A Secure Design Methodology to Prevent Targeted Trojan Insertion during Fabrication

Arjun Suresh, Siva Nishok Dhanuskodi, Daniel Holcomb
University of Massachusetts Amherst, MA, USA
{arjunsuresh, sdhanusk, dholcomb}@umass.edu

***Abstract*—Hardware Trojans are malicious modifications made to a circuit for the purpose of altering its behavior. If Integrated Circuit (IC) fabrication is outsourced to an untrusted foundry, learning the correspondence between the layout and logical design nodes can enable that foundry to strategically choose target sites for adding Hardware Trojans to the layout. Split manufacturing mitigates this threat by printing only FEOL layers in the untrusted foundry, and then completing the design in a trusted facility. In this work we propose a new synthesis-based approach for k-secure personalization in split manufacturing.**

Based on the design, cell library, and user-specified security level k, our approach automatically chooses a subset of library cells for technology mapping, and then scalably partitions the mapped design to identify security-critical wires that must be withheld from the untrusted foundry. We demonstrate our approach by implementing four publicly available benchmark designs in a 15 nm open cell library across a wide range of security levels from 10-100,000. We show that our method appropriately tailors library cell choices according to design and security level, and that the secure partitioning algorithm scales to thousands of cells and beyond. Designs created with our method achieve a high level of security even while withholding as little as 9% of wires from the untrusted foundry. Our method is furthermore able to explore tradeoffs by generating a set of equally-secure designs that are Pareto-optimal in area versus number of wires withheld.

I. Introduction

Most IC design houses employ a fabless business model, and outsource manufacturing to massive contract foundries that can better exploit economies of scale in semiconductor fabrication. Even security-critical designs rely on outsourced fabrication to gain access to leading process nodes, which makes protecting circuits an important concern. Malicious circuits might be inserted by actors at an untrusted foundry to steal sensitive data, gain privileged system access through hardware backdoors, or perform a Denial-of-Service attack. Tampered circuits are suspected to have caused failure of radar systems and also leaked sensitive military communications [1], [2]. Tamper prevention has been explored widely by the research community and government agencies [3].

In an attempt to prevent tampering, previous works have investigated split fabrication [4], [5], where an untrusted foundry partially fabricates a chip, which is then completed at a trusted facility, typically by adding its upper metal layers. This attempt suffers from overheads, specifically area and delay overheads caused by top layer routing. It is thus imperative to provide design security while minimizing overhead, and this is what our work is based on.

In this work we present an automated synthesis flow which makes judicious use of programmability while providing quantifiable security against targeted attacks by an adversarial foundry.

Our specific contributions include:

- **Automated synthesis flow** which is technology and CAD tool agnostic to provide quantifiable security against targeted manipulations
- **Secure technology mapping** of circuits by heuristic pruning of cell libraries
- **Integration of new hardware primitives** with limited programmability to mitigate area overheads while ensuring that cell choices do not deanonