universität freiburg

# Hierarchical Stochastic SAT and Quality Assessment of Logic Locking

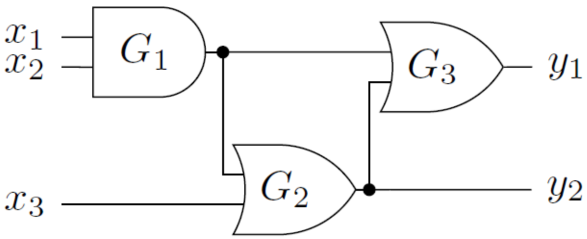
**Alpine Verification Meeting 2024, originally at SAT'24**

Christoph Scholl, **Tobias Seufert**, Fabian Siegwolf

Department of Computer Science
University of Freiburg
Germany

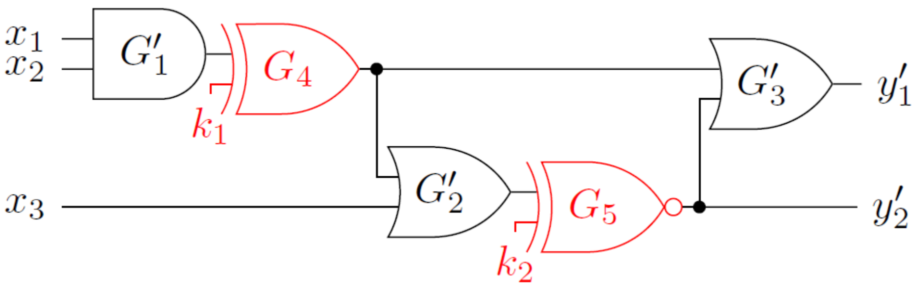
# 1. Introduction to Logic Locking

- Logic Locking protects Integrated Circuits (ICs) from **unauthorized usage (e.g., overproduction from untrusted foundry)**

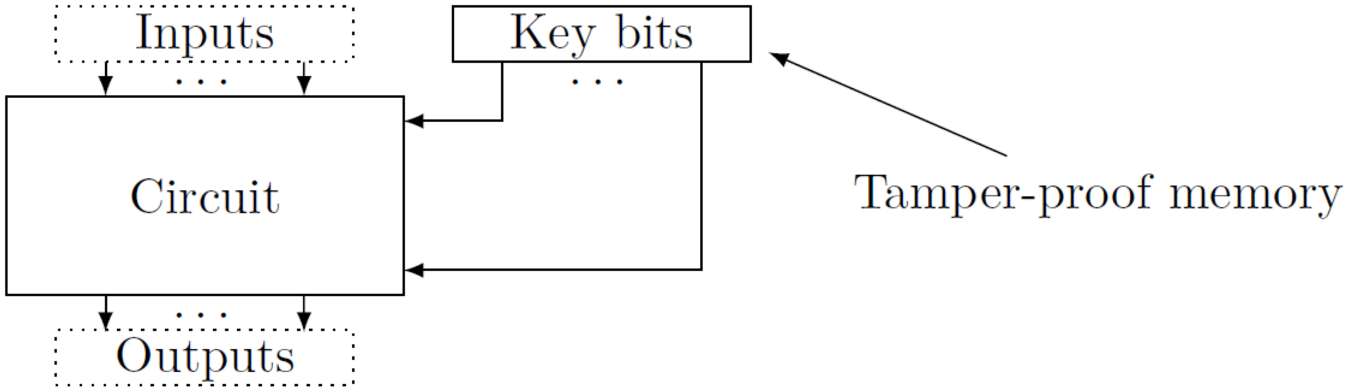



Original circuit

$\Rightarrow$




„Locked“ circuit



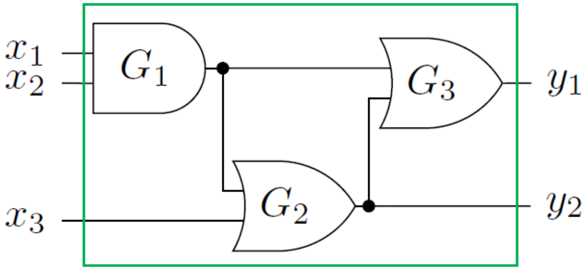
# 2. Quality Assessment of Logic Locking

- **Goal: Attacks** against logic locking should be **impossible** (or too expensive to be realistic)
- **Attacker model**:
  - Attacker has only access to the locked ICs
  - Attacker can buy an unlocked IC on the market
  - Has to find out the key by „trial-and-error“
- **Possible weaknesses** of logic locking methods:
  - Not only one key is unlocking, but several keys (many, a large fraction?)
  - There are many key patterns which are not completely correct, but „almost“, since they produce correct outputs for „almost all“ input patterns

$\Rightarrow$ **Quality measures** for logic locking and **precise quality assessment** using **formal methods!**

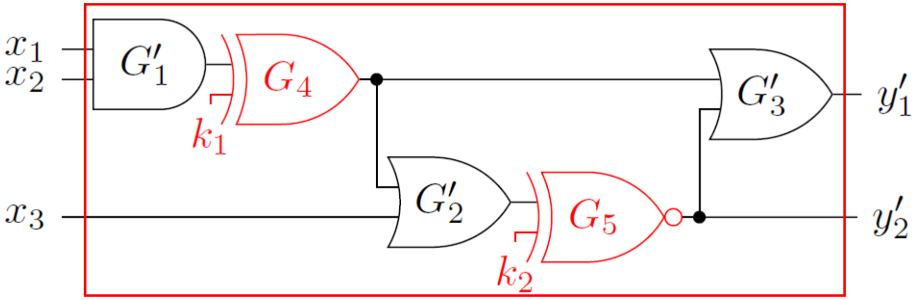
# 3. Reduction to Existing SAT-related Problems

