Available online at www.sciencedirect.com

Procedia Computer Science 00 (2023) 000–000

www.elsevier.com/locate/procedia

27th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2023)

Heuristic Optimizations of Boolean Circuits with Application in Attribute-Based Encryption

Alexandru Ioniţă[a,1,*], Denis-Andrei Banu[a], Iulian Oleniuc[a]

[a] *"Alexandru Ioan Cuza" University of Iaşi, Address, Iaşi and Postcode, Romania*
[b] *"Simon Stoilow" Institute of Mathematics of the Romanian Academy, Address, Bucharest and Postcode, Romania*

Abstract

We propose a method of optimizing monotone Boolean circuits by re-writing them in a simpler, equivalent form. We use in total six heuristics: Hill Climbing, Simulated Annealing, and variations of them, which operate on the representation of the circuit as a logical formula. Our main motivation is to improve performance in Attribute-Based Encryption (ABE) schemes for Boolean circuits. Therefore, we show how our heuristics improve ABE systems for Boolean circuits. Also, we run tests to evaluate the performance of our heuristics, both as a standalone optimization for Boolean circuits and also inside ABE systems.

Keywords: Heuristics; Nature-inspired models; Encryption; Public-key cryptography; Boolean circuit minimization; Hill Climbing; Simulated Annealing

1. Introduction

Modern software systems rely more and more on cloud services for hosting services. These systems bring up a big privacy problem. While using such a service for hosting, including database storage, the Cloud Service Provider usually has access to all the sensitive data, such as client names, addresses, and, depending on the application hosted, possibly medical data or other types of private documents. The obvious solution would be to encrypt the data stored in the cloud. However, this raises the problem of finding expressive encryption systems, in order to obtain a fine-grained access granting mechanism, as we do not wish to offer access to a sensitive document to an unauthorized person.

For example, let's consider a scenario where a healthcare provider's app is hosted in the cloud. There, people could upload medical documents, download medical test results and talk with doctors. In order to grant access only to those who should have it, and to no one else, the old-fashioned way would be to generate a new encryption key for each document and encrypt it with the respective key. However, a document's decryption key should be shared with all the

[*] Corresponding author.
E-mail address: alexandru.p.ionita@gmail.com

persons who should be able to decrypt it. Having hundreds or thousands of such documents will make this approach infeasible, as each one of the documents will have its own keys.

Here comes in handy Attribute-Based Encryption (ABE), a relatively new encryption system, first introduced in [11]. With the ease of ABE, we can encrypt a document under certain attributes (such as "Cardiovascular", "Neurological") or even numerically, such as "Year:2023", "Priority:3". The decryption keys issued will contain an access structure that operates on such attributes. An example access structure could be: `(Cardiovascular OR Neurological) AND (Year:2022 OR Year:2021)`.

Multiple documents could be encrypted, each one under its own attribute set. When generating decryption keys, a single decryption key will decrypt many documents, if the attributes in the ciphertexts match the access policy in the key. Thus, we can create a system where we have the ability to grant access control based on descriptive attributes, using a single decryption key for multiple documents.

This type of ABE presented above, where ciphertexts have associated attributes that are matched with the access policies linked to decryption keys, is called *key-policy* ABE. In contrast, there also exists another flavor of ABE, namely *ciphertext-policy* [4], where the access policy is linked to the ciphertext, and the decryption keys have attributes associated with them.

As the complexity of a system increases, so does the complexity of the access structures. Therefore, a challenge for the ABE systems is to find more and more expressive access policies for which we can build efficient ABE systems. For example, the first ABE system [11] uses Boolean trees as access structures, where nodes can be `AND` and `OR` gates. A more expressive access structure could be a monotone Boolean circuit. Note that not all Boolean circuits can be expressed as access trees, but rather as *directed acyclic graphs*. However, for such access structures, the existing ABE schemes are inefficient [23, 13, 12] or rely on mathematical primitives for which there is no secure construction known [10, 9].

From this problem we get our motivation: we need to re-write a Boolean circuit in an equivalent form, such that the current ABE systems, like [23], will improve in efficiency.

1.1. Our Contribution

The most efficient Attribute-Based Encryption scheme for Boolean circuits from bilinear maps uses a secret sharing technique on Boolean circuits, which results in an exponential expansion in key (or ciphertext) size in the worst-case scenario. Our algorithms are optimizing the access structure – Boolean circuits – by finding equivalent circuits, for which the secret sharing is more efficient.

Besides our optimization results, we provide open access to our source code as a library which can be used to optimize Boolean circuits for the [23] scheme. Moreover, we also provide an archive with the test cases we used to evaluate the performance of our work, such that subsequent works could be compared to ours using the same datasets.

