Improved Metrics for Obfuscated ICs

Mutian Zhu, Matthew French, and Peter A. Beerel

Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, California 90089
{mutianzh, pabeerel}@usc.edu

University of Southern California, Information Sciences Institute
Arlington, Virginia 22203
mfrench@isi.edu

***Abstract*—This paper focuses on combinatoric-centric metrics to measure the effectiveness of passive obfuscation techniques. Brute force measures estimate the expected number of guesses required for an adversary to obtain the obfuscation element keys, ignoring any correlation data that the attacker may have between obfuscation elements. We propose an improved formula for the expected number of guesses that takes into account such correlation data and show initial results that demonstrate such data can dramatically lower the expected number of guesses. We then describe the computational challenges associated with computing this metric and an approximation approach that may help mitigate this issue.**

I. Introduction

The globalization of the integrated circuit market has led to DoD losing secure, trusted, access to state of the art fabrication nodes. In order to leverage state-of-the-art capabilities provided by commercial foundries, DoD needs solutions which ensure 1) that the IP fabricated offshore is not reverse engineered, and 2) that the risk of hardware Trojan insertion is eliminated through either prevention or detection and mitigation. Research and development in these areas has been active, but uncoordinated, resulting in a wide variety of passive and active obfuscation techniques which address specific pieces of circuitry or stages in the IC development and fabrication process, yet no agreement on metrics, making it difficult to assess the ability of obfuscation IP to protect Critical Program Information (CPI) and support mission requirements.

One approach for measuring passive obfuscation is to utilize a combinatoric centric metric which represents the work load an adversary has to perform to decode in order to begin their attack (steal IP, bypass active obfuscation, insert a Trojan etc). Setting defenses, or work load, to be much greater than that of which an attacker can achieve over a desired window of operation is a concept that has been well proven and established in the selection of cryptographic key lengths [1], [2]. Setting the key length beyond the computational reach of an adversary for the period of time desired also lends itself well to establishing mission oriented metrics, defining theoretical upper bounds, and measuring achieved performance.

DISTRIBUTION A. Approved for public release: distribution unlimited.
Peter A. Beerel is also Chief Scientist at Reduced Energy Microsystems.

In hardware reverse engineering and IP leakage though, circuits contain additional metadata (cell type, placement, drive strength etc), which may provide deobfuscation clues and shorten the effective key length or adversary work load. Adversaries are constantly innovating new attack types, not anticipated by defenders. Decoding one obfuscated element, may provide clues to an attacker on how to decode other elements obfuscated using the same technique. There may also be inadvertent similarities in different types of obfuscation. These correlations can drastically reduce the combinatoric work load that an attacker would actually encounter. This paper's first contribution is to define an improved expected number of attacks needed by an attacker given correlation information about obfuscation elements. It then describes the computational challenges associated with computing this metric and an approximation approach that may help mitigate this issue.

II. Expected Number of Attacks

Our model of a circuit is that it has n obfuscation elements, also referred to simply as gates in this paper, that with-out-loss-of-generality can have one of 2 possible functions. The adversary's job is to guess which of the 2^n possible combinations is the correct function by testing one combination at a time. We assume that each obfuscation gate takes on each type with equal probability. So a combination can be modeled by a sequence of identical distributed Bernoulli random variables $X_i (i = 1, 2, ..., n)$ with $p = 0.5$. The joint distribution of all X_i is the probability of each combination, referred to in this paper as a guess or key, being correct.

A. Brute Force Attack

If an attacker chooses one combination randomly each attack, the probability of success at the first attack is $\frac{1}{2^n}$, and the probability of success at the k-th $(k > 1)$ attack is $\frac{2^n-1}{2^n}\frac{2^n-2}{2^n-1}\cdots\frac{2^n-k+1}{2^n-k+2}\frac{1}{2^n-k+1}$ which also gives $\frac{1}{2^n}$. So the expected number of attacks K is:

$$E[K] = 2^{n-1} + 0.5 \tag{1}$$

B. Correlation Driven Attack

If the correlation coefficients ρ_{ij} between all pairs of X_i and X_j are known, the attacker can obtain a better estimation of the joint distribution. The optimal attack strategy would then be to sort the probabilities in descending order $p_{s1}, p_{s2}...p_{s2^n}$ and start from the combination with highest probability to lowest probability. The expected number of attacks K would then be:

