Heuristic Optimizations for Boolean Formulas with Applications in Attribute-Based Encryption

Bachelor's Thesis in Computer Science

Iulian Oleniuc

June 2023

Table of Contents

Introduction

Cloud Computing Context

- Modern enterprise software relies more and more on cloud services.
- **Security Problem:** Every user has access to all the sensitive data.
- We need ways to provide access to data based on certain criteria.
- **Solutions:** Role-Based Access Control and Attribute-Based Access Control.
- ABAC is more *expressive* than RBAC.

Attribute-Based Encryption

Introduction

- ABE was introduced by Goyal et al. in 2006.
- Attributes are assigned to both users and files.
- Complex *fine-grained* access policies can be defined over the attributes.
- They support conjunctions, disjunctions, and (k, n)-threshold gates.
- ABE comes in two flavors: CP-ABE and KP-ABE.

Key-Policy ABE

Introduction

- Let $\mathcal{U} \triangleq \{A, B, C, D\}$ be the universe of attributes.
- An access policy φ is embedded into the secret key of the user.
- The ciphertext is associated with a set of attributes S.

Example

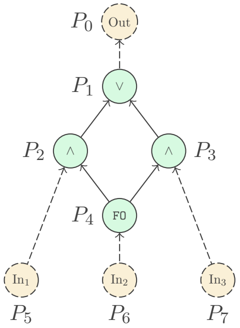
- Let $\varphi = ((A \vee B) \wedge C)$.
- They can decrypt a file with $S = \{A, C\}$.
- They cannot decrypt a file with $S = \{A, B\}$.

KP-ABE for Boolean Circuits
Introduction

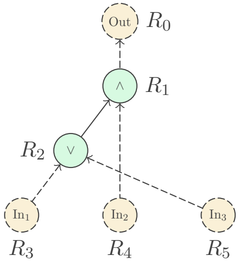
- The policies were originally modeled through Boolean trees.
- Țiplea and Drăgan show how we can replace them with Boolean *circuits* (which are more expressive).
- The running time of each phase of the scheme depends strictly on the number of *pairings* (i.e., operations involving bilinear maps) being computed.
- This equals the number of paths from the output node to the input nodes.
- **Our Goal:** We want to minimize this number!

Example of Equivalent Circuits

Introduction



(a)



(b)

Figure: The first circuit corresponds to formula $((\mathrm{In}_1 \wedge \mathrm{In}_2) \vee (\mathrm{In}_2 \wedge \mathrm{In}_3))$ and has cost 4. The second one corresponds to the equivalent formula $((\mathrm{In}_1 \vee \mathrm{In}_3) \wedge \mathrm{In}_2)$ and has cost 3.

Our Contribution

Introduction

- We developed a set of heuristics that optimize Boolean formulas with up to 60% improvement in real-world scenarios.
- We sent a paper (alongside a C++ library) to the KES conference on this topic and it was accepted.
- We also came up with a completely different idea of optimizing Boolean circuits with great cryptographic potential.
- We sent a more detailed paper (alongside a Python library) for this thesis.

Table of Contents

Monotone Boolean Formula Optimization

The Optimization Problem

Monotone Boolean Formula Optimization

- A Boolean formula is said to be *monotone* if it does not contain any negations.
- The *cost* of a formula φ, denoted by $c(\varphi)$, is the number of literals in φ.
- **Our Problem:** For a given monotone Boolean formula φ, find an equivalent monotone Boolean formula ψ with a cost as low as possible compared to the one of φ. In other words, find some ψ such that $\psi \equiv \varphi$ and $c(\psi) \ll c(\varphi)$.

Clean Formulas

Monotone Boolean Formula Optimization

- We say that a formula is *clean* if its associated AST does not contain any of the following structural design flaws:
  1. any node with only one child;
  2. any node with the same operator as its parent;
  3. any node with two or more children of the same formula.
- In order to maintain these invariants, we need to *trim* the tree.

