# Developing a Deterministic Polymorphic Circuit Generator Using Boolean Logic Representation

UNIVERSITY OF SOUTH ALABAMA

Trinity L. Stroud, J. Todd McDonald, Todd R. Andel, and William Mahoney

tls1627@jagmail.southalabama.edu, {jtmcdonald,tandel}@southalabama.edu, wmahoney@unomaha.edu

University of Nebraska Omaha

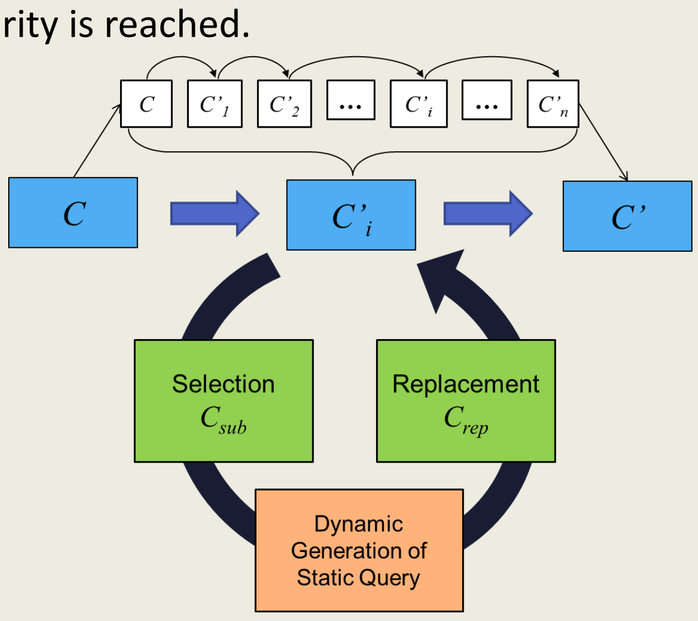
## ABSTRACT

Intellectual property (IP) is currently embedded in both software and hardware that are used in almost every area of society today. As companies can typically have billions of dollars invested in such IP, the theft of IP has become a major concern for tech companies and countries around the world. This research concerns efficient algorithms for generating polymorphic variants of programs represented as Boolean logic circuits, which can serve as a defense against adversarial reverse engineering and, ultimately, IP theft.

## MOTIVATION

An **iterative selection/replacement (ISR)** algorithm [1,2] selects small parts of a circuit (a selection of gates or *sub-circuit*) and then replaces those gates with a functionally equivalent sub-circuit. Doing this repetitively induces overlapping structural changes until some desired level of overhead/cost or security is reached.

**Replacement** can be accomplished using pre-generated static libraries, but it requires **factorial** generation time and storage: this limits ISR to only small selections. Our research proposes a **linear** time, dynamic algorithm using Boolean logic representation.




## RANDOM BOOLEAN LOGIC EXPANSION

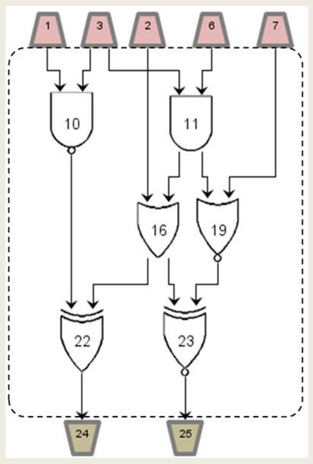
**BENCH Netlist**

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

10 = NAND(1, 3)
11 = AND(3, 6)
16 = OR(2, 11)
19 = NOR(11, 7)
22 = XOR(10, 16)
23 = NXOR(16, 19)
```

**Circuit Schematic**




**Boolean Expression**