$$E[K] = \sum_{i=1}^{2^n} i \times P_{si} \tag{2}$$

C. Entropy Lower Bound

If we know the entropy $h(P) = -\sum_{i=1}^{2^n} P_i log(P_i)$ of the joint distribution, we can also directly give a lower bound of the expected number of attacks [3]:

$$E[K] \geq \frac{1}{4} 2^{h(P)} + 1 \tag{3}$$

III. Solving for the Joint Distribution

Given ρ_{ij}, we can solve for the marginal distribution of a pair of gates: $P_{X_i X_j}$. Each marginal probability can be written as a linear combination of joint probabilities: $\sum_{i=1}^{2^n} a_i P_{X_1 \cdots X_n} = P_{X_i X_j}$. We have $\binom{n}{2}$ pairs which can give us a system of linear equations:

$$M \times \underline{P} = \underline{b} \tag{4}$$

Here, M is a $4\binom{n}{2} \times 2^n$ binary parameter matrix that has one row for each possible combination of pair values and one column for every possible guess. For each row, there is a one in the columns associated with the guesses that include that specific combination of pair values and a zero in all other columns. $\underline{P}$ is the vector of 2^n joint probabilities and $\underline{b}$ is the vector of marginal probabilities, These linear equations will give us a family of solutions. According to Maximum Entropy principle [4], we should choose the solution with maximum entropy. In particular, if an attacker assumes the distribution with maximized entropy to build the attack sequence, then the maximum entropy distribution provides the expected number of attacks. In fact, this is the largest number of attacks among distributions that satisfy Equation 4 because the maximizing entropy gives the distribution that is closest to uniform. In reality the underlying distribution could have more diversity so the expected number of attacks can be smaller.

IV. Approximation Algorithm

Unfortunately, solving Equation 4 is computationally challenging because it involves solving the probability of each combination. Having n obfuscation elements means we have 2^n variables to solve. This motivates developing a faster means of obtaining an approximation of the expected number of attacks. One idea for an approximation is that since each linear equation is a sum of joint probabilities, we can quickly eliminate an equation by assuming that all variables in it have identical values. In particular, we propose to use this estimation to eliminate the equations with smaller sums quickly and solve the remaining equations to get more precise probabilities for those most likely combinations. This section first develops this idea symbolically using binary expressions and then describes a preliminary implementation in python using integer sets.

A. Logical Representation

1) Group: We can describe all the keys that are summed in an equation through a binary expression. For example, equation $P_{X_1 X_2}(0,0) = \sum P_{X_1 \cdots X_n}(0,0,x_3,\cdots,x_n) = b_0$ is the sum of probabilities of all keys with the first and second digits equal to zero. We can use a binary expression to describe this group of keys:

$$G_0 = \neg 1 \wedge \neg 2 \tag{5}$$

2) Operations: Two main operations needed in this approximation approach are finding the overlapped keys between two groups and the remaining keys in a group after another group has been removed from consideration. They both can be implemented by the logical operations upon the binary expressions of two groups. The intersection between two groups is simply the $\wedge$ operation between the binary expressions of them. For example, given $G_0 = 1 \wedge 2$ and group $G_1 = 3 \wedge 4$, their intersection is:

$$G_0 \cap G_1 = 1 \wedge 2 \wedge 3 \wedge 4 \tag{6}$$

If group G_0 has been removed from consideration, then the binary expression of remaining keys in group G_1 can be represented by $G_0 \cap (G_1)^C$:

$$G_0 \cap (G_1)^C = (1 \wedge 2 \wedge \neg 3) \vee (1 \wedge 2 \wedge \neg 4) \tag{7}$$

3) Number of keys In a Group: We need the number of obfuscation keys in a group given its binary expression to calculate probabilities and expectation.

For a binary expression that only contains $\wedge$ operation, such as $1 \wedge 2 \wedge \cdots k-1 \wedge k$, the number of obfuscation keys N in this group is

$$N(1 \wedge 2 \wedge \cdots k-1 \wedge k) = 2^{n-k} (k \leq n) \tag{8}$$

where n is the total number of gates and k is the number of gates in this expression. For an expression with mixed $\wedge$ and $\vee$ operations, we can use the inclusive-exclusive law from probability theory:

$$N(\cup_{i=1}^{n} B_i) = \sum_{i=1}^{n} N(B_i) - \sum_{i<j} N(B_i \cap B_j) + \cdots (-1)^{n+1} N(\cap_{i=1}^{n} B_i) \tag{9}$$

where B_i is a binary expression that only contains $\wedge$ operation.