Example

- Formula $((a \wedge b)) \equiv (a \wedge b)$ breaks invariant (1).
- Formula $(a \vee (b \vee c)) \equiv (a \vee b \vee c)$ breaks invariant (2).
- Formula $(a \wedge a \wedge b) \equiv (a \wedge b)$ breaks invariant (3).

Trimming the AST (1)

Monotone Boolean Formula Optimization

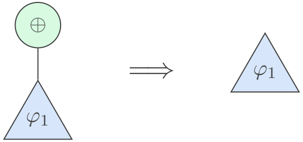


Figure: (φ_1) becomes φ_1.

Trimming the AST (2)

Monotone Boolean Formula Optimization

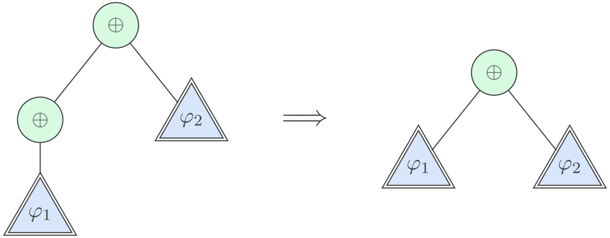


Figure: $((\varphi_1) \oplus \varphi_2)$ becomes $(\varphi_1 \oplus \varphi_2)$.

Trimming the AST (3)

Monotone Boolean Formula Optimization

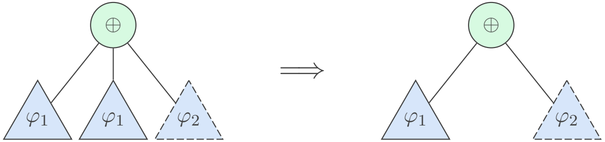


Figure: $(\varphi_1 \oplus \varphi_1 \oplus \varphi_2)$ becomes $(\varphi_1 \oplus \varphi_2)$.

Operating on the AST

Monotone Boolean Formula Optimization

- In order for our heuristics to work, we need to define various operations capable of both increasing and decreasing the cost of the formula.
- **Factorization** decreases the cost: $((a \wedge b) \vee (a \wedge c)) \equiv (a \wedge (b \vee c))$.
- **Absorption** decreases the cost: $(a \vee (a \wedge b)) \equiv a$.
- **Distribution** increases the cost: $(a \wedge (b \vee c)) \equiv ((a \wedge b) \vee (a \wedge c))$.
- Every such operation must be followed by a trim of the AST.

Operating on the AST (Factorization)

Monotone Boolean Formula Optimization

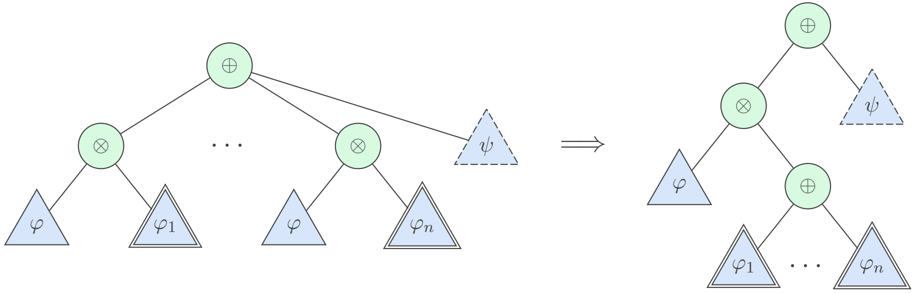


Figure: $(\varphi \otimes \varphi_1) \oplus \cdots \oplus (\varphi \otimes \varphi_n)$ becomes $(\varphi \otimes (\varphi_1 \oplus \cdots \oplus \varphi_n))$.

