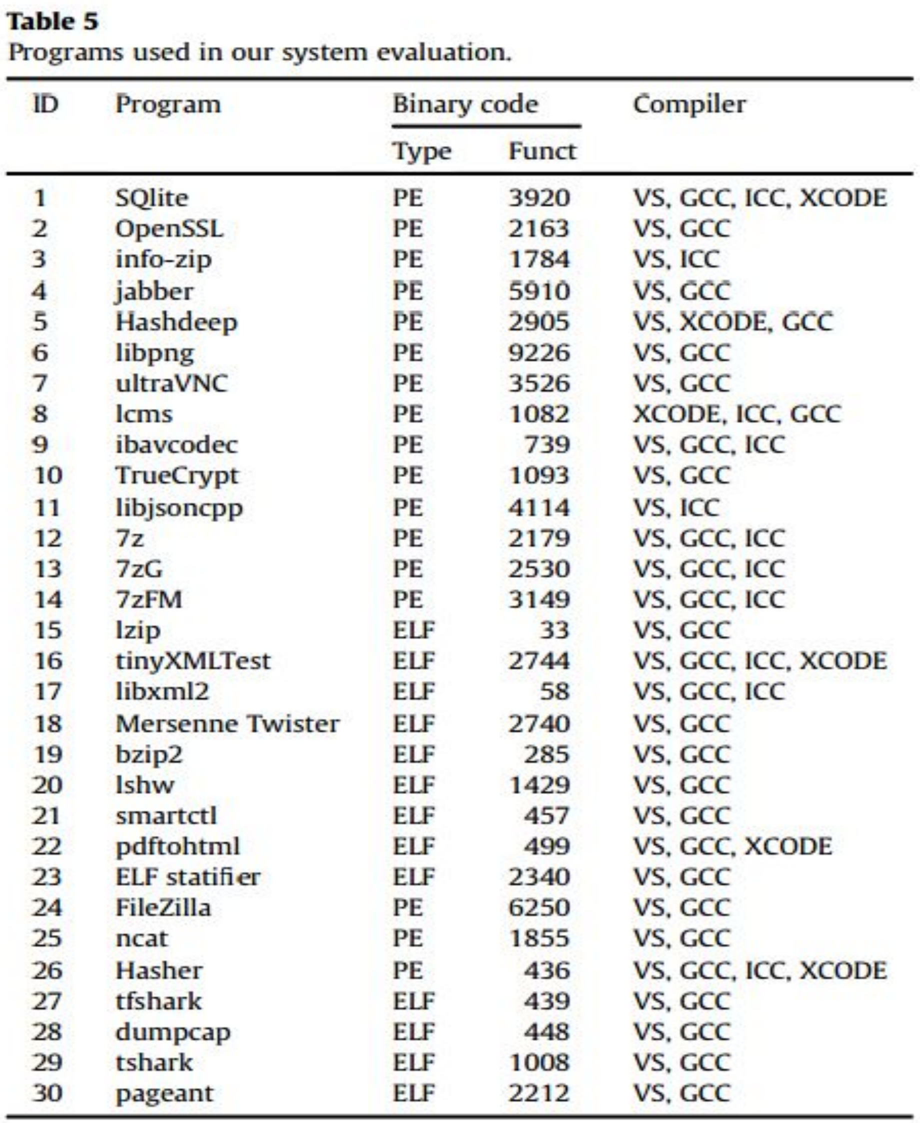


Table 5
Programs used in our system evaluation.

ID	Program	Binary code		Compiler
		Type	Funct	
1	SQlite	PE	3920	VS, GCC, ICC, XCODE
2	OpenSSL	PE	2163	VS, GCC
3	info-zip	PE	1784	VS, ICC
4	jabber	PE	5910	VS, GCC
5	Hashdeep	PE	2905	VS, XCODE, GCC
6	libpng	PE	9226	VS, GCC
7	ultraVNC	PE	3526	VS, GCC
8	lcms	PE	1082	XCODE, ICC, GCC
9	ibavcodec	PE	739	VS, GCC, ICC
10	TrueCrypt	PE	1093	VS, GCC
11	libjsoncpp	PE	4114	VS, ICC
12	7z	PE	2179	VS, GCC, ICC
13	7zG	PE	2530	VS, GCC, ICC
14	7zFM	PE	3149	VS, GCC, ICC
15	lzip	ELF	33	VS, GCC
16	tinyXMLTest	ELF	2744	VS, GCC, ICC, XCODE
17	libxml2	ELF	58	VS, GCC, ICC
18	Mersenne Twister	ELF	2740	VS, GCC
19	bzip2	ELF	285	VS, GCC
20	lshw	ELF	1429	VS, GCC
21	smartctl	ELF	457	VS, GCC
22	pdftohtml	ELF	499	VS, GCC, XCODE
23	ELF statifier	ELF	2340	VS, GCC
24	FileZilla	PE	6250	VS, GCC
25	ncat	PE	1855	VS, GCC
26	Hasher	PE	436	VS, GCC, ICC, XCODE
27	tfshark	ELF	439	VS, GCC
28	dumpcap	ELF	448	VS, GCC
29	tshark	ELF	1008	VS, GCC
30	pageant	ELF	2212	VS, GCC