2. Related Work

2.1. Attribute-Based Encryption for Boolean Circuits

The first ABE systems were introduced in two flavors, *key-policy* [11] and *ciphertext-policy* [4], both of them supporting Boolean trees as access structures. Then, the problem of finding more expressive access structures arose. Boolean circuits, for example, cover a much larger range of access structures. Unlike a Boolean tree, a Boolean circuit does not limit the fan-out of its gates to one. Finding an efficient ABE scheme for Boolean circuit access structures is an important open problem in cryptography. Garg et al. [10] introduced the first ABE system for Boolean circuits. However, their system relies on multi-linear maps and the MDDH assumption, for which there is no known mathematical construction [1, 22]. Other approaches, such as [23, 12], offer constructions relying on secure and efficient mathematical primitives – bilinear maps. However, the decryption key could expand exponentially for some circuits.

2.2. Boolean Circuit Minimization

The problem of re-writing Boolean circuits in order to obtain a more compact form, with fewer gates is well-known in scientific literature as Boolean Formula Minimization. One of the most well-known algorithms is the Quine-

McCluskey Algorithm [19, 17]. However, the problem of Boolean Formula Minimization requires the functions to be given as truth tables. This is impractical for use in an ABE scenario since the Boolean truth table is exponentially larger than the access structure. In ABE we are given an access structure as a Boolean Circuit, and we need to minimize it without computing the truth table. Another algorithm for Logic optimization of Boolean circuits is Espresso [5, 6]. The main advantage of Espresso is that it uses various heuristics to minimize the circuits, which makes it far more efficient than Quine-McCluskey and allows it to run on higher inputs. It operates on multiple-valued and multiple-output Boolean functions described by min-terms, where each element is assigned a state – "True", "False", or "Don't care". Other newer Boolean Minimizers have been proposed, following similar ideas with [20, 8]. However, all these approaches differ from our scope, since their optimization of Boolean circuits is not the same as the one required for ABE systems. We give more details on this in Section 4.

2.3. Heuristic Optimizations in Cryptography

The problem of optimizing cryptographic schemes for obtaining better time performance has been approached before. However, most of the time, the optimization is very particular in order to be compatible with a specific cryptosystem. For example, [21] presents a survey on such optimizations for cryptographic systems with applicability in Wireless Sensor Networks. The heuristics compared include genetic algorithms and nature-inspired methods such as "Particle Swarm" or "Ant Colony". From our knowledge, there is no previous work on optimizing the decryption phase of ABE using heuristic approaches. The only work related to ABE systems is a very recent work [18] that applies a heuristic optimization to ABE for conversion between different types of underlying bilinear maps primitives. This is however a very different type of optimization, both in means and scope, compared to ours.

3. Prerequisites

When referring to a logical formula, we distinguish between variables and literals: Variables are symbols that can take values 1 or 0, while literals represent atomic logical formulas (a variable or its negation). Therefore, if a variable appears multiple times in a formula, it is counted each time as a distinct literal. For example, the formula $A \wedge B \vee A \wedge C$ has 3 variables and 4 literals.

A Boolean circuit is a Directed Acyclic Graph over a set of input wires, concluding to a single output wire, with internal nodes representing logical gates of type `AND`, `OR` or `NOT`. These gates may have fan-out greater than 1. A *monotone* Boolean circuit is a circuit without negation gates. A Boolean tree is a Boolean circuit where each node has a fan-out of a maximum of one.

Access Structures [3]. Let $\{p_1, \ldots, p_n\}$ be a set of parties. A collection $A \subseteq 2^{\{p_1,\ldots,p_n\}}$ is *monotone* if $(B \in A \wedge B \subseteq C) \rightarrow C \in A$. An access structure is a monotone collection $A \subseteq 2^{\{p_1,\ldots,p_n\}}$ of non-empty subsets of $\{p_1, \ldots, p_n\}$. Sets in A are called *authorized*, while sets not in A are called *unauthorized*.

3.1. KP-ABE Model

Key-Policy Attribute-Based Encryption scheme, as first described in [11], consists of four algorithms:

setup(λ) A randomized algorithm that takes as input the implicit security parameter λ and returns the public and secret keys (MPK and MSK).

encrypt(m, A, MPK) A probabilistic algorithm that encrypts a message m under a set of attributes A with the public key MPK, and outputs the ciphertext E.

keygen(C, MPK, MSK) This algorithm receives an access structure C, public and master keys MPK and MSK, and outputs the corresponding decryption keys DK.

decrypt(E, DK, MPK) Given the ciphertext E and the decryption keys DK, the algorithm decrypts the ciphertext and outputs the original message.