B. Implementation In Python

1) Symbolic Representation: We implemented this algorithm in Python by symbolically representing each gate as an integer, and representing the binary expression of a group of keys using a list of sets $\left[B_1, B_2, B_3, \cdots\right]$, where B_i is a set of integers which represents the $\wedge$ operation of gates.

For example, $\left[\{1,2,-3\},\{1,2,-4\}\right]$ is the representation of Binary Expression 7 described above. The positiveness and negativeness of an integer indicates the type of a gate to be 1 or 0. So, in summary, the $\wedge$ operation is implicitly represented by sets in python, and the $\vee$ operation is represented by lists of such sets.

2) Data Structure: A group of keys $G[i]$ can be represented by a three tuple $\langle A, N, L\rangle$, where $G[i].A$ is the average probability of each key in group $G[i]$, $G[i].N$ is the number of keys in this group, and $G[i].L$ is the symbolic representation of group $G[i]$'s binary expression. This tuple contains all the information we need to perform the reduction of groups and calculate the expected number of attacks. All the groups that have been removed are put in the list R and the remaining groups are in the list G. The pseudocode of a preliminary version of this approximation algorithm is described in Algorithm 1.

C. Algorithmic Optimization

1) Optimized Resorting: The algorithm iteratively chooses the group with smallest average probability to remove from G. After a group is removed from G, we must remove any overlapping part from the remaining groups and resort them so that we can choose the group with smallest average probability in the next iteration. Unfortunately, if we do this naively the complexity is very high.

To mitigate this complexity issue we observe that the updated average probability of group $G[i]$, denoted as $G[i].A$, will always grow after updating. This is because the removed keys are always assigned with smaller average probability than $G[i].A$ since we always choose the group with smallest average probability to remove first. Hence the new average, denoted as $G[i]'.A$, is greater than $G[i].A$. So if we choose group $G[i]$ with original average probability $G[i].A$ being the smallest, and has new average $G[i]'.A$ after updating, then the groups that with original average probability larger than $G[i]'.A$ do not need to be concerned. We only need to update the groups that have original average probability within the range of $\left[G[i].A, G[i]'.A\right]$ and pick the one that has the smallest average probability after updating. By applying this idea we reduce the number of times we resort and number of groups to resort in each iteration. This is captured in the while condition (line 13) in Algorithm 1.

2) Simplified Updating: Notice that we may update several groups in each iteration. The list G is used to store the original $G[i]$'s. If $G[i]$ is updated in the current iteration, there is a new list called $Cache$ to store the updated $G[i]$ at $Cache[i]$. In the next iteration if $G[i]$ still needs to be updated, we can use $Cache[i]$ to update with only the newly added group in R, which is faster than updating G with all the groups in R. Besides, if $G[i]$ is removed, $Cache[i]$ will be removed as well so that each group has same index for its original version in G and updated version in $Cache$. We applied this idea in our python implementation but for simplicity did not include it in the pseudocode shown in Algorithm 1.

Algorithm 1 Iterative Approximation Algorithm

```
1:  procedure APPROXIMATE EXPECTATION(G)
2:      G, R = REDUCE GROUPS(G, 0)
3:      P = SOLVE FOR REMAINING KEYS(G)
4:      E = COMBINE RESULTS(P, R)
5:      Return E
6:  end procedure
7:  procedure REDUCE GROUPS(G, R)
8:      removed = 0
9:      while 2^N - removed > threshold do
10:         Sort G in increasing order of G[i].A
11:         smallest = 0, i = 1, minavg = UPDATE(G[0],R).A
12:         range = range of updates needed
13:         while i < len(G) and G[i].A is within range do
14:             newavg = UPDATE(G[i],R).A
15:             if newavg < minavg then
16:                 smallest = i, minavg = newavg
17:             end if
18:             i = i + 1;
19:         end while
20:         G[smallest] = UPDATE(G[smallest],R)
21:         removed = removed + G[smallest].N
22:         Append G[smallest] to R;
23:         Delete G[smallest] from G
24:     end while
25:     for the rest of groups G[i] in G do
26:         G[i] = UPDATE(G[i],R)
27:     end for
28:     Return G, R
29: end procedure
30: procedure SOLVE FOR REMAINING KEYS(G)
31:     Form Eq. 4 for groups in G
32:     Find sol. minizing Euclidean dist. to uniform distr.
33:     Shift probabilities so that they are all non-negative
34:     Return P
35: end procedure
36: procedure COMBINE RESULTS(P,R)
37:     Sort P in descending order
38:     E = 0, start = threshold
39:     for i = 1 to threshold do
40:         E = E + i * P[i − 1]
41:     end for
42:     for i = len(K)-1 to 0 do
43:         end = start + R[i].N, span = start + 1 + end
44:         E = E + R[i].A * [(span∗R[i].N)/2]
45:         start = end
46:     end for
47:     Return E
48: end procedure
49: procedure UPDATE(G[i], R)
50:     Store a copy of G[i] in G_cp
51:     for each the group R[j] in R do
52:         Remove overlapped keys and update G_cp
53:     end for
54:     Return G_cp
55: end procedure
```