## Inverted Miter Circuit



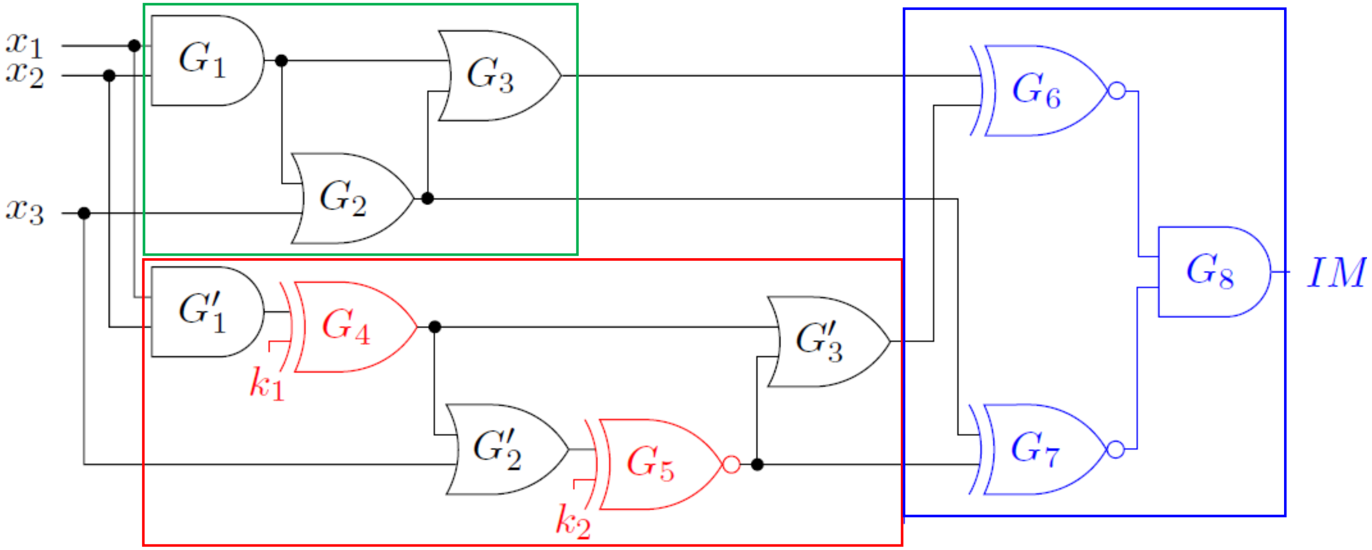

Original circuit

+




„Locked“ circuit

⇓




Inverted Miter Circuit

# 3. Reduction to Existing SAT-related Problems
## Quality Measures

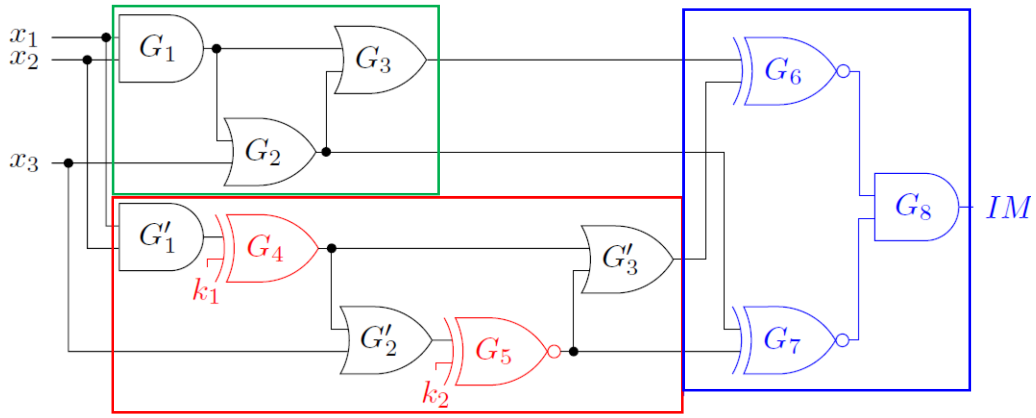
- A **key** is called **unlocking**, if it computes the correct outputs for all possible inputs.

- Check whether there exists an unlocking key: $\exists \vec{K} \forall \vec{X}: f_{IM}(\vec{X}, \vec{K})$

- Check whether there exists a key different from the original (intended) unlocking key $\vec{K}_{orig}$ that unlocks the circuit:

$$\exists \vec{K} \forall \vec{X}: [f_{IM} \wedge (\vec{K} \neq \vec{K}_{orig})]$$

- Compute the **fraction** of unlocking keys: $\text{Ɐ}^{0.5} \vec{K} \forall \vec{X}: f_{IM}$ (can be reformulated as projected model counting)



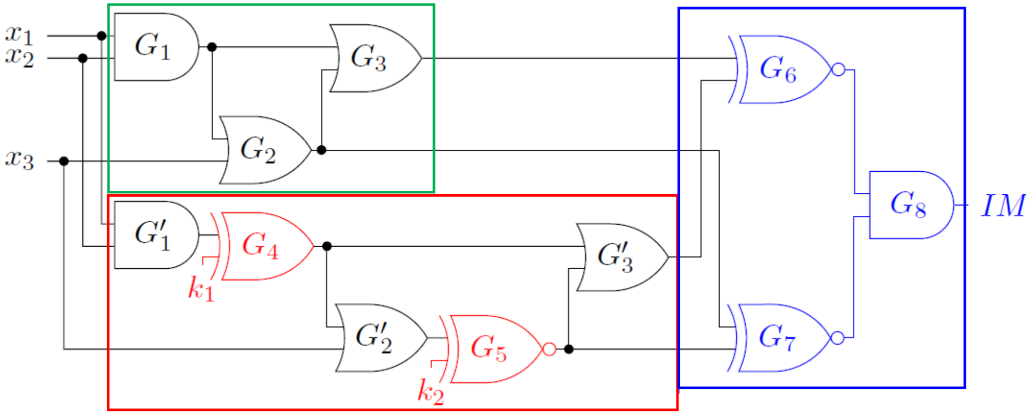
# 3. Reduction to Existing SAT-related Problems
## Quality Measure: Existence of Keys with High Criticality

- What if a key is „almost unlocking“?
- **Def.**: The **criticality of a key** is defined as the quotient of the number of input assignments for which the key produces a correct output and the total number of input assignments.

⇒ We do not want to have keys different from $\vec{K}_{orig}$ that have a criticality higher than $c$ (close to 1).

- Check whether such a key exists:

$$\left(\exists \vec{K}\, \text{Я}^{0.5} \vec{X}: [f_{IM} \wedge (\vec{K} \neq \vec{K}_{orig})]\right) > c$$




⇒ Stochastic SAT (**SSAT**)

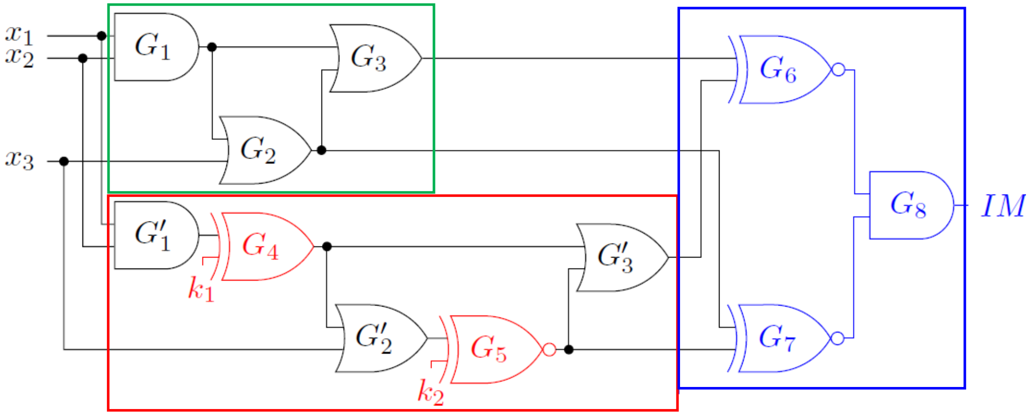
# 3. Reduction to Existing SAT-related Problems
## Quality Measure: Average Criticality of Keys

- A few keys with high criticality are not too bad …

$\Rightarrow$ Compute the average criticality of all keys:

$$ᴚ^{0.5}\vec{K}\,ᴚ^{0.5}\vec{X}: f_{IM}$$

$\Rightarrow$ **Model Counting**




- But is this what we actually want to compute?
- Two examples with average criticality $\approx$ 0.5:
  - Case 1: The original key has criticality 1, all others criticality 0.5 $\Rightarrow$ **no security problem**
  - Case 2: One half of the keys has criticality 1, the other half has criticality 0 $\Rightarrow$ **severe security problem**