```
o24 = ((i1 * i3)' ^ (i2 + (i3 * i6)))
o25 = ((i2 + (i3 * i6)) ^ ((i3 * i6) + i7)')'
```

**Circuit Representation**

| # | Original | | Expansion | Law | Relative Gates |
|---|---|---|---|---|---|
| 1 | 0 | = | A * 0 | Annihilation | CONST0 |
| 2 | 0 | = | A * A' | Complementation | CONST0 |
| 3 | 0 | = | A ^ A | Annihilation | CONST0 |
| 4 | 1 | = | A + 1 | Annihilation | CONST1 |
| 5 | 1 | = | A + A' | Complementation | CONST1 |
| 6 | 1 | = | (A ^ A)' | Annihilation | CONST1 |
| 7 | A | = | A * A | Idempotence | AND |
| 8 | A | = | A + A | Idempotence | OR |
| 9 | A | = | A * (A + B) | Absorption | AND,OR |
| 10 | A | = | A + (A * B) | Absorption | OR, AND |
| 11 | A | = | A + 0 | Identity | OR, CONST0 |
| 12 | A | = | A ^ 0 | Identity | XOR, CONST0 |
| 13 | A | = | A * 1 | Identity | AND, CONST1 |
| 14 | A | = | (A')' | Involution | NOT |
| 15 | A | = | (A * B') + (A * B) | Annihilation | AND,OR,NOT |
| 16 | A | = | (A + B) * (A + B') | Annihilation | AND,OR,NOT |
| 17 | A' | = | A ^ 1 | Negation | XOR, CONST1 |
| 18 | A' | = | (A' * B') + (A' * B) | Negation | AND,OR,NOT |
| 19 | A' | = | (A' + B) * (A' + B') | Negation | AND,OR,NOT |
| 20 | (A + B)' | = | A' * B' | De Morgan's | NOR |
| 21 | (A * B)' | = | A' + B' | De Morgan's | NAND |
| 22 | A ^ B | = | (A + B) * (A' + B') | Derivation | XOR |
| 23 | A ^ B | = | (A' * B) + (A * B') | Derivation | XOR |
| 24 | (A ^ B)' | = | (A + B)' + (A * B) | Negation | NXOR |
| 25 | A * (B + C) | = | (A * B) + (A * C) | Distributivity | AND,OR |
| 26 | A + (B * C) | = | (A + B) * (A + C) | Distributivity | OR,AND |
| 27 | (A * B) * C | = | A * (B * C) | Associativity | AND |
| 28 | (A + B) + C | = | A + (B + C) | Associativity | OR |

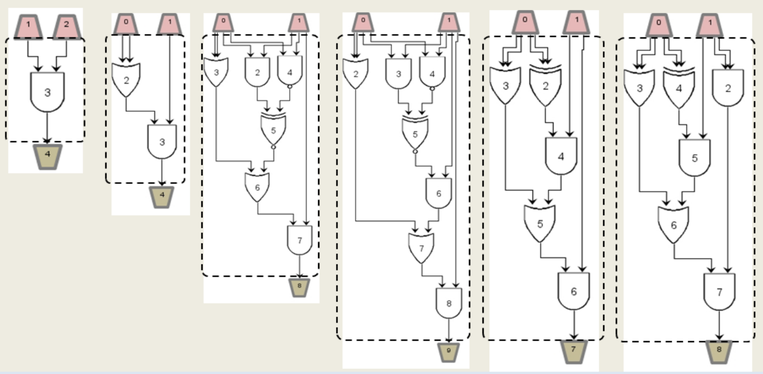
**Logic Expansion Rules**

Given a sub-circuit, we represent the structure as a Boolean expression. Boolean laws of logic reduction can be applied in reverse to expand the Boolean expression: we call this **Random Boolean Logic Expansion (RBLE)**. RBLE randomly selects a sub-expression and applies a random (if more than one exists) expansion from the set of Logic Expansion Rules.

**Example Expansion with Boolean Expression and Circuit Form**

```
o1 = (i0 * i1)                                   size(C)=1
1: ((i0+i0)* i1)                          rule 8, size(C')=2
2: (((i0+i0)+0)*i1)                      rule 11, size(C')=6
3: (((i0 + i0)+(i1*0))*i1)                rule 1, size(C')=7
4: (((i0 + i0)+(i1*(i0^i0)))*i1)          rule 3, size(C')=5
5: (((i0+i0)+(i1*(i0^i0)))*(i1*i1))       rule 7, size(C')=6
o1 = (((i0+i0)+(i1*(i0^i0)))*(i1*i1))            size(C')=6
```

C → C'



| Size: | 1 | 2 | 6 | 7 | 5 | 6 |
|---|---|---|---|---|---|---|
| Expansion: | 1 | 2 | 3 | 4 | 5 | |

## ALGORITHM AND METHODOLOGY