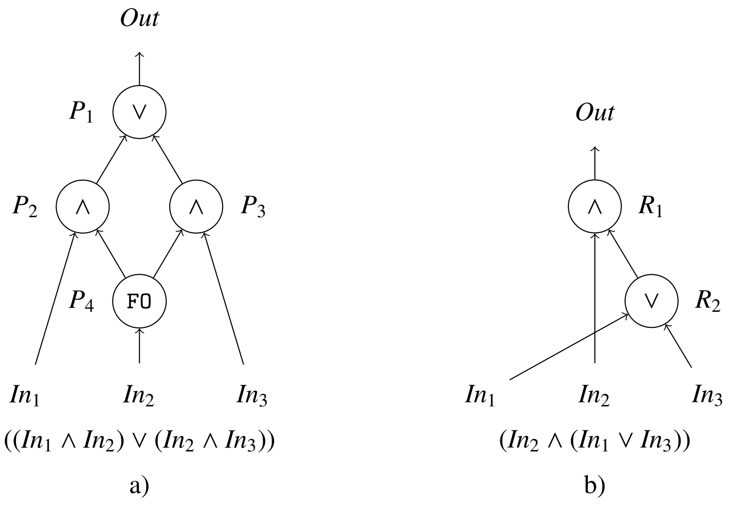


Fig. 1. Two equivalent Boolean circuits, alongside their equivalent logical formulas

3.2. *High-Level Description of Secret Sharing in KP-ABE for Boolean Circuits from [23]*

The *key-policy* ABE for Boolean circuits scheme from [23] uses bilinear maps as key components to the construction. Therefore, the running time of the four algorithms is strictly related to the number of pairing operations that are computed, since these are by far the most expensive ones. Therefore, our goal will be to minimize the number of pairings that are computed. Unfortunately, due to space limitations, we are not able to provide more details about the bilinear maps as mathematical primitives. However, they are not needed in order to understand our work, but rather just to acknowledge the fact that the pairing operations resulting from the bilinear maps are the most expensive in these ABE systems.

The *keygen* algorithm uses the Boolean circuit as an access structure in order to generate decryption keys. On the top of the Boolean circuit we will have an output node (we will refer to it as O_C), and each of the input nodes (In_i, where $i = \overline{1, n}$) of the circuit will have an attribute (labeled from 1 to n) attached to it. Then, a secret sharing technique will be applied to the circuit top-down, starting from O_C and ending in the input nodes. Each input node In_i will have some values associated with it. For each of these values, in the decryption phase, we must compute a *pairing* operation.

Due to the construction of the secret sharing technique from [23], the number of shares each attribute will receive in the end is equal to the number of paths from that input node (associated with the attribute) to the output node of the circuit. Therefore, the total number of pairings that will be executed in the decryption phase is equal to the total number of paths from the input nodes to the output node. Therefore, our goal is to find equivalent forms of the circuit such that the total number of paths is minimized.

We will define a cost function c for a circuit C: $c(C)$ should compute the number of shares the secret sharing technique in ABE is producing on C. This function can also be computed from the logical formula of the circuit: the number of literals in the formula represents the number of paths from the top of the circuit to the bottom since each literal corresponds to a leaf in the Abstract Syntax Tree of the logical formula.

One good example of two equivalent Boolean circuits is depicted in Fig. 1. The first circuit (Fig. 1a) leads in the secret sharing scheme from Ţiplea-Drăgan scheme [23] to a total number of 4 shares: one for In_1, two for In_2 and one for In_3. However, the equivalent circuit from Fig. 1b leads only to 3 shares. Therefore, the decryption time will be roughly 25% smaller if we use the second circuit.

4. Boolean Circuit Minimization for ABE

Since the problem of Boolean Minimization is well-studied, the first obvious choice while would be to try to use an existing algorithm. However, the existing algorithms optimize the circuit for constructing Programmable Logic Arrays (PLAs). Therefore, the input and output circuits will be in a format similar to DNF, which allows the easy construction of a PLA. Even if we convert the Boolean circuits from ABE's input to a logic minimizer such as Espresso, then we would have to also process the output. The DNF format in which these minimizers output the formula is an inefficient form for a circuit, w.r.t. the secret sharing scheme used in ABE. The DNF is the most uncompressed form a circuit can have. The logic minimizers are trying to optimize the number of gates used, while we want to minimize the number of literals in the logic expression associated with the circuit, or the number of paths from the output to the input nodes.