# 3. Reduction to Existing SAT-related Problems

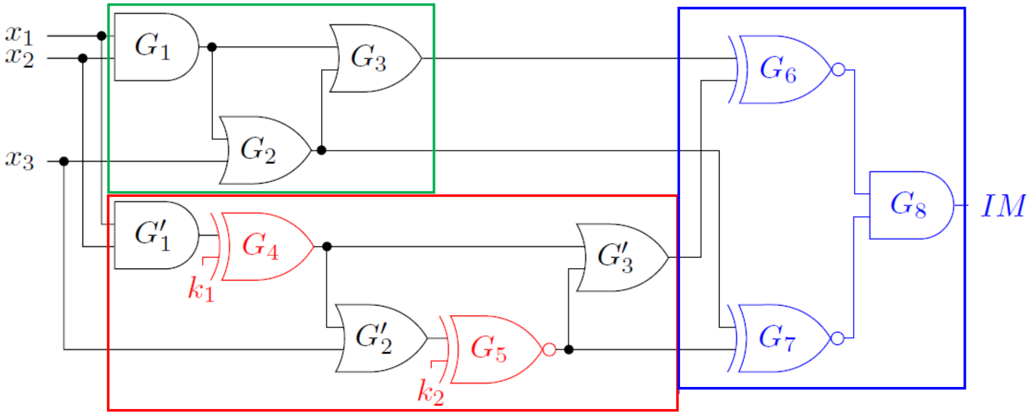
## Quality Measure: Fraction of Keys with High Criticality

- What we actually want to have is:

  **To keep the fraction of keys with high criticality low**!

- **How to compute this?**
- In principle (but not efficient at all!):
  - Compute for each fixed key $\vec{K}_{fix}$ its criticality:

    $\text{ꓤ}^{0.5}\vec{X}: f_{IM}|_{\vec{K}_{fix}}$
  - Compare the criticality with the „acceptable criticality bound“ $c$: I.e. check whether $\text{ꓤ}^{0.5}\vec{X}: f_{IM}|_{\vec{K}_{fix}} > c$
  - Compute the fraction of keys for which the comparison holds ⇒ „**fraction of keys with high criticality**“
  - Compare this fraction with „allowed value“ $d$.

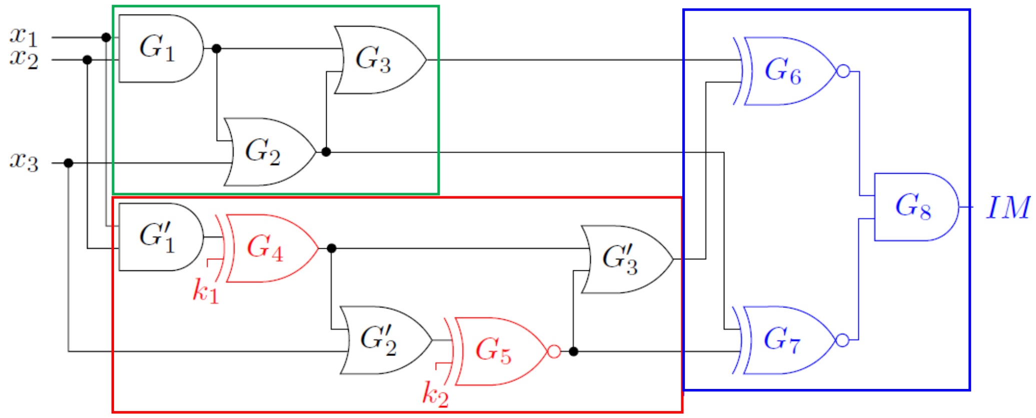


# 3. Reduction to Existing SAT-related Problems
## Quality Measure: Fraction of Keys with High Criticality

- Compute for each fixed key $\vec{K}_{fix}$ its criticality:
  $$\text{ꓤ}^{0.5}\vec{X}\colon f_{IM}|_{\vec{K}_{fix}}$$
- Compare the criticality with the „acceptable criticality bound“ $c$: I.e. check whether $\text{ꓤ}^{0.5}\vec{X}\colon f_{IM}|_{\vec{K}_{fix}} > c$
- Compute the fraction of keys for which the comparison holds
  $\Rightarrow$ „**fraction of keys with high criticality**“
- Compare this value with „allowed value“ $d$.

$\Rightarrow$ You have to solve a formula like

$$((\text{ꓤ}^{0.5}\vec{K}((\text{ꓤ}^{0.5}\vec{X}\colon f_{IM}) > c)) > d).$$




$\Rightarrow$ New formula class **Hierarchical Stochatic SAT** (**HSSAT**)

# 4. Hierarchical Stochastic SAT
## Syntax Definition

**(Detailed formal definition in the paper)**

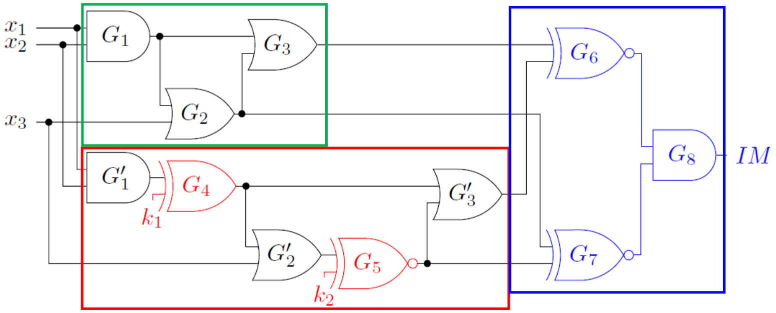
- Any Boolean formula is an HSSAT formula.
- If $\Phi$ is an HSSAT formula, then
  - $(\exists x \Phi)$ is an HSSAT formula,
  - $(\forall x \Phi)$ is an HSSAT formula,
  - $(\rotatebox[origin=c]{180}{R}^{p} x \Phi)$ with $p \in [0, 1]$ is an HSSAT formula,
  - $(\Phi\ op\ q)$ with $op \in \{<, \leq, >, \geq, =, \neq\}$, $q \in [0, 1]$ is an HSSAT formula.

