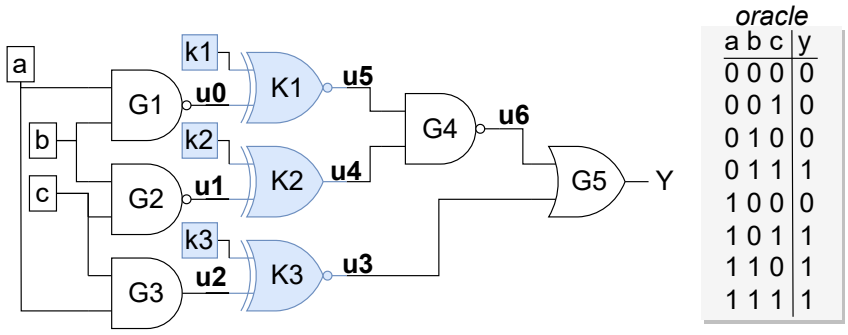

Fig. 9. Logic locked example design with its "Oracle" as a truth table

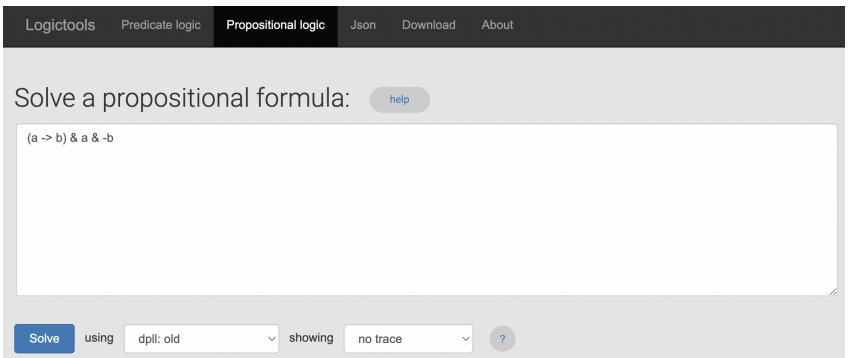

Fig. 10. The web browser-based logictools SAT solver from [40]

To begin, let us construct a miter circuit using two copies of the locked design. For readability, we will label the internal nets of the first copy as shown in Fig. 9 and represent the second copy's nets with "w". We distinguish between copy 1's keys and copy 2's keys by appending 'a' and 'b', respectively. The miter circuit is shown as a formula in Fig. 11.

circuit copy 1

```
((~(a&b) <=> u0) & (~(b&c) <=> u1) & ((c&a) <=> u2) & (~(k1a+u0) <=> u5)
& ((k2a + u1) <=> u4) & (~(k3a+u2) <=> u3) & (~(u5&u4) <=> u6) &
(y <=> (u6 | u3)))
```

&

circuit copy 2

```
((~(a&b) <=> w0) & (~(b&c) <=> w1) & ((c&a) <=> w2) & (~(k1b+w0) <=> w5)
& ((k2b + w1) <=> w4) & (~(k3b+w2) <=> w3) & (~(u5&w4) <=> w6) &
(yx <=> (w6 | w3)))
```

output compare

```
&
(y + yx)
```

Fig. 11. Miter circuit as a formula: SAT Attack Iteration #1

When we solve using `dpll:old`, we receive the following feedback:

```
Clause set is true if we assign values to variables as: u0 a -b u1 c
    u2 u5 k1a -u4 k2a u3 k3a u6 y w0 w1 w2 w5 k1b w4
-k2b -w3 -k3b -w6 -yx
```

In other words, the miter circuit formula is **satisfiable**! Of particular interest is the value of the inputs that the solver has found, being $a = 1$, $b = 0$ (see -b in the feedback), and $c = 1$. This is a distinguishing input pattern. In this case, circuit copy 1 produces an output of 1, while circuit copy 2 produces an output of 0. We can check the Oracle to see which is the intended output; from Fig. 9, the correct output should be 1. As we can see in line 6 of Algorithm 1, we should make a new formula by adding more copies of the circuit but with the constraint that the output produced with the DIP of $a = 1, b = 0, c = 1$ should be 1. The formula for the next iteration of the SAT attack thus looks like Fig. 12.

```
miter circuit from before  &
circuit copy 3  ((~(aa&bb) <=> uu0) & (~(bb&cc) <=> uu1) & ((cc&aa) <=> uu2) &
                (~(k1a+uu0) <=> uu5) & ((k2a + uu1) <=> uu4) & (~(k3a+uu2) <=> uu3) &
                (~(uu5&uu4) <=> uu6) &
                (yy <=> (uu6 | uu3)))
                &
                ((~(aa&bb) <=> ww0) & (~(bb&cc) <=> ww1) & ((cc&aa) <=> ww2) &
circuit copy 4  (~(k1b+ww0) <=> ww5) & ((k2b + ww1) <=> ww4) & (~(k3b+ww2) <=> ww3) &
                (~(uu5&ww4) <=> ww6) &
                (yy <=> (ww6 | ww3)))
                &
                   (aa &
correct input/output  ~bb &
                cc &
                yy )
```

Fig. 12. SAT Attack Iteration #2

We add the miter circuit formula from iteration #1 to new copies of the circuit. Note the new internal node names "uuX" and "wwX" to represent the next copy, as well as "aa", "bb", "cc", and "yy" for the DIP and corresponding output from the oracle. Note also that the variables representing the key (e.g., k1a, k1b) are the same as those in the initial miter circuit formula; this forces keys found in future iterations to make the circuit produce the correct output for the DIP. Once again, using the SAT solver produces the following feedback:

```
Clause set is true if we assign values to variables as: u0 -a b -u1 c
     -u2 u5 k1a u4 k2a -u3 k3a -u6 -y w0 -w1 -w2 w5
k1b w4 k2b w3 -k3b -w6 yx uu0 aa -bb uu1 cc uu2 uu5 -uu4 uu3 uu6 yy
    ww0 ww1 ww2 ww5 -ww4 -ww3 ww6
```

This time, the DIP that is found is $a = 0, b = 1, c = 1$, which the Oracles tells us should produce the output 1. We can do yet another iteration of the SAT attack, adding Fig. 13 to our growing formula.