4.1. The Approach

We thought about how we can obtain, in general, equivalent logical formulas, starting from some given expression. We have observed three operations that we may apply to a monotone (without negations) logical formula to obtain equivalent ones. We will refer to these operations as *factorization*, *defactorization* and *absorption*:

1. *factorization* – Using the fact that `OR` is distributive under `AND` (and vice-versa), we can search for common factors inside logical formulas. For example, $(A \wedge B) \vee (A \wedge C)$ can be factorized into $A \wedge (B \vee C)$. Note that this operation always obtains a formula with a strictly lower cost, since it reduces the common factor.
2. *defactorization* – This is the inverse operation of *factorization*. We choose a conjunction, and we "split the parenthesis", resulting in a cross-product of the elements involved. For example, $A \wedge (B \vee C)$ can be defactorized into $(A \wedge B) \vee (A \wedge C)$. Note that this operation gives us an equivalent expression, but with a strictly higher cost.
3. *absorption* is the operation of eliminating 1s after the factorization process. For example, $A \vee (A \wedge B)$ can be factorized into something like $A \wedge (1 \vee B)$. However, the B term is actually "shadowed" by 1. Therefore, we can replace the initial formula with A, ignoring the term B. In our implementation, we have embedded the *absorption* in the *factorization* procedure. Therefore, throughout the rest of the paper, we will only refer to the first two operations – *factorization* and *defactorization*.

4.2. Implementation of Operations

In order to find a common factor inside multiple terms in a Boolean expression, we make use of the Abstract Syntax Tree (AST) associated with it. Let $\mathcal{T}_\varphi$ be the AST for some formula φ. For each node T_i, we will denote with $f(T_i)$ the logical formula associated with the subtree rooted in T_i, with $parent(T_i)$ the parent node of T_i, and with $children(T_i)$ the set of children nodes. Then, in order to get a common factor, we need to get two nodes T_1 and T_2 such that:

- $f(T_1) = f(T_2)$ – The formula $f(T_1)$ will be the common factor.
- $parent(T_1) \neq parent(T_2)$ – This is more of a consistency check. A well-formed formula should not have in the AST two siblings with the same formula, as one of them is clearly irrelevant.
- $parent(parent(T_1)) = parent(parent(T_2))$ – Nodes T_1 and T_2 should have a common grandparent in the AST, as shown in Fig. 2.

After this operation, the overall cost of the circuit will drop by $c(T_2)$, since we eliminate this part of the circuit.

There are some particular cases in the *factorization* and *defactorization* processes, but for the sake of simplicity, we decided to omit these cases here.

4.3. Hill Climbing

The factorization operation will always reduce the cost of the formula. It made us think it is suitable to be used in a Hill Climbing algorithm as we always move towards an optimum and at the end we will reach that optimum. However, the found optimum can be a *local* optimum and it means that we depend on the initial form of the formula.

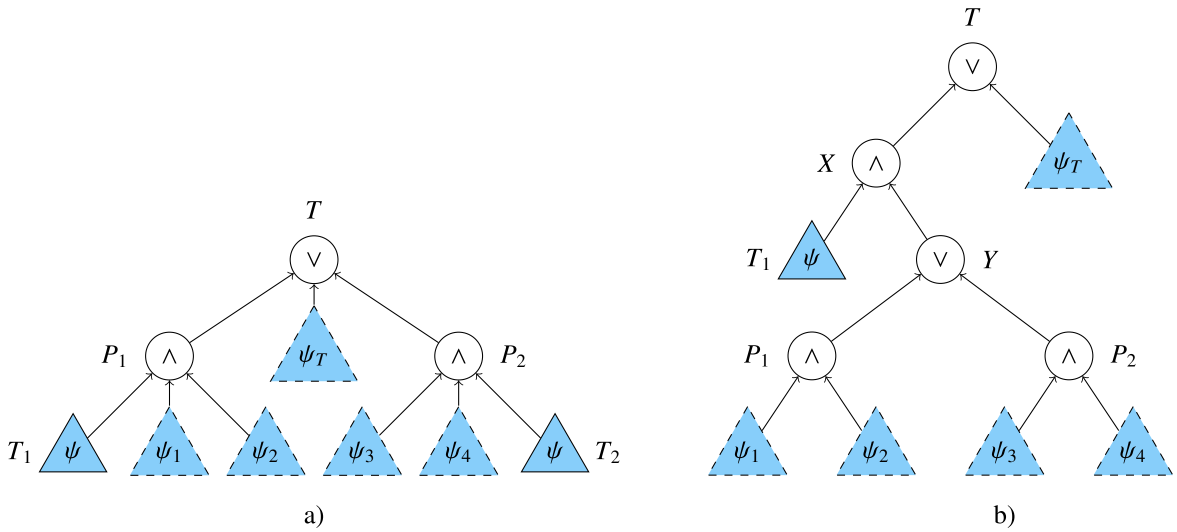


Fig. 2. Modification of an AST after a factorization

Therefore, until there is no possible factorization left, we randomly choose two nodes that can be factorized, and we apply the operation. When there is no factorization possible anymore, it means we have reached a local maximum.