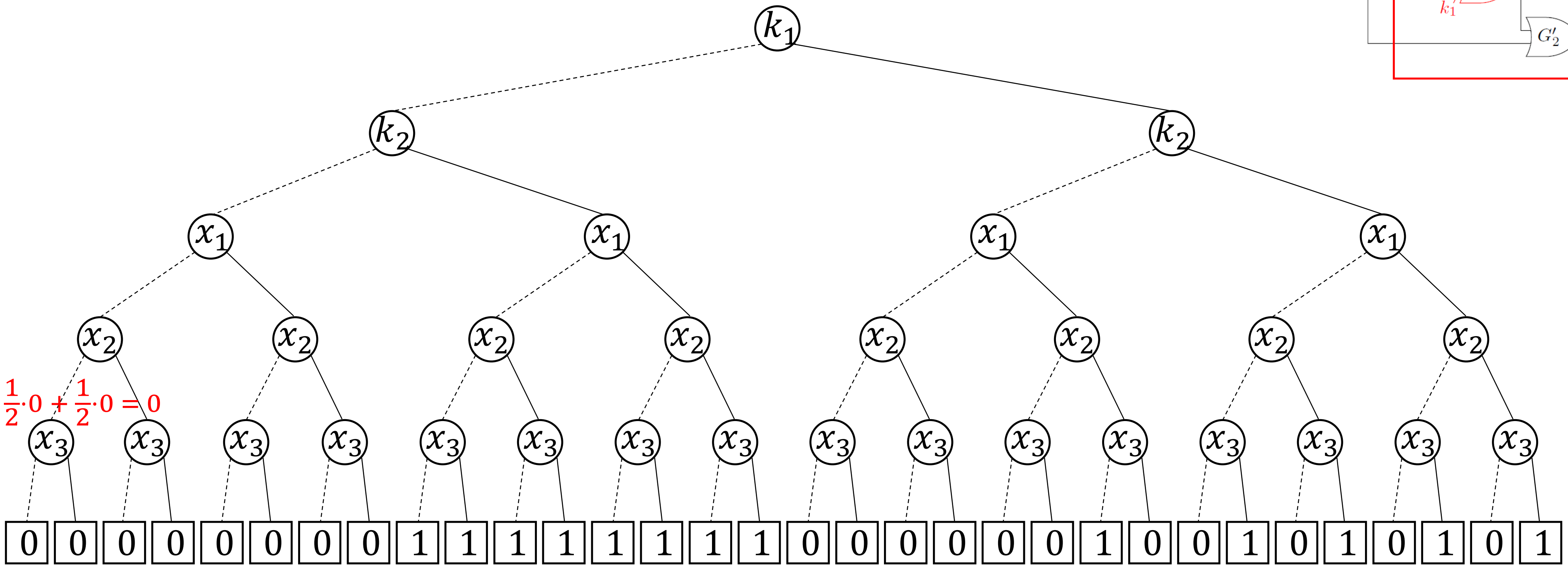
„nested comparison“

# 4. Hierarchical Stochastic SAT

## Semantics Definition, explained by Example

- Semantics definition similar to SSAT, but with „nested comparisons".
- **Example** (cont.): $[(\text{ꓤ}^{0.5} k_1 \text{ꓤ}^{0.5} k_2 [(\text{ꓤ}^{0.5} x_1 \text{ꓤ}^{0.5} x_2 \text{ꓤ}^{0.5} x_3 : f_{IM}) \geq 0.99]) \geq 0.3]$

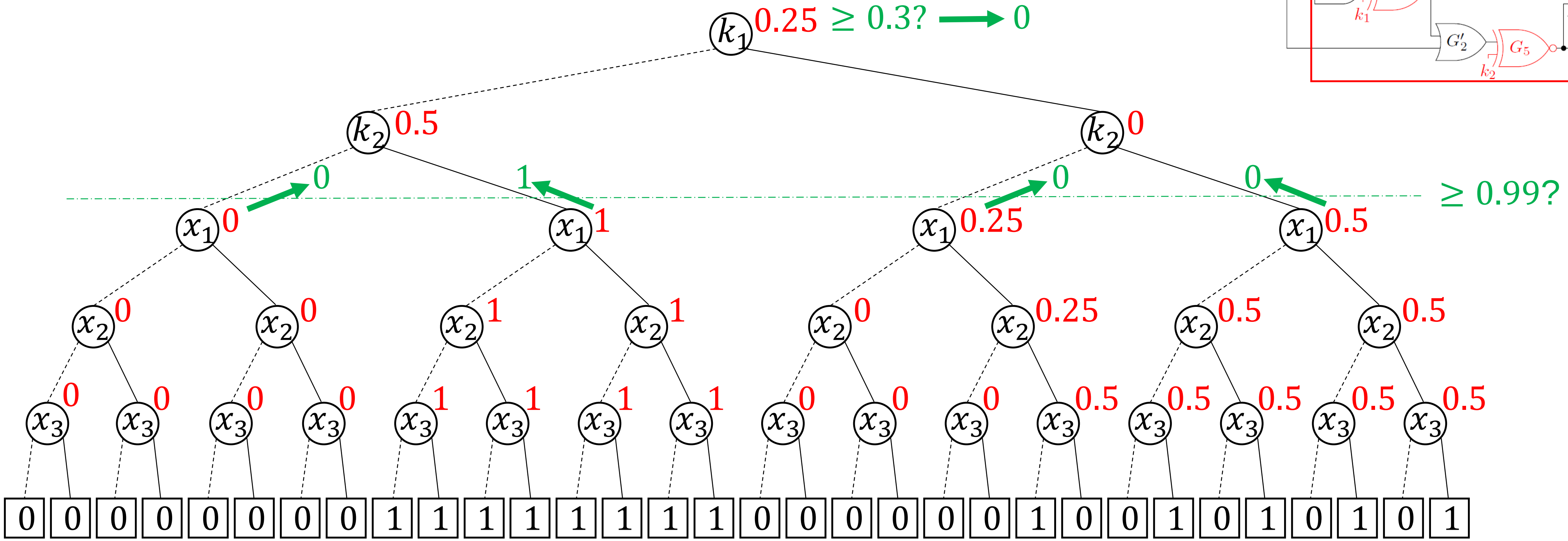


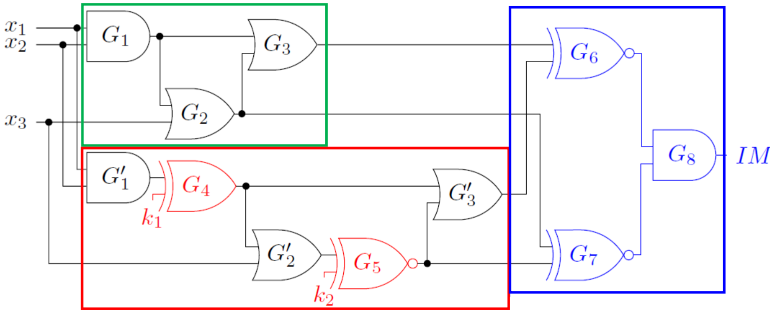



# 4. Hierarchical Stochastic SAT

## Semantics Definition, explained by Example

- Semantics definition similar to SSAT, but with „nested comparisons“.
- **Example** (cont.): $[(\text{ꓤ}^{0.5}k_1\text{ꓤ}^{0.5}k_2[(\text{ꓤ}^{0.5}x_1\text{ꓤ}^{0.5}x_2\text{ꓤ}^{0.5}x_3 : f_{IM}) \geq 0.99]) \geq 0.3]$