```
   input  : C,P,n
   output : C', where ∀x : C(x) = C'(x)
1  BE ← convert(C); done ← false;
2  fixed ← 0; attempts ← 0; numexp ← 0;
3  while not done do
4      expansions ← profile(BE);
5      expansion ← select(expansions);
6      BE ← apply(BE, expansion);
7      Ĉ ← realize(BE);
8      if P == FIXED then
9          fixed ← fixed + 1;
10         if fixed = n then
11             C' ← Ĉ; done ← true;
12         end
13     else
14         if (P == STRICTSIZE and size(Ĉ) = n) then
15             C' ← Ĉ; done ← true;
16         else if (P == TARGETSIZE and size(Ĉ) >= n) then
17             C' ← Ĉ; done ← true;
18         else
19             numexp ← numexp + 1;
20             if numexp > MAXEXPANSIONS then
21                 BE ← convert(C); z ← 0;
22                 attempts ← attempts + 1;
23                 if attempts > MAXATTEMPTS then
24                     C' ← null; done ← true;
25                 end
26             end
27         end
28     end
29 end
30 return C';
```

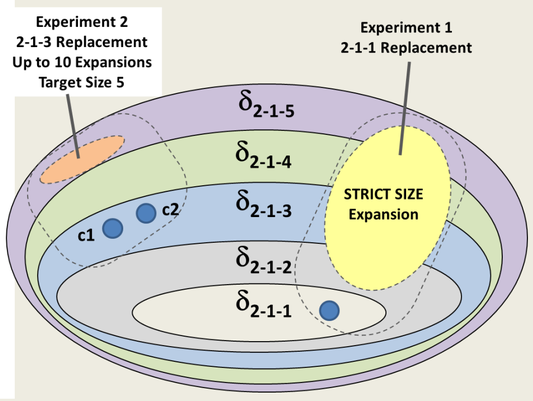
C: Circuit
P: Expansion Policy
n: Policy Value
C': Circuit Variant

### Expansion Policies:

- Fixed
  *n* is # of expansions
- Strict Size
  *n* is = # of gates
- Target Size
  *n* is >= # of gates

### Experiments:

- Compare circuits **chosen** from static libraries vs circuits **expanded** by RBLE
- Determine if RBLE produces **uniform distribution** of random variants

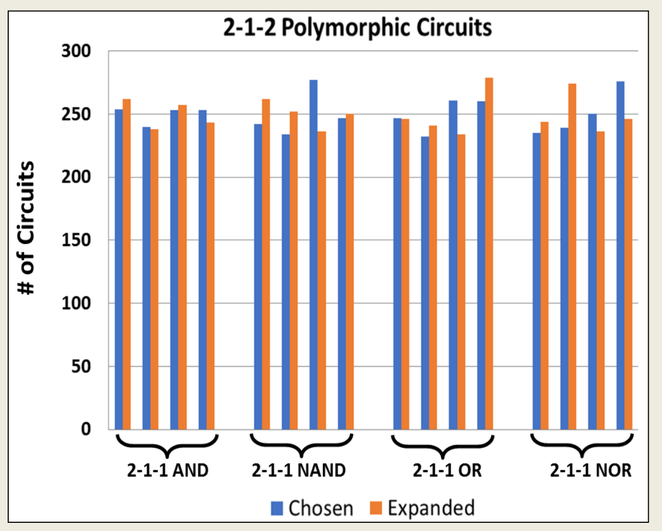


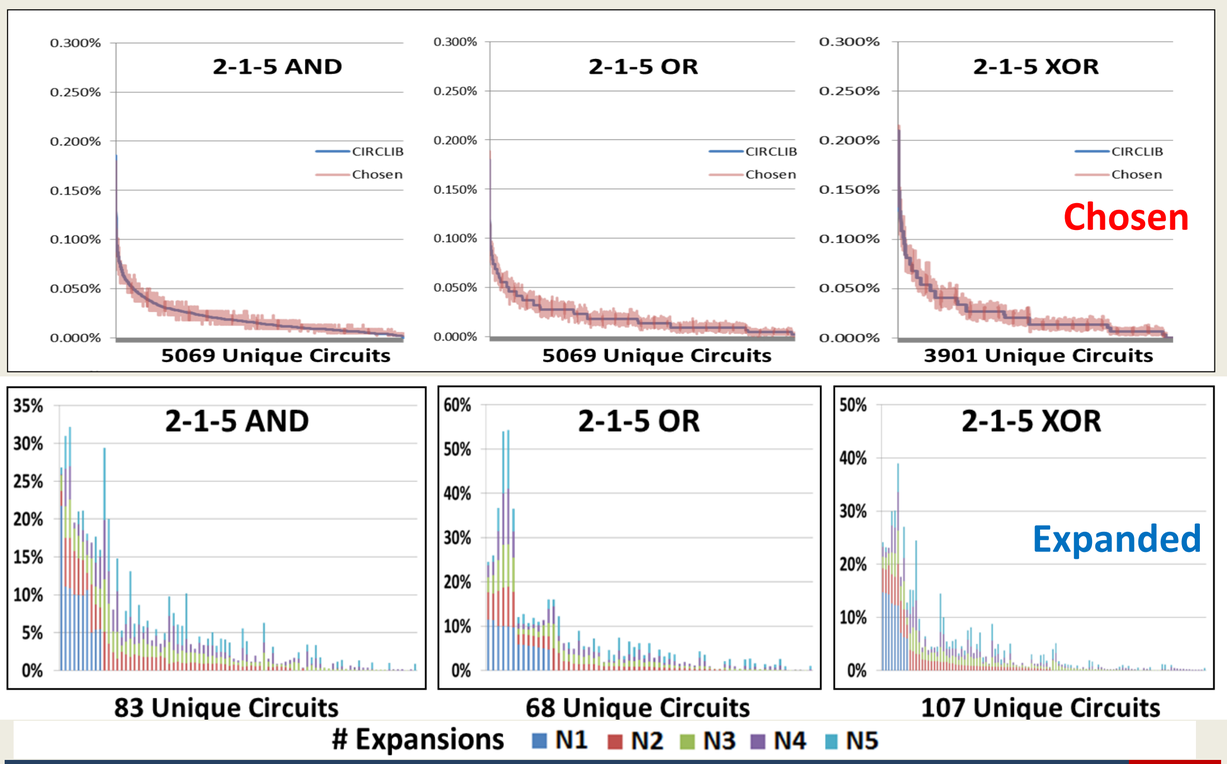