Operating on the AST (Absorption)

Monotone Boolean Formula Optimization

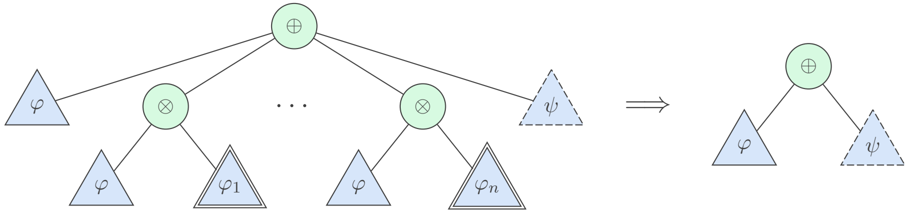


Figure: $\varphi \oplus (\varphi \otimes \varphi_1) \oplus \cdots \oplus (\varphi \otimes \varphi_n)$ becomes φ.

Operating on the AST (Distribution)

Monotone Boolean Formula Optimization

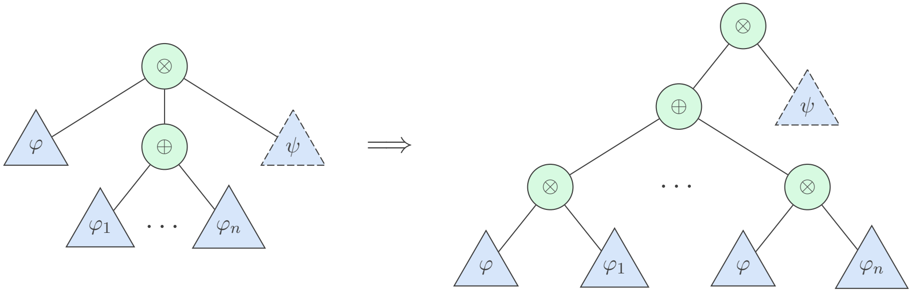


Figure: $\varphi \otimes (\varphi_1 \oplus \cdots \oplus \varphi_n)$ becomes $((\varphi \otimes \varphi_1) \oplus \cdots \oplus (\varphi \otimes \varphi_n))$.

Proposed Heuristics (Naïve)

Monotone Boolean Formula Optimization

- The Naïve approach randomly tries to apply factorizations and absorptions until there are no available contexts anymore.
- It does not care how good the improvement of a given transformation is, nor what future optimization potential it offers.

Algorithm Naïve

Require: The AST T.

```
1 while trying to decrease $c(T)$ succeeded do
2     nothing
3 end while
```

Proposed Heuristics (Hill Climbing)

Monotone Boolean Formula Optimization

- We view the problem as a directed graph whose nodes are ASTs.
- Its edges are transitions from one state to another (factorizations and absorptions).
- The algorithm starts from the initial state T and repeatedly moves to a *better* (i.e., $c(T') < c(T)$) neighboring state T'.
- At each step, it randomly chooses only k neighbors (i.e., k random contexts where the operation can be applied) and picks up one having the minimum cost.
- **Downside:** Since Hill Climbing greedily chooses the best transition at each step, it can get stuck in a local minimum.

Proposed Heuristics (Hill Climbing)

Monotone Boolean Formula Optimization

Algorithm Hill Climbing

Require: The AST T.

```
while true do
    $c_{\min} \leftarrow \infty$
    $T_{\min} \leftarrow$ null
    for $i = \overline{1, k}$ do
        let $T'$ be a clone of $T$
        if trying to decrease $c(T')$ failed then          ▷ no more neighbors of $T$
            exit
        end if
        $c \leftarrow c(T')$
        if $c < c_{\min}$ then
            $c_{\min} \leftarrow c$
            $T_{\min} \leftarrow T'$
        end if
    end for
    $T \leftarrow T_{\min}$
end while
```

Proposed Heuristics (Simulated Annealing)

Monotone Boolean Formula Optimization

- Simulated Annealing is inspired by the real process of *physical annealing*.
- The latter consists of heating up a material until it reaches a specific temperature; while hot, its molecular structure is more *susceptible* to change.
- After that, it will be cooled down slowly, with the goal of changing its structure to a desired one; while cool, its structure is more *resistant* to external forces.