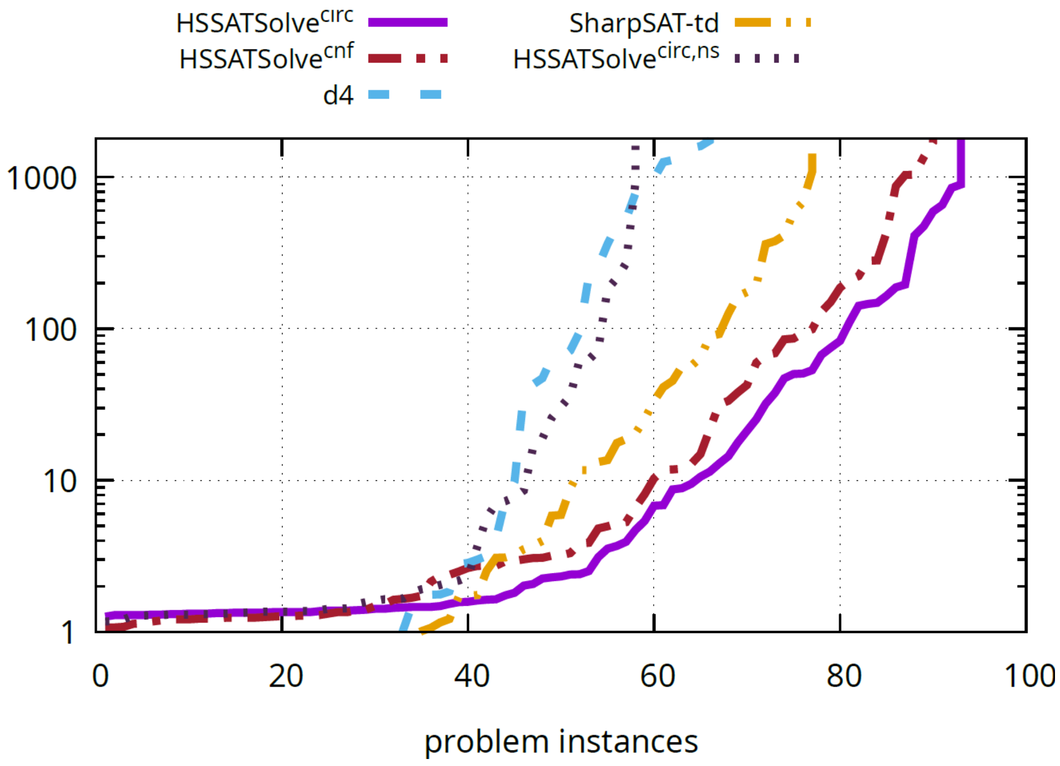


# 7. Experimental Results
## Part I: Sanity Check for Prototype Solver Using Known HSSAT Subclasses

### Model Counting

- Average Criticality of Keys: $ᴚ^{0.5}\vec{K}ᴚ^{0.5}\vec{X}: f_{IM}$

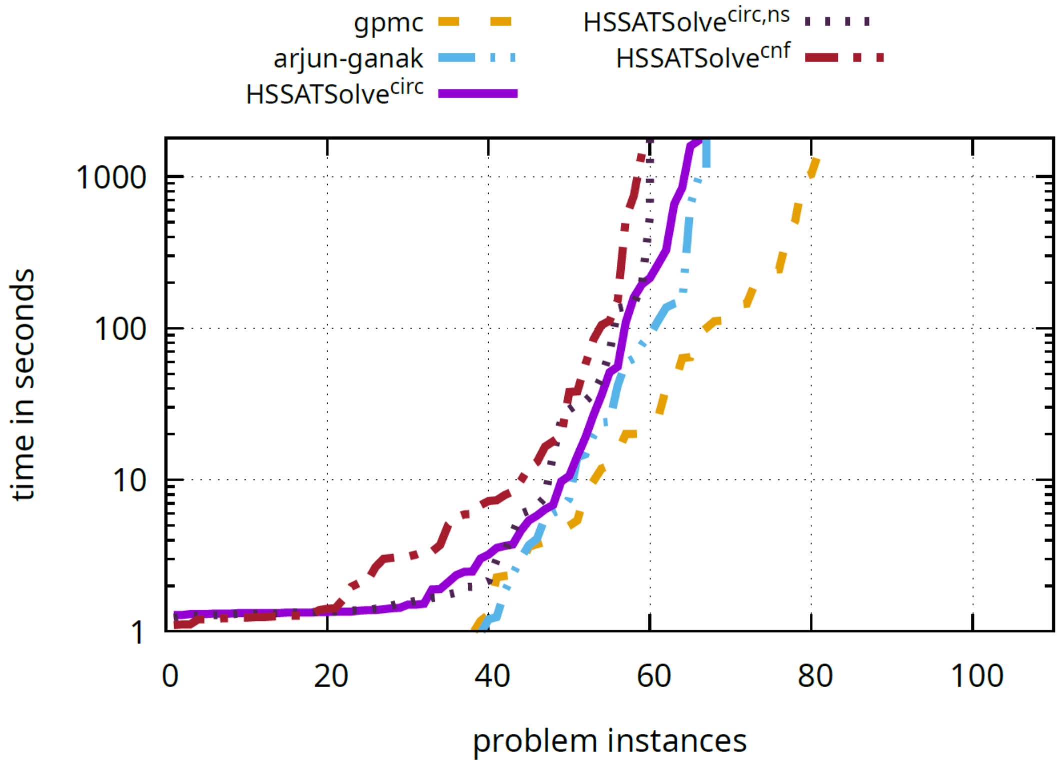


# 7. Experimental Results

## Part I: Sanity Check for Prototype Solver Using Known HSSAT Subclasses

### Projected Model Counting

- Fraction of not unlocking keys: $\text{Ɐ}^{0.5}\vec{K}\exists\vec{X}: \neg f_{IM}$