4.4. Simulated Annealing

Simulated Annealing, proposed first time in [16], is a probabilistic method for finding the global minimum of a cost function. This method is inspired by the cooling of metals. In general, at each iteration of the algorithm, a new solution is considered. The probability that the solution is accepted varies with the temperature and the solution score. The higher the temperature, the higher the probability of accepting the solution. Similarly the higher the score, the higher probability of accepting the solution.

We have a temperature that starts with a value $t_{\max}$ and drops over time with a cooling rate c. The acceptance probability function used is $e^{-\Delta/t}$, where Δ is the difference between the neighbor formula cost and the current formula cost. The neighbor is obtained from the current formula by randomly choosing one operation between factorization and defactorization, and a random place where it can be applied in the circuit. The probability that we used to pick defactorization is 25%. Then, if defactorization is chosen, the probability to accept the new neighbor is given by the acceptance function. The algorithm looks as follows:

```
 1: $t \leftarrow t_{\max}$
 2: for $i \in \{1, 2, \ldots, L\}$ do
 3:     $operation \leftarrow$ defactorization if random(0, 1) $< d$ else factorization
 4:     choose a neighbor $u$ using $operation$
 5:     $\Delta \leftarrow \text{cost}(u) - current\_cost$
 6:     if $operation =$ defactorization then
 7:         accept $u$ with probability $e^{-\Delta/t}$
 8:     else
 9:         accept $u$
10: $t \leftarrow (1 - c) \cdot t$        # cooling
11: if $t > t_{\min}$ then
12:     goto 2
13: apply factorization until formula is not improved anymore
```

We used the following parameters: $t_{\max} = 100$, $t_{\min} = 10$, $c = 0.1$, $l = 25$. The values for $t_{\max}$ and t_{min} were chosen in this way because the cost of each logical formula from the dataset is between 3 and 100.

The factorization operation will decrease the cost, while defactorization will increase it. Using the simulated annealing algorithm, at the beginning we have a higher chance to accept defactorization in order to better explore the solution space. During this time, the rate of acceptance for defactorization decreases until we accept only factorizations, in order to improve the final solution as much as possible.

4.5. Custom Heuristic

Besides the classical Hill Climbing and Simulated Annealing heuristics we tried to create a new one that is meant to combine both *factorization* and *defactorization* operations but in a simpler way than Simulated Annealing. In this algorithm, we have $k_{\max}$ iterations. At each iteration, we choose either to factorize or defactorize the current formula. At iteration k the probability to choose defactorization is $\frac{k_{\max}-k}{5k_{\max}}$. It means that, at first iteration, we have a 25% chance to choose defactorize and it slowly decreases until reaching 0% at the last iteration.

Then we choose the formula with the smallest cost among all $k_{\max}$ iterations. Finally, we apply factorization on this formula until it can't be improved anymore.

4.6. Iterated Versions

In the simple Hill Climbing algorithm the solution converges very quickly to a local optimum. But, if at one step we choose to factorize a different pair of nodes, we can end up in a different local optimum, which can have a smaller cost.

In order to find better solutions, we run the Hill Climbing algorithm multiple times and choose the best solution among them. This gives us the opportunity to explore more and finally, we can end up in the *global* optimum. However, there is the possibility that the best equivalent formula (the *global* optimum) can't be obtained from the initial formula only by doing *factorizations*. There are many formulas where Iterated Hill Climbing gives good results but it can't find the global optimum regardless of the number of iterations we give.

For similar reasons, we have also constructed iterated versions of our other heuristics: Simulated Annealing and the Custom Heuristic. Finally, we have six algorithms – three main ones and three iterated versions of them.

5. Practical Tests

5.1. Dataset Description

We have generated four datasets, each of them with some particularities. The first three datasets consist of randomly generated logical formulas. Their numbers of variables and literals respect the values in Table 1. In this case, we tried to simulate real-world scenarios, where, when dealing with access structures, the most usual way of defining an access structure is by enumerating the minimum sets of participants which should have access to decrypt the data. Therefore, we constructed our formulas bottom-up by constructing formulas for minimum authorized sets, and then linking them together with AND or OR nodes.

	Variable count	Literal count
Dataset 1	20 – 25	20 – 40
Dataset 2	20	60 – 90
Dataset 3	25 – 35	160 – 200
Dataset 4	15 – 25	40 – 120

Table 1. Dataset parameters

Moreover, in the generation of all our datasets, we have run a special procedure called "trim", which ensures that the generated logical formulas do not have obvious design flaws. One such example could be the formula: $(A \land B) \lor (A \land B) \lor (A \land C)$. Here it is obvious that one of $(A \land B)$ could be removed from the expression.