Proposed Heuristics (Simulated Annealing)

Monotone Boolean Formula Optimization

Algorithm Simulated Annealing

```
Require: The AST $T$.
$t \leftarrow t_{\max}$
while $t > t_{\min}$ do
    for $i \in \overline{1, n}$ do
        let $T'$ be a clone of $T$
        $c_1 \leftarrow c(T')$
        if rand$(0, 1) < p$ then
            try to increase $c(T')$
        else
            try to decrease $c(T')$
        end if
        $c_2 \leftarrow c(T')$
        $\Delta \leftarrow (c_2 - c_1)/c_1$                ▷ $\Delta$ as percentage
        if $\Delta < 0$ or rand$(0, 1) < e^{-\Delta/t}$ then
            $T \leftarrow T'$
        end if
    end for
    $t \leftarrow t/(1 + \beta \cdot t)$                   ▷ cooling $t$
end while
while trying to decrease $c(T)$ succeeded do
    nothing
end while
```

Proposed Heuristics (Custom Heuristic)

Monotone Boolean Formula Optimization

- Some sort of a lighter version of Simulated Annealing.
- It keeps track of the moments when no more factorizations or absorptions could be made, in order to forcefully make a distribution on the next step.

Algorithm Custom Heuristic

```
Require: The AST $T$.
factorizable ← true
for $i \in \overline{1,n}$ do
    if not factorizable or rand$(1, \alpha \cdot n) < n - i$ then
        try to increase $c(T)$
        factorizable ← true
    else if trying to decrease $c(T)$ failed then
        factorizable ← false
    end if
end for
while trying to decrease $c(T)$ succeeded do
    nothing
end while
```

Table of Contents

Testing Our Main Approach

Datasets

Testing Our Main Approach

- We generated two kinds of datasets.
- The first kind consists of the datasets `small`, `medium`, and `large`, named according to the average size of the formulas they contain.
- The second kind consists of just the dataset `real`, written manually, containing formulas that model a specific access policy that arises in practical ABE systems, namely *comparison queries*.

Comparison Queries

Testing Our Main Approach

- **Example:** $(((1965 \leq y \wedge y \leq 1980) \vee y \geq 1997) \wedge (6 \leq m \wedge m \leq 8))$.
- A pattern always occurring in this kind of policy is the comparison between two integers, one known a priori and the other being an input.

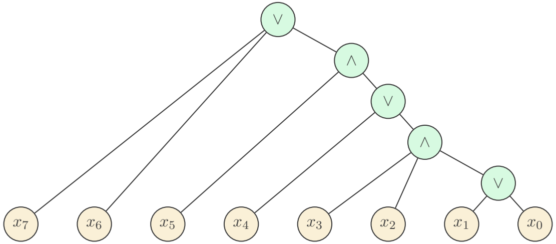


Figure: The AST corresponding to the comparison query $x \geq 45 = (00101101)_2$. Note that $(x_7 x_6 \cdots x_0)_2$ is the base 2 representation of the input x on 8 bits.

Results

Testing Our Main Approach

	Heuristic	Avg. Score (%)	Max. Score (%)	Avg. Time (s)
small	Naïve	25.65	27.21	0.00
	Hill Climbing	27.16	27.54	0.02
	Simulated Annealing	28.77	38.51	0.57
	Custom Heuristic	19.61	35.46	0.07
medium	Naïve	33.15	35.92	0.01
	Hill Climbing	35.89	36.66	0.09
	Simulated Annealing	22.55	40.73	1.32
	Custom Heuristic	19.11	37.07	0.15
large	Naïve	46.81	50.56	0.04
	Hill Climbing	53.40	54.19	0.34
	Simulated Annealing	38.31	58.42	2.60
	Custom Heuristic	25.32	51.30	0.41
real	Naïve	3.78	5.31	0.00
	Hill Climbing	5.31	5.31	0.01
	Simulated Annealing	39.73	60.65	0.89
	Custom Heuristic	13.38	32.58	0.11

Table: The results for each dataset and each heuristic.