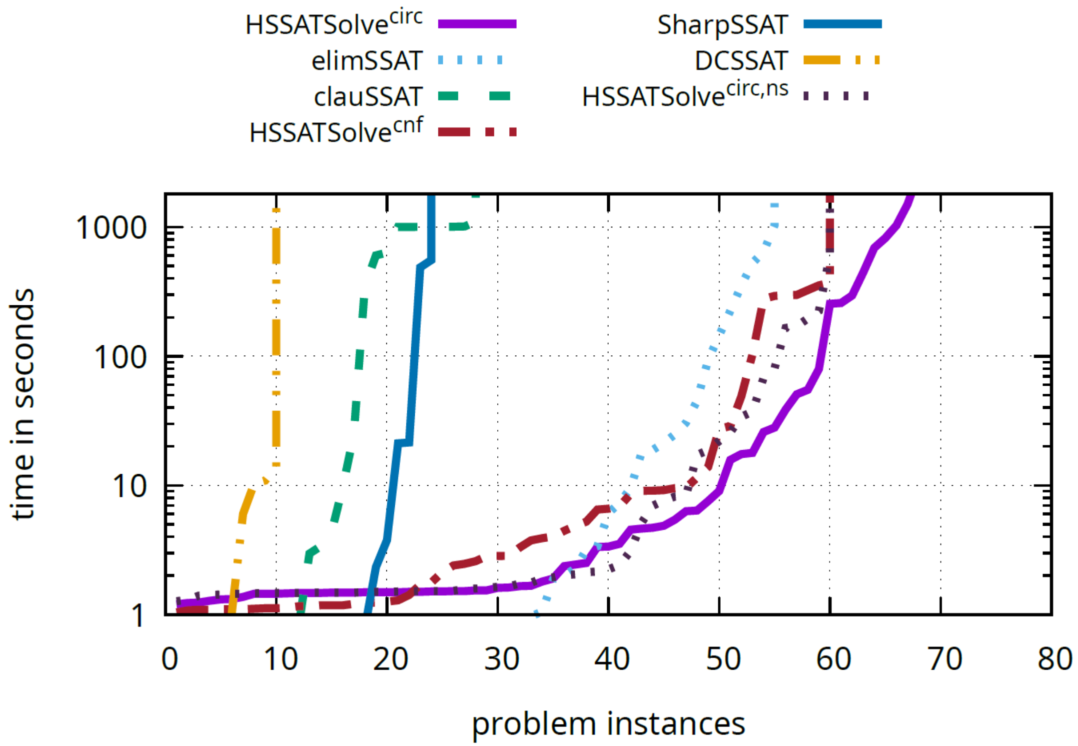


# 7. Experimental Results

## Part I: Sanity Check for Prototype Solver Using Known HSSAT Subclasses

### Stochastic SAT (SSAT)

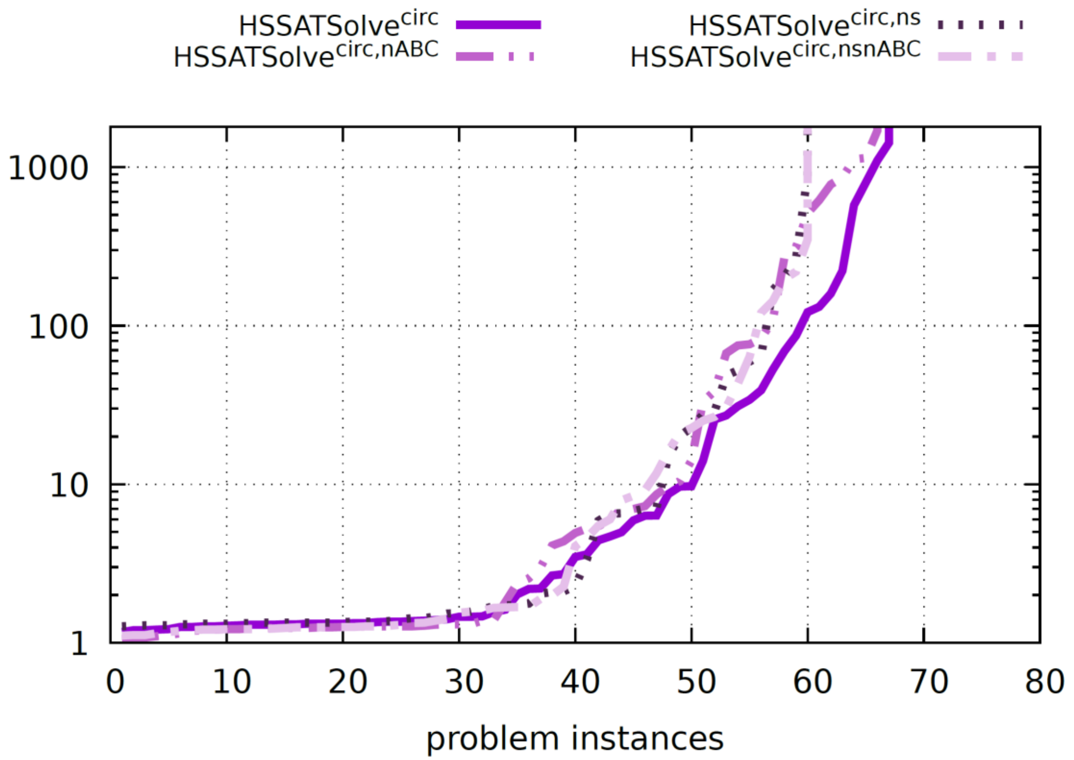
- Existence of Keys with High Criticality: $(\exists \vec{K}\, \text{ᴚ}^{0.5} \vec{X}\colon [f_{IM} \wedge (\vec{K} \neq \vec{K}_{orig})]) > c$



# 7. Experimental Results
## Part II: Results for HSSAT Formulas

### Hierarchical Stochastic SAT (HSSAT)

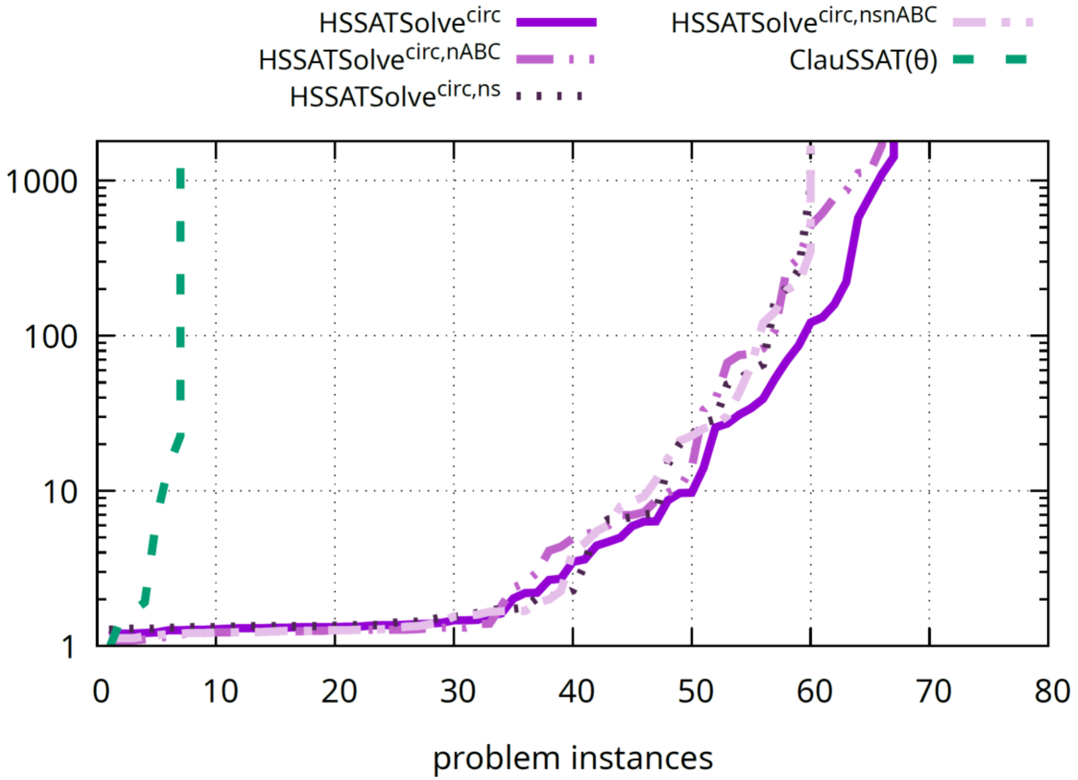
- Fraction of Keys with High Criticality: $((\rotatebox{180}{$R$}^{0.5}\vec{K}((\rotatebox{180}{$R$}^{0.5}\vec{X}: f_{IM}) > c)) > d)$.



# 7. Experimental Results
## Part II: Results for HSSAT Formulas

### Hierarchical Stochastic SAT (HSSAT)

- Fraction of Keys with High Criticality: $((Ⅎ^{0.5}\vec{K}((Ⅎ^{0.5}\vec{X}: f_{IM}) > c)) > d)$.



# 8. Conclusions and Future Work

- New problem class HSSAT, motivated by quality assessment of logic locking
- HSSAT is PSPACE complete (as QBF and SSAT)
- First ROBDD-based prototype solver HSSATSolve
- First interesting results in the application domain
- Provides benchmarks also for subclasses of HSSAT

- Improve solver
- Compare different logic locking methods with precise evaluation of quality measures

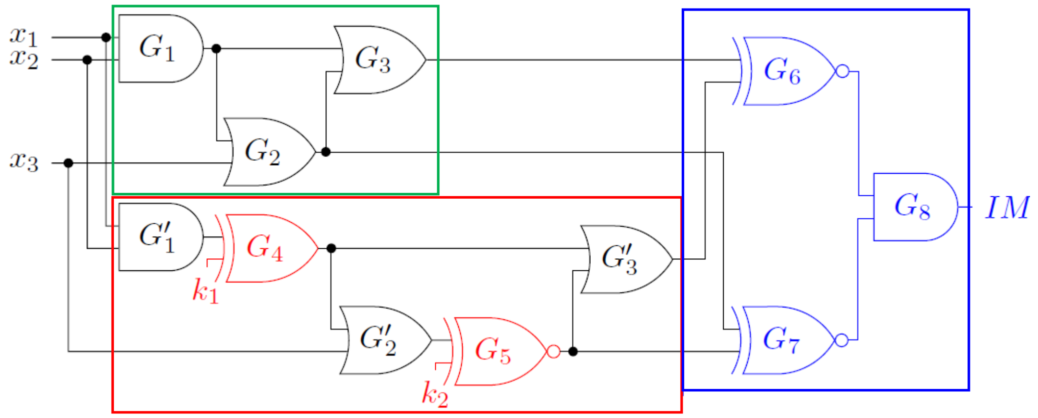
# 3. Reduction to Existing SAT-related Problems

## Quality Measure 1: Key Uniqueness

- Check whether there exists a key different from the original (intended) unlocking key $\vec{K}_{orig}$ that unlocks the circuit:

$$\exists \vec{K} \forall \vec{X} : [f_{IM} \wedge (\vec{K} \neq \vec{K}_{orig})]$$

$\Rightarrow$ Quantified Boolean Formula (**QBF**)