The fourth test is created by taking a concrete example of a complex access policy that can arise in practical systems: comparison queries. There are several works on ABE with access structures supporting such queries [14, 4, 24]. When dealing with numerical numbers as attributes, the access structure may require to allow decryption for values smaller than some threshold value. Therefore, we created the fourth dataset with such cases. In order to perform a comparison query, a numerical attribute A in the range 0 to N can be split into $\log_2(N)$ smaller attributes, each of them representing a bit in A's binary representation. Then, we can create Boolean circuits which can handle multiple comparison queries. An example of such a Boolean circuit is seen in Fig. 3. There, A_0, A_1, A_2, A_3, A_4 represent attributes for the bits of the numeral value of A. The attribute A_i is True if and only if A has the ith bit set in its binary representation. The Access structures corresponding to such Boolean circuits could be something as `((Year <= 2022 AND Year >= 2020) OR (Year <= 1990)) AND (Month >= 2 AND Month <= 5)`.

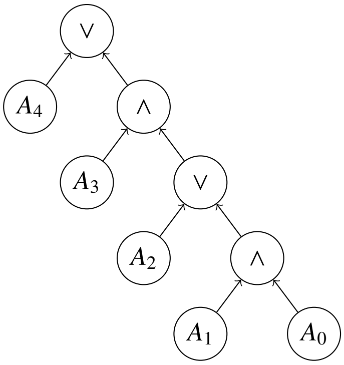


Fig. 3. Subcircuit for the comparison "$A \geq 11$" ($11 = 1011_{(2)}$)

5.2. Results

Since our heuristics are probabilistic, we want to compute the expected optimization we will obtain. Therefore, for each formula, we run each algorithm 16 times and we computed the average of the optimized percentage after each run. Also, we keep track of the maximum optimization obtained over all iterations. This value will be close to the upper bound of the algorithms. In Table 2 we show the *mean optimization* (MO), *best over iterations* (BOI), and *average running time* (ART) for all the formulas in each set. We have run all three algorithms presented above: HC stands for *Hill Climbing*, SA for *Simulated Annealing*, and CH for *Custom Heuristic*. We also have the iterated versions of these algorithms, denoted with IHC, ISA, and ICH, respectively.

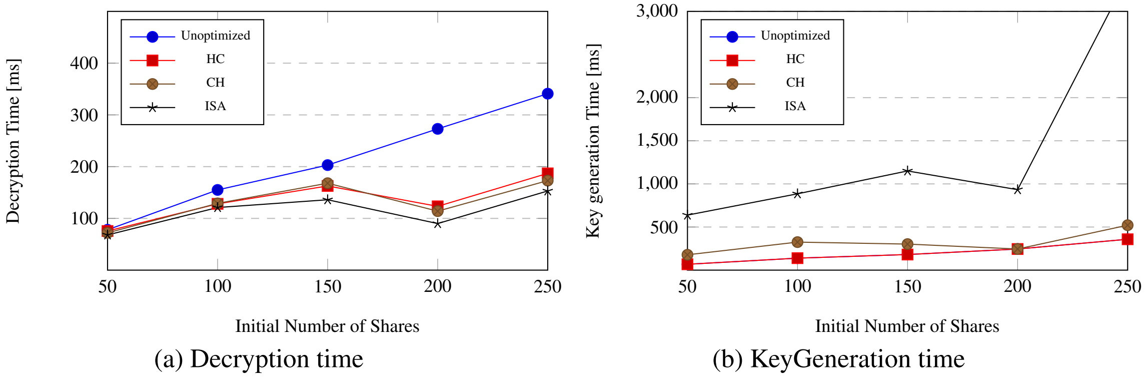
We see that Hill Climbing (HC in the table) is by far the fastest, and it also gives decent results for the first three datasets; however they are very low for the one with more practical formulas. Iterated Simulated Annealing (ISA) obtains some very good results on all sets, but its running time is the largest, making it even slower than Iterated Custom Heuristic. The latter beats (or at least ties) all the other algorithms on every set and every metric, and it also has a decently low running time.

Depending on the circumstances, one of the algorithms above could be chosen to optimize the ABE access structure, having a trade-off between running time and optimization. In order to better understand this trade-off, we have made the following experiment:

1. Generate 5 Boolean circuit access structures, with 50, 100, 150, 200, and 250 literals each.
2. For each of these access structures, run the HC, CH, and ISA optimization heuristics presented in this paper.
3. Construct an ABE system with each of these access structures (20 in total) and run the KeyGeneration and Decryption (after a previous Encryption) algorithms.
4. Add the running time of the optimization heuristic to the KeyGeneration algorithm.
5. Repeat steps 1-4 for 30 times and compute the mean value for each case.