Iterated Simulated Annealing always yields the best result, but at a very high cost.

Hill Climbing (for general scenarios) and Iterated Custom Heuristic (for ABE systems) ensure some very good improvements for a way lower running time.

Table of Contents

Alternative Approach

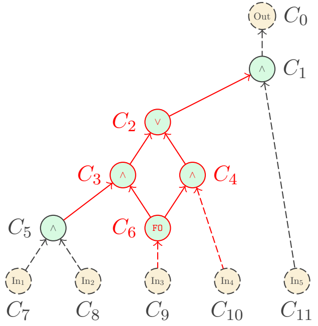
Subcircuit Replacement Approach

Alternative Approach

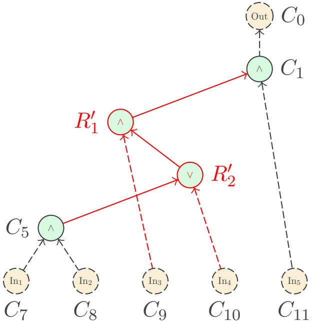
- This new approach, unlike the previous one, does not operate on the AST associated with the formula, but rather on its corresponding Boolean circuit (i.e., the actual representation of the ABE policy).
- Briefly, this technique is about searching for certain subcircuit patterns inside the given circuit and replacing them with lower-cost equivalent subcircuits.
- Unfortunately, it yielded zero improvements for any non-trivial circuit.
- However, we believe it can be adjusted to obtain good results if we use cryptographic strategies to improve the *secret sharing* of subcircuits (using CAS nodes), instead of just relying on the logical equivalence of their formulas.

Replacement Example

Alternative Approach



(a) $((((\mathrm{In}_1 \wedge \mathrm{In}_2) \wedge \mathrm{In}_3) \vee (\mathrm{In}_3 \wedge \mathrm{In}_4)) \wedge \mathrm{In}_5)$



(b) $((((\mathrm{In}_1 \wedge \mathrm{In}_2) \vee \mathrm{In}_4) \wedge \mathrm{In}_3) \wedge \mathrm{In}_5)$

Figure: The process of replacing the occurrence of $\mathcal{P}$ in $\mathcal{C}$ with a copy of $\mathcal{R}$.

Table of Contents
Conclusions

Achievements
Conclusions

- We proposed multiple heuristics for optimizing monotone Boolean formulas.
- They provide a considerable improvement in circuit size for the KP-ABE scheme described by Țiplea and Drăgan.
- Our optimization strategy can have a significant impact on cloud systems that implement fine-grained cryptographic access control over data through ABE.
- Our paper was accepted for the KES conference.
- **Further Work:** There is still room for improvement, especially for the second approach.

Python Library

Conclusions

- We wrote an open-source Python library, publicly available for anyone to use.
- Its purpose is the testing of various heuristics and comparing the results on the same datasets as we used.
- It also provides a CLI that can be used directly to generate lower-cost equivalent formulas for those in the input.

Bibliography

- V. Goyal, O. Pandey, A. Sahai, and B. Waters. *Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data.* 2006
- F.L. Țiplea and C.C. Drăgan. *Key-Policy Attribute-Based Encryption for Boolean Circuits From Bilinear Maps.* 2014
- A. Ioniță, D. Banu, and I. Oleniuc. *Heuristic Optimizations of Boolean Circuits with Application in Attribute-Based Encryption.* 2023
- A. Ioniță. *Optimizing Attribute-Based Encryption for Circuits using Compartmented Access Structures.* 2023

Q & A

Thank you for your attention!