Results

- F1 measure:
  - Similarity between binaries

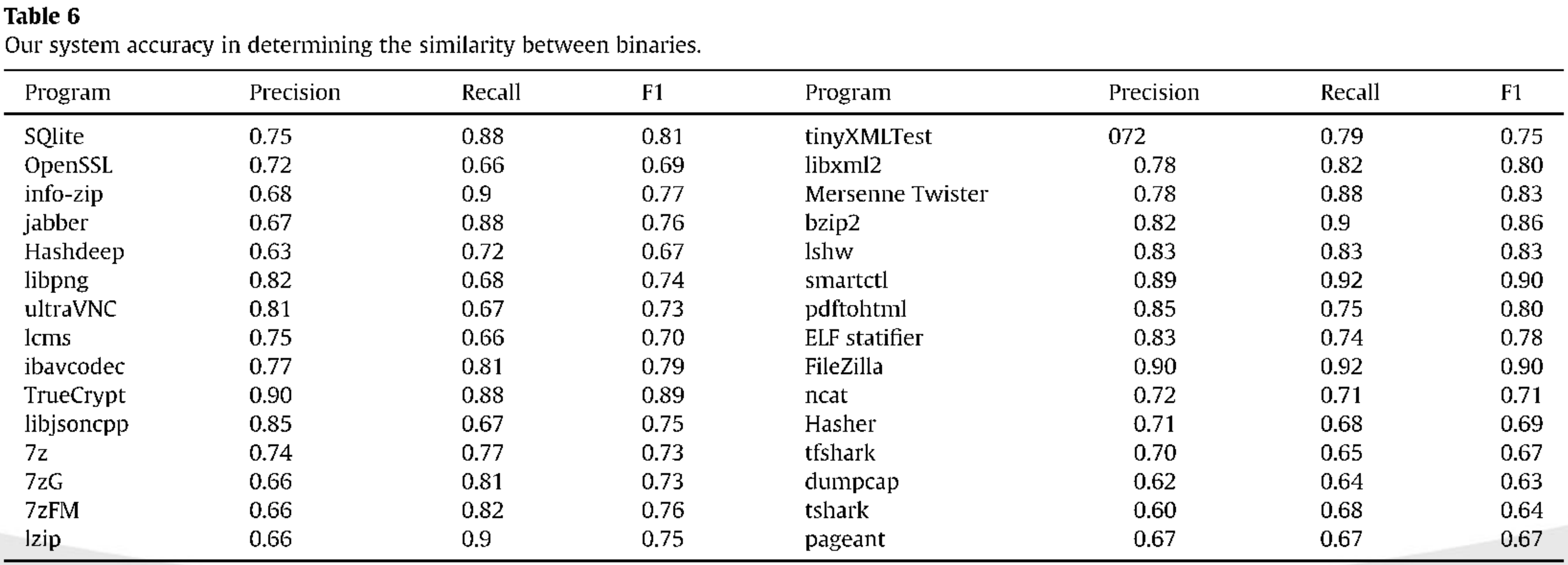


Table 6
Our system accuracy in determining the similarity between binaries.

Program	Precision	Recall	F1	Program	Precision	Recall	F1
SQlite	0.75	0.88	0.81	tinyXMLTest	072	0.79	0.75
OpenSSL	0.72	0.66	0.69	libxml2	0.78	0.82	0.80
info-zip	0.68	0.9	0.77	Mersenne Twister	0.78	0.88	0.83
jabber	0.67	0.88	0.76	bzip2	0.82	0.9	0.86
Hashdeep	0.63	0.72	0.67	lshw	0.83	0.83	0.83
libpng	0.82	0.68	0.74	smartctl	0.89	0.92	0.90
ultraVNC	0.81	0.67	0.73	pdftohtml	0.85	0.75	0.80
lcms	0.75	0.66	0.70	ELF statifier	0.83	0.74	0.78
ibavcodec	0.77	0.81	0.79	FileZilla	0.90	0.92	0.90
TrueCrypt	0.90	0.88	0.89	ncat	0.72	0.71	0.71
libjsoncpp	0.85	0.67	0.75	Hasher	0.71	0.68	0.69
7z	0.74	0.77	0.73	tfshark	0.70	0.65	0.67
7zG	0.66	0.81	0.73	dumpcap	0.62	0.64	0.63
7zFM	0.66	0.82	0.76	tshark	0.60	0.68	0.64
lzip	0.66	0.9	0.75	pageant	0.67	0.67	0.67