D. Challenges

There are two challenges in this preliminary version. The first one is that there can exist groups that have no remaining keys left but still have group probability non-zero after updating. In other words, there can exist equations like $0 + 0 + \cdots + 0 = b_i$ where $b_i > 0$ in our system of linear equations. In our implementation, we skip these bad groups and choose the next group. But, in the future, we will try to make better use of them.

The second challenge is related to solving for the probabilities of the remaining keys. In many cases, Equation 4 we generate from the updated G does not have a solution. This means we cannot use the convex optimization method to solve for the distribution with maximized entropy. Instead, we use the pseudo-inverse to solve the problem. In particular, we observed that given the sum of the probabilities of remaining keys, if the keys follow the uniform distribution each key will have probability of sum of probabilities divided by the number of remaining keys. We denote this distribution as $\underline{P_u}$. If we multiply $\underline{P_u}$ with the reduced parameter matrix M from Equation 4, we get the new equation:

$$M \times \underline{P_u} = \underline{b_u} \tag{10}$$

Then we subtract Equation 4 by Equation 10 and get:

$$M \times (\underline{P} - \underline{P_u}) = \underline{b} - \underline{b_u} \tag{11}$$

Consequently, $M^+(\underline{b} - \underline{b_u})$ is the least square solution with minimum norm, where M^+ is the pseudo-inverse of M. Hence the solution P_{min} that minimize the Euclidean distance from the uniform distribution P_u is:

$$\underline{P_{min}} = M^+(\underline{b} - \underline{b_u}) + \underline{P_u} \tag{12}$$

However, the problem is that $\underline{P_{min}}$ can contain negative probabilities which are not realistic. So we shift all the probabilities in $\underline{P_{min}}$ by the same amount to ensure that every probability is non-negative.

This shifting is not ideal, but we believe that it has limited effects for two reasons. First, compared to the total number of keys, the amount of remaining keys is small and they have smaller weights in calculating the expectation since they are at the beginning of the attack sequence. Secondly, since we neglect all the groups with zero keys but non-zero group probability, we have reduced the total amount of probabilities for our distribution and thus shifting $\underline{P_{min}}$, to some degree, compensates for this.

The results of these challenges is that we have yet to prove whether or not this approximation algorithm provides a proper bound for the expected number of attacks or be able to bound its accuracy. Addressing this issue is part of our future work.

V. Simulations and Comparisons

We implemented all described approaches in python running Anaconda on a 4-core Intel i5-4210M CPU at 2.6GHz. The input of the program is the correlation coefficient matrix $corr$. The program generates the matrix M and marginal probabilities $\underline{b}$ and finds a solution of the system of linear equations that maximizes the entropy of the joint distribution. It then computes the expected number of guesses using both the maximum entropy lower bound (Eq. 3), as well as the direct calculation shown in (Eq. 2), the latter requires us to rank order the probabilities $\underline{P}$.

We evaluate the approach by randomly creating the correlation matrix $corr$ by calculating the sample covariance matrix of Bernoulli random vectors obtained via a random affine transformation of Gaussian elementary variables. For each number of obfuscation elements, we run each algorithm 5 times and take the average of results. We tested two python libraries to find the distribution with maximum entropy without group reduction. The first is a basin hopping algorithm [5] and the second is a convex optimization solver [6]. In both cases, we pre-process the linear system of equations to remove redundant constraints via singular valued decomposition.