	Dataset 1			Dataset 2			Dataset 3			Dataset 4		
	MO	BOI	ART	MO	BOI	ART	MO	BOI	ART	MO	BOI	ART
HC	15.0 %	16.3 %	0.00 s	35.1 %	41.9 %	0.00 s	42.6 %	56.5 %	0.01 s	4.8 %	7.2 %	0.00 s
IHC	16.3 %	16.3 %	0.08 s	42.0 %	42.1 %	0.34 s	56.5 %	56.5 %	2.03 s	7.2 %	7.2 %	0.11 s
SA	26.9 %	43.1 %	0.16 s	41.0 %	59.1 %	0.86 s	43.0 %	58.6 %	0.76 s	32.3 %	50.0 %	0.10 s
ISA	40.1 %	46.9 %	2.39 s	57.8 %	66.0 %	13.1 s	59.3 %	63.1 %	13.6 s	48.5 %	50.4 %	1.44 s
CH	24.8 %	44.0 %	0.13 s	35.8 %	61.8 %	0.38 s	39.8 %	59.4 %	0.42 s	20.8 %	47.8 %	0.14 s
ICH	43.0 %	47.8 %	2.74 s	61.3 %	68.6 %	7.50 s	60.6 %	64.9 %	7.82 s	48.5 %	50.4 %	2.94 s

Table 2. (Iterated) Hill Climbing/ Simulated Annealing/ Custom Heuristic results on each dataset



(a) Decryption time

(b) KeyGeneration time

Fig. 4. ABE performance tests

To ensure clarity and ease of interpretation of the plot in Fig. 4, we chose only three of our Heuristics for the second test, namely: HC, CH, and ISA. The first two choices were made based on the low running time while having similar optimization results. Lastly, we have also chosen an iterated version of one of our algorithms - ISA - to be able to evaluate the trade-off between running time and optimization in the iterated versions of the algorithms.

The results can be observed in Fig. 4: In a) we can see how the decryption time relates to the size of the original access structure. Since the Boolean circuit optimization should take place in the KeyGeneration phase, the second plot (Fig. 4b) presents the relation between the increase in KeyGeneration time and the access structure size. Here, the key generation time for the unoptimized access structure is the same with the HC optimization, since the running time for HC is negligible. It is obvious that the more aggressive the optimization process is, the more time the KeyGeneration takes. Depending on the circumstances, an inefficient KeyGeneration which produces efficient Decryption algorithms may be preferred. This is actually the case in applications where the key generation takes place in some server in the cloud, while the decryption is on systems with limited capabilities, such as mobile phones, or even nodes in Wireless Sensor Networks (Such an ABE system was proposed for example in [7]).

We can also observe that we have a drop in optimization for the access structure with 200 shares, compared to the one with 150 shares. This is due to the fact that we have run these tests with a single access structure for each number of shares, rather than generating multiple access structures and computing a median optimization. Also, we want to emphasize that the optimization of the Boolean circuit depends on the exact circuit. Two different circuits will have different potentials for optimization. From the two tests we ran, we can conclude that HC produces decent optimization while the computation overhead is negligible. CH and SA produce better optimization while having some small computational overhead. The iterated versions exploit the full potential of these algorithms but require a considerable additional amount of time.

5.3. *Library*

Our heuristics are publicly available on GitHub [15] as a library. It provides the possibility of running a specific algorithm over a logical formula and returns the best equivalent formula found. Also, another goal of the library is to be integrated with existing ABE systems in order to optimize their performance in real-world systems.

Benchmarking dataset. We have also provided `txt` files with our logical expressions which we used to test our heuristics. These can be used as references for further work on similar problems, in order to make relevant comparisons between similar algorithms with ours. The tests can be found in the `inputs` folder in our repository.

6. Conclusions

We have proposed multiple heuristic optimizations for minimizing monotone Boolean circuits, which, as we can see from the tests we ran, provide a substantial improvement in the circuit size. Each of our heuristics behaves differently in terms of optimization and running time, depending on the size and structure of the Boolean circuit. Since we drew our motivation from the problem of optimizing ABE systems for Boolean circuits, we emphasize that our optimizations translate into much smaller decryption keys and decryption times for these encryption systems. This can have an important impact on cloud systems that use ABE schemes to implement cryptographic access control over data: the heuristics will be applied only once when the decryption key is generated, and then, each time the respective decryption key will be used, the decryption time will be smaller compared to the scenario where we do not optimize it using one of our algorithms. For example, using the HC version of our heuristics will add almost no additional time overhead in the key generation process, while still providing an optimization between 7% and 40% in decryption time. We have compiled our work into a library [15] written in C++, which is publicly available for anyone to use. This could be easily integrated with an existing ABE implementation written in C++, such as OpenABE [2].