# 3. Reduction to Existing SAT-related Problems

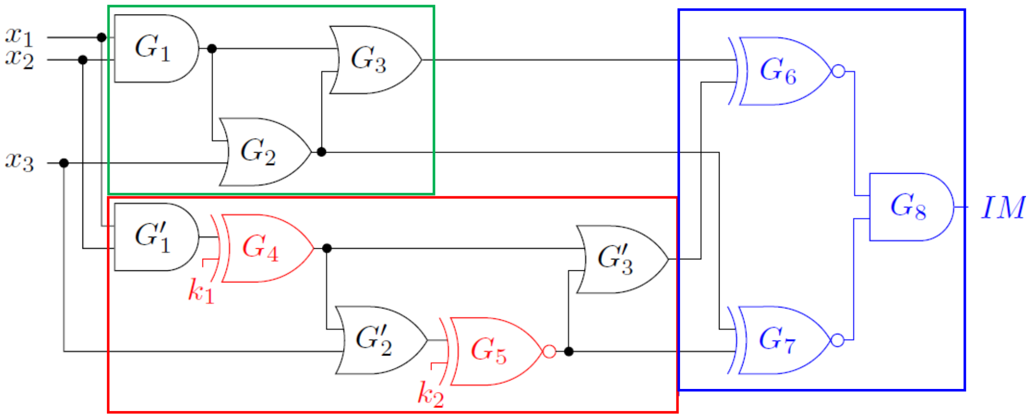
## Quality Measure 2: Fraction of Unlocking Keys

- We do not want to have a large number of unlocking keys …

$\Rightarrow$ Compute the **fraction** of unlocking keys:

$$\rotatebox[origin=c]{180}{R}^{0.5}\vec{K}\forall\vec{X}: f_{IM}$$

- Here the random quantifier $\rotatebox[origin=c]{180}{R}^{p}$ is defined as in **Stochastic SAT** (**SSAT**) formulas $\Phi$ which compute satisfying probabilities $\Pr[\Phi]$:
  - $\Pr[\Phi] = 0$, if $\Phi \equiv 0$,
  - $\Pr[\Phi] = 1$, if $\Phi \equiv 1$,
  - $\Pr[\rotatebox[origin=c]{180}{R}^{p} x\Phi] = (1-p)\cdot\Pr[\Phi|_{\neg x}] + p\cdot\Pr[\Phi|_{x}]$,
  - $\Pr[\exists x\Phi] = \max(\Pr[\Phi|_{\neg x}], \Pr[\Phi|_{x}])$,
  - $\Pr[\forall x\Phi] = \min(\Pr[\Phi|_{\neg x}], \Pr[\Phi|_{x}])$.



# 3. Reduction to Existing SAT-related Problems
## Quality Measure 2: Fraction of Unlocking Keys

- We do not want to have a large number of unlocking keys …

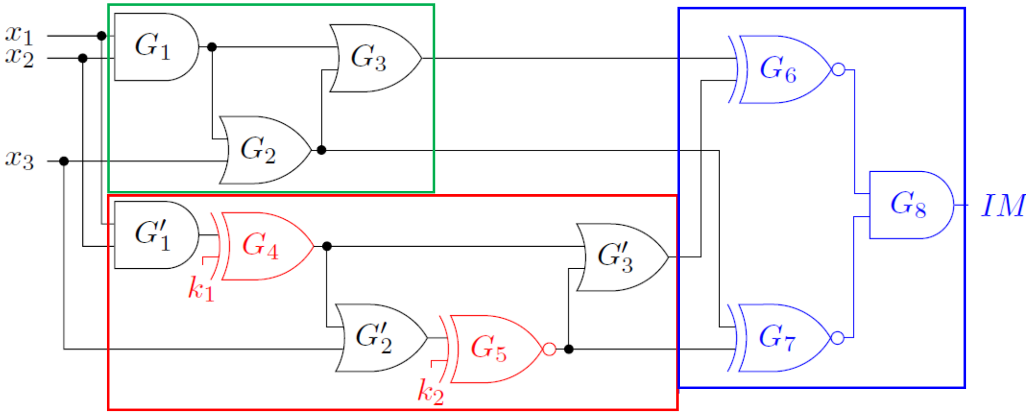
⇒ Compute the **fraction** of unlocking keys:

$$\rotatebox[origin=c]{180}{R}^{0.5}\vec{K}\forall\vec{X}\colon f_{IM}$$

⇒ Stochastic SAT (**SSAT**)

- But if we compute the negation (fraction of keys which are **not** unlocking) we only need **Projected Model Counting**:

$$\rotatebox[origin=c]{180}{R}^{0.5}\vec{K}\exists\vec{X}\colon \neg f_{IM}$$

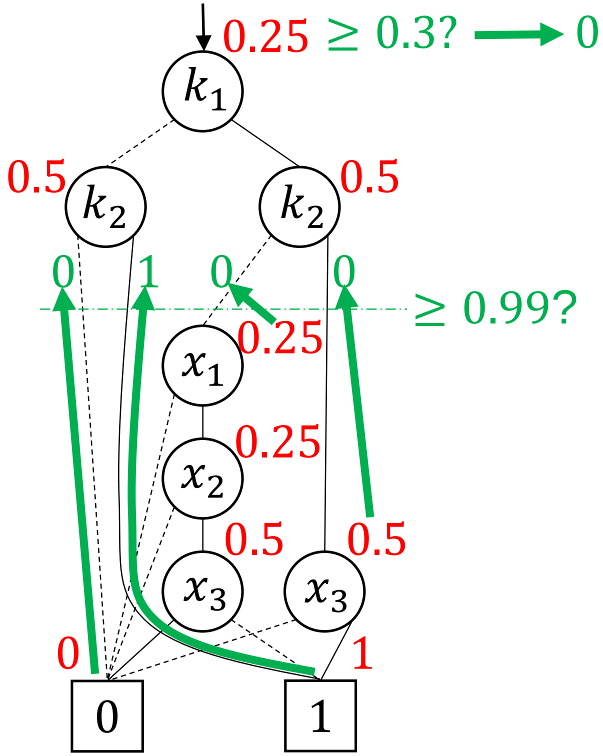


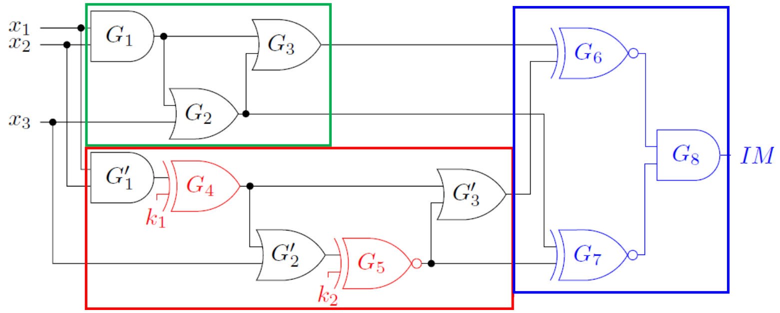
# 5. Prototype for Solving HSSAT
## An ROBDD-based Algorithm

- Semantics definition immediately suggests two solution approaches:
  - DPLL-based algorithm
  - ROBDD-based algorithm
- Here: **ROBDD-based algorithm** as a prototype
  - Build ROBDD for the matrix with variable order according to the prefix of the HSSAT formula
  - Do a bottom-up evaluation of the ROBDD (similar to the decision tree)
    - Node sharing (isomorphism reductions) just increase the efficiency
    - „Long edges“ (Shannon reductions) increase the efficiency, but need some special attention, if they „cross levels with nested comparisons“

# 5. Prototype for Solving HSSAT
## An ROBDD-based Algorithm

- **Example** (cont.): $[(Я^{0.5}k_1 Я^{0.5}k_2[(Я^{0.5}x_1 Я^{0.5}x_2 Я^{0.5}x_3 : f_{IM}) \geq 0.99]) \geq 0.3]$