Running the SAT solver, we find that the formula is still satisifiable, with the following feedback:

```
Clause set is true if we assign values to variables as: u0 -a b u1 -c -u2
    u5 k1a -u4 k2a u3 -k3a u6 y w0 w1 -w2 w5 k1b w4 -k2b -w3 k3b -w6 -yx
    uu0 aa -bb uu1 cc uu2 uu5 -uu4 -uu3 uu6 yy ww0 ww1 ww2 ww5 ww4 ww3 -
   ww6 uuu0 -aaa bbb -uuu1 ccc -uuu2 uuu5 uuu4 uuu3 -uuu6 yyy www0 -www1
    -www2 www5 -www4 -www3 www6
```

If we construct the new formula for another iteration and run that through the solver (see the Appendix for the full formula), we finally receive the following feedback:

```
Clause set is false for all possible assignments to variables.
```

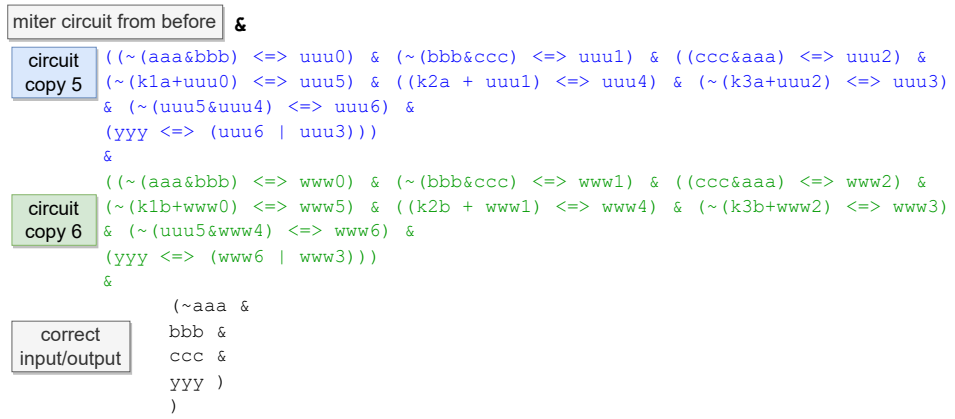

Fig. 13. SAT Attack Iteration #3

The formula is **unsatisfiable**! This means that the solver can no longer find any DIPs; in other words, all the remaining inputs and keys will make the two copies of the circuit behave equivalently. Because the formula also adds constraints that any remaining keys make the circuit produce the correct input/output behavior (from the previous DIPs), any key that satisfies the formula as a whole, without the `y + yx` constraint, should be a correct key. Thus, running the formula with that constraint missing, gives us:

```
Clause set is true if we assign values to variables as: -u0 a b
-u1 c u2 -u5 k1a -u4 -k2a u3 k3a u6 y -w0 -w1 w2 -w5 k1b -w4 -k2b w3 k3b
    w6 yx uu0 aa -bb uu1 cc uu2 uu5 uu4 uu3 -uu6 yy
ww0 ww1 ww2 ww5 ww4 ww3 -ww6 uuu0 -aaa bbb -uuu1 ccc -uuu2 uuu5
-uuu4 -uuu3 uuu6 yyy www0 -www1 -www2 www5 -www4 -www3 www6 uuuu0 -aaaa
    bbbb uuuu1 -cccc -uuuu2 uuuu5 uuuu4 -uuuu3 -uuuu6 -yyyy wwww0 wwww1 -
    wwww2 wwww5 wwww4 -wwww3 -wwww6
```

If we hone in on the variables representing the key bits, we can see that $k1a = k1b = 1, k2a = k2b = 0, k3a = k3b = 1$.

4.5 Discussion

The arrival of the "SAT attack" marked a turning point in the logic locking domain. Recent survey works such as that by Chakraborty et al. [5] provide a good overview of the various techniques which you should now be able to better appreciate. Several post-SAT techniques try to reduce the number of keys that are pruned with each iteration of the SAT attack (e.g., [47]), while others try to introduce circuit structures that cause issues for SAT solvers (e.g., [31]). New defense techniques are proposed and countered, even now, as the "cat-and-mouse" game continues. Other recent work includes the proposal of FPGA-based redaction [19] or universal circuits [4], among other strategies.

While this tutorial covered the first formulation of the SAT attack from Subramanyan et al. [39], there are a few more things we should note. As we mentioned in subsubsection 4.1.2, this tutorial focused on a combinational design, so you are probably interested to know what happens in more realistic systems, where we have memory elements and sequential logic. In the early days of logic locking, the notion of Oracle access is often paired with the idea of a fully-scanned design with an adversary-accessible scan chain (as you explored in the previous case study in Section 3). With design-for-test structures like the scan-chain, an adversary can treat the different parts