Applications

- Authorship Attribution
- Clone Detection

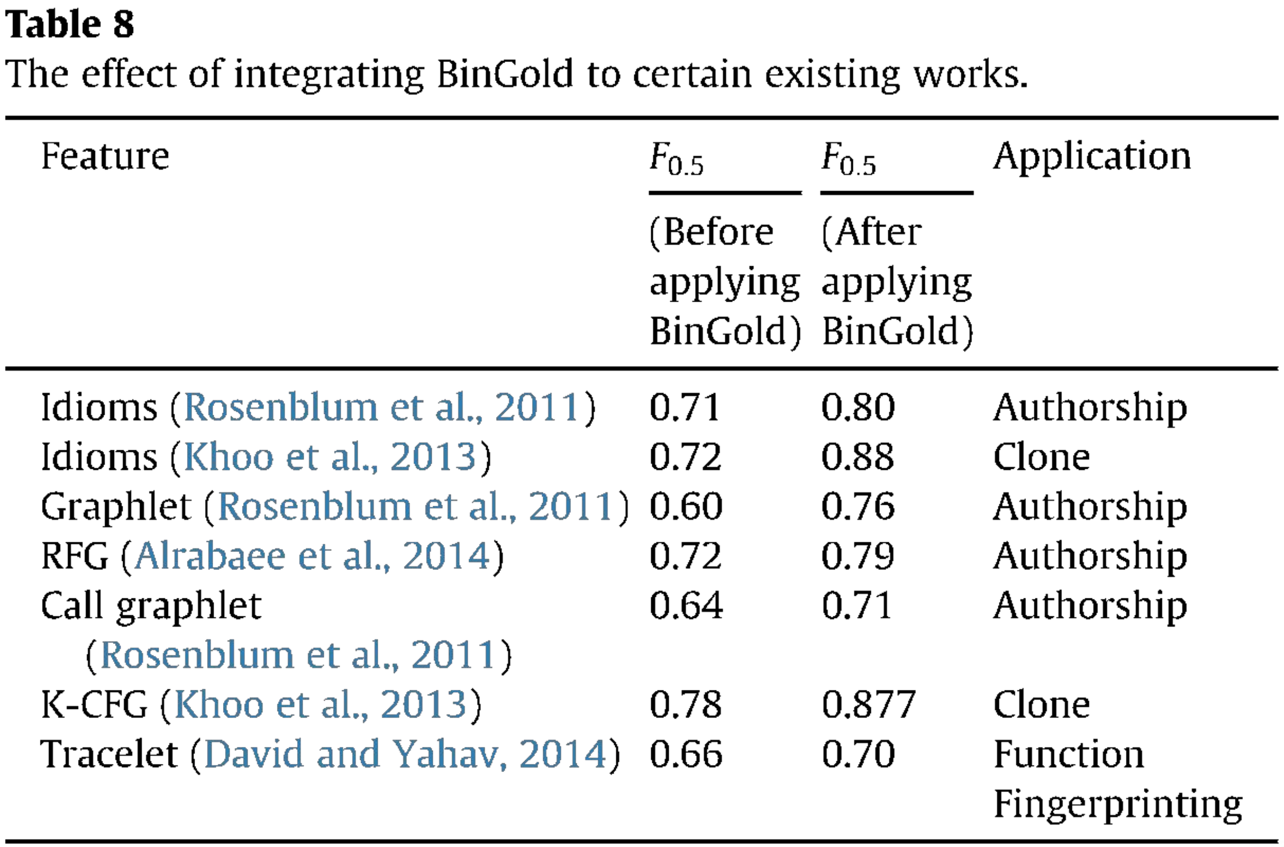


Table 8
The effect of integrating BinGold to certain existing works.

Feature	$F_{0.5}$	$F_{0.5}$	Application
	(Before applying BinGold)	(After applying BinGold)	
Idioms (Rosenblum et al., 2011)	0.71	0.80	Authorship
Idioms (Khoo et al., 2013)	0.72	0.88	Clone
Graphlet (Rosenblum et al., 2011)	0.60	0.76	Authorship
RFG (Alrabaee et al., 2014)	0.72	0.79	Authorship
Call graphlet (Rosenblum et al., 2011)	0.64	0.71	Authorship
K-CFG (Khoo et al., 2013)	0.78	0.877	Clone
Tracelet (David and Yahav, 2014)	0.66	0.70	Function Fingerprinting

Comparison

System	Compilers	Compilation settings	Refactoring tools	Source obfuscation	Binary obfuscation
BinSlayer	●	◐	◐	◐	●
Binjuice	●	◐	●	●	●
Bitshread	●	◐	●	●	●
BinDiff	◐	◐	◐	◐	◐
Reandavouz	●	●	●	●	●
BinLib	●	●	●	●	●
BinGold	○	○	○	○	○