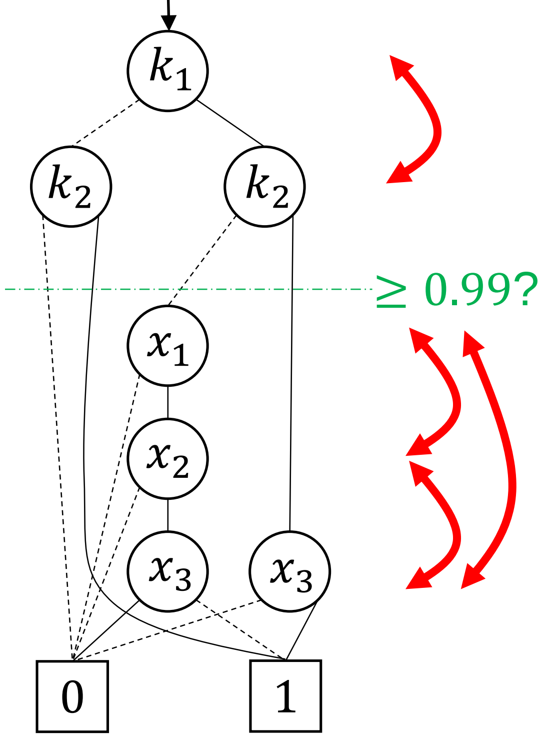


# 6. Two Improvements

- Some **flexibility** wrt. ROBDD **variable order**:
  - Exchanging variables within blocks of identical quantifiers allowed
  - But: Quantifier blocks are cut by nested comparisons!
  
  $\Rightarrow$ Dynamic variable ordering by **group sifting**
- In case of matrix in CNF:
  **Semantic gate detection** with **UNIQUE**[1]
  - Be careful!
  - Needs adjustment for nested comparisons

- **Example** (cont.):
  $[(\text{Я}^{0.5}k_1\text{Я}^{0.5}k_2[(\text{Я}^{0.5}x_1\text{Я}^{0.5}x_2\text{Я}^{0.5}x_3: f_{IM}) \geq 0.99]) \geq 0.3]$



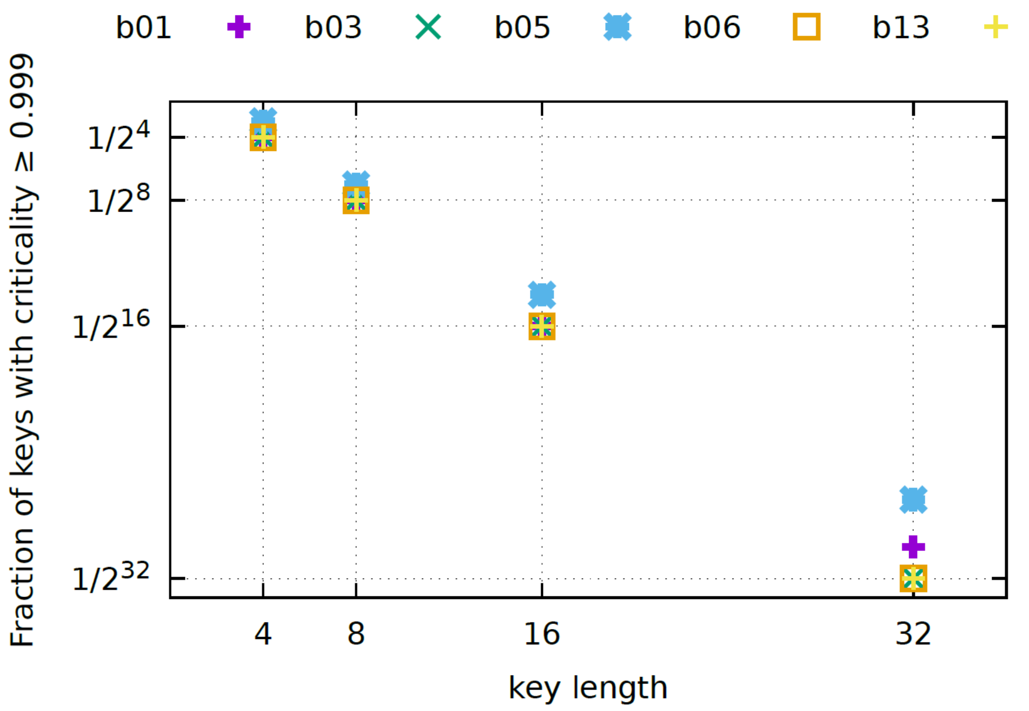

[1]F. Slivovsky: Interpolation-based semantic gate extraction and its applications to QBF preprocessing. CAV 2020.

# 7. Experimental Results

## Part II: Results for HSSAT Formulas

### Fraction of Keys with High Criticality – Different Key Lengths

- Fraction of keys with criticality $\geq 0.999$
- Results for different circuits and key lengths 4, 8, 16, 32

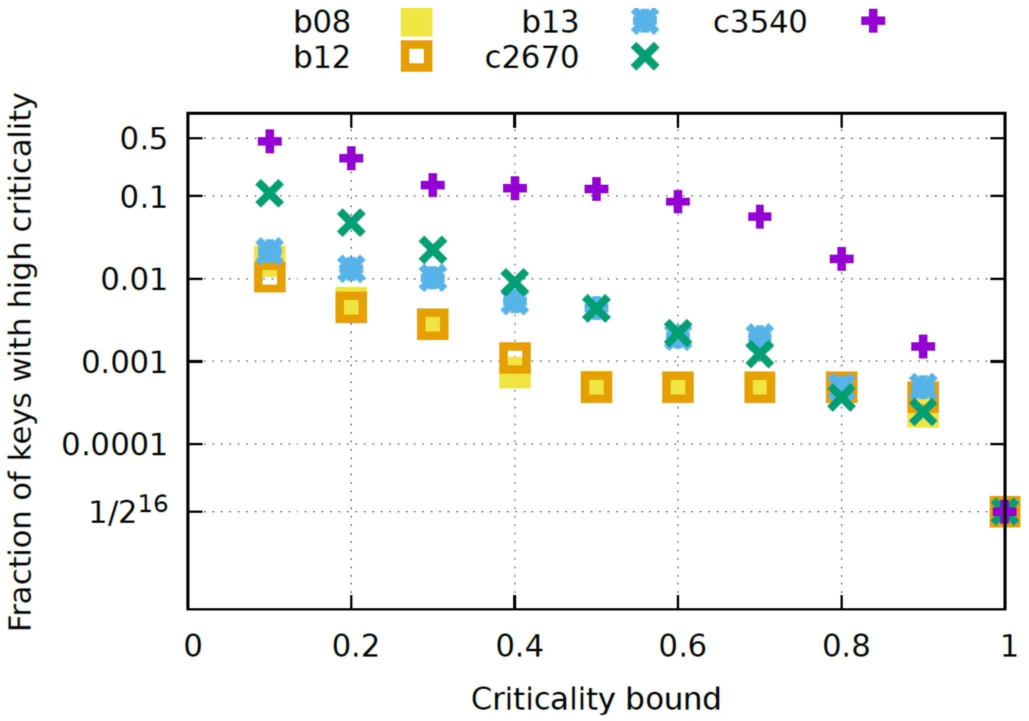


# 7. Experimental Results

## Part II: Results for HSSAT Formulas

### Fraction of Keys with High Criticality – Different Criticality Bounds

- Fixed key length of 16
- Results for different circuits, fraction of keys with criticality ≥ 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0



# 1. Introduction to Logic Locking
## Method

- **Scenario**:
  - Foundry delivers locked ICs to the design house
  - Design house stores secret key in non-volatile tamper-proof memory
  - Unlocked chips are sold by design house