## ANALYSIS AND SAMPLE RESULTS

We generate and analyze replacement circuits by choosing from pre-generated static libraries and RBLE expansion. The goal is to determine whether RBLE creates circuit distributions consistent with static choices, which are uniformly random. Generated/studied 13,360,000 circuit variants as semantically equivalent replacements for simple δ2−1−1 and δ2−1−3 circuits

### Key Results:

- RBLE exhibited instances of **uniformity** under **strict size** expansion policy
- RBLE can only reach a **small % of comparable circuits** uniformly vs. chosen static library selection under **fixed** expansion policy




## REFERENCES

[1] J. McDonald, Yong Kim, and Daniel Koranek. 2011. Deterministic circuit variation for anti-tamper applications. *Proceedings of the Cyber Security and Information Intelligence Research Workshop (CSIIRW '11)*, October 12-14, 2011, Oak Ridge, TN, USA. DOI: 10.1145/2179298. 2179376.

[2] J. McDonald and Yong C. Kim. 2011. Examining Tradeoffs for Hardware-Based Intellectual Property Protection. 2012. *Proceedings of the 7th International Conference on Information Warfare (ICIW-2012)*, March 22-23, 2012, University of Washington, Seattle, USA.

## ACKNOWLEDGEMENTS

This work is partially funded by National Science Foundation award 1811578 in the NSF 17-576 Secure and Trustworthy Cyberspace (SaTC) program

The 4th NSF Secure and Trustworthy Cyberspace Principal Investigator Meeting (2019 SaTC PI Meeting), October 28-29, 2019 | Alexandria, Virginia