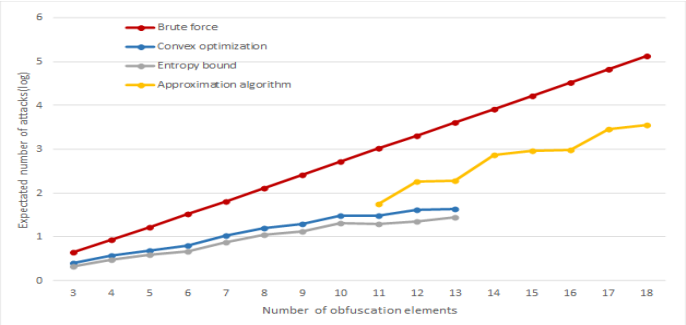
The results shown in Fig. 1 suggests that knowledge of the correlations can massively reduce the expected number of attacks compared to the brute force attack. Note that the expected number of attacks is always larger than the lower bound given by the entropy method. The expected number of attacks calculated by basin-hopping algorithm and convex optimization method are very close, so their results are overlapped on the log base graph in Fig. 1(a). For this reason, we only plot the results of the convex optimization method in the graph.

Both the basin hopping and convex optimization have a threshold for number of obfuscation elements where the running time has a steep increase. In particular, the basin hopping algorithm exceeds 1 hour of run-time after $n = 8$ and the convex optimization based algorithm exceeds 1 hour of run-time after $n = 13$.

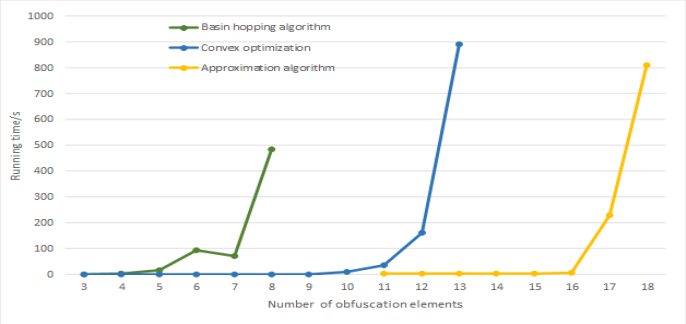
We tested the approximation algorithm for case when the number of obfuscation elements exceeds 11. We set the threshold for the number of remaining keys to be 1024 for stop removing groups. In terms of run-time complexity, the iterative approximation algorithm does significantly better than the convex optimization exceeding 1 hour of run-time only after $n = 18$. This demonstrates that the approximation algorithm has some promise. However, given we are interested in analyzing circuits with hundreds if not thousands of obfuscation elements, the algorithm must be further refined to minimize its complexity and maximize the number of obfuscation elements it can handle. One idea is to store the symbolic representation and perform logical operation through more efficient data structures such as a binary decision diagrams (BDDs). Alternatively, we may need to apply higher level divide and conquer approaches that decompose circuits with large number of obfuscation elements into smaller subcircuits that are analyzed independently with this approach, combining the results using more conservative techniques.

VI. Conclusions

This paper proposes an advanced combinatoric-centric metric to measure the effectiveness of passive obfuscation tech-



(a) Expected number of attacks



(b) Running time

Fig. 1: Simulation result for up to 18 obfuscation elements

niques. In particular, if an attacker follows strategy in this paper, the proposed metric represents an upper bound to the expected number of guesses needed by an attacker that has correlation information about the obfuscation elements. The paper demonstrates the dramatic reduction in guesses that correlation may yield as well as describes the computational challenges associated with computing the metric as well as explores an approximation approach that may help mitigate this challenge.

REFERENCES

[1] RSA Labratories, "Whitepaper: A cost-based security analysis of symmetric and asymmetric key lengths," 2017, https://www.emc.com/emc-plus/rsa-labs/historical/a-cost-based-security-analysis-key-lengths.htm.

[2] A. Lenstra and E. Verheul, "Selecting cryptographic key sizes," *Journal of Cryptography*, 2001.

[3] J. L. Massey, "Guessing and entropy," in *Proc. of the IEEE International Symposium on Information Theory*, 1994, p. 204.

[4] E. T. Jaynes, *Probability theory: The logic of science*. Cambridge University Press, 2003.

[5] "SciPy open-source python-based ecosystem," http://scipy.org/, accessed: 10/01/2017.

[6] "CVXOPT: python software for convex optimization," http://cvxopt.org/, accessed: 10/01/2017.