Furthermore, there is still space for even bigger improvements, by combining the heuristic with cryptographic improvements: At the moment, our algorithms are limited by searching for Boolean circuits that are equivalent to the original one. However, by using cryptographic strategies for improving secret sharing for parts of a circuit, we could offer our heuristics more space for searching for better solutions. This remains an interesting problem to be studied.

References

[1] Albrecht, M., Davidson, A., 2017. Are graded encoding scheme broken yet.

[2] B. Waters, M. Green, J.A.A.D.M.R., 2018. Openabe. https://github.com/zeutro/openabe.

[3] Beimel, A., 2011. Secret-sharing schemes: a survey, in: International conference on coding and cryptology, Springer. pp. 11–46.

[4] Bethencourt, J., Sahai, A., Waters, B., 2007. Ciphertext-policy attribute-based encryption, in: 2007 IEEE symposium on security and privacy (SP'07), IEEE. pp. 321–334.

[5] Brayton, R.K., Hachtel, G.D., Hemachandra, L.A., Newton, A.R., Sangiovanni-Vincentelli, A.L.M., 1982. A comparison of logic minimization strategies using espresso: An apl program package for partitioned logic minimization, in: Proceedings of the International Symposium on Circuits and Systems, pp. 42–48.

[6] Brayton, R.K., Hachtel, G.D., McMullen, C., Sangiovanni-Vincentelli, A., 1984. Logic minimization algorithms for VLSI synthesis. volume 2. Springer Science & Business Media.

[7] Chatterjee, S., Das, A.K., 2015. An effective ecc-based user access control scheme with attribute-based encryption for wireless sensor networks. Security and Communication Networks 8, 1752–1771.

[8] Coudert, O., Madre, J.C., Fraisse, H., 1993. A new viewpoint on two-level logic minimization, in: Proceedings of the 30th international Design Automation Conference, pp. 625–630.

[9] Drăgan, C.C., Ţiplea, F.L., 2015. Key-policy attribute-based encryption for general boolean circuits from secret sharing and multi-linear maps, in: International Conference on Cryptography and Information Security in the Balkans, Springer. pp. 112–133.

[10] Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B., 2013. Attribute-based encryption for circuits from multilinear maps, in: Annual Cryptology Conference, Springer. pp. 479–499.

[11] Goyal, V., Pandey, O., Sahai, A., Waters, B., 2006. Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM conference on Computer and communications security, pp. 89–98.

[12] Hu, P., Gao, H., 2017a. Ciphertext-policy attribute-based encryption for general circuits from bilinear maps. Wuhan University Journal of Natural Sciences 22, 171–177.

[13] Hu, P., Gao, H., 2017b. A key-policy attribute-based encryption scheme for general circuit from bilinear maps. IJ Network Security 19, 704–710.

[14] Ionita, A., 2022. Weighted attribute-based encryption with parallelized decryption. IACR Cryptol. ePrint Arch. , 605URL: https://eprint.iacr.org/2022/605.

[15] Ionita, A., Banu, D., Oleniuc, I., 2023. Circuit minimizer. https://github.com/Juve45/Boolean-Circuit-Minimizer-for-ABE.

[16] Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. science 220, 671–680.

[17] McCluskey, E.J., 1956. Minimization of boolean functions. The Bell System Technical Journal 35, 1417–1444. doi:10.1002/j.1538-7305.1956.tb03835.x.

[18] de la Piedra, A., Venema, M., Alpár, G., 2022. Abe squared: Accurately benchmarking efficiency of attribute-based encryption. Cryptology ePrint Archive .

[19] Quine, W.V., 1952. The problem of simplifying truth functions. The American mathematical monthly 59, 521–531.

[20] Sapra, S., Theobald, M., Clarke, E., 2003. Sat-based algorithms for logic minimization, in: Proceedings 21st International Conference on Computer Design, IEEE. pp. 510–517.

[21] Sasi, S.B., Sivanandam, N., 2015. A survey on cryptography using optimization algorithms in wsns. Indian Journal of Science and Technology 8, 216.

[22] Ţiplea, F.L., 2018. Multi-linear maps in cryptography, in: Conference on Mathematical Foundations of Informatics, pp. 241–258.

[23] Ţiplea, F.L., Drăgan, C.C., 2014. Key-policy attribute-based encryption for boolean circuits from bilinear maps, in: International Conference on Cryptography and Information Security in the Balkans, Springer. pp. 175–193.

[24] Zhu, Y., Hu, H., Ahn, G.J., Yu, M., Zhao, H., 2012. Comparison-based encryption for fine-grained access control in clouds, in: Proceedings of the second ACM conference on Data and Application Security and Privacy, pp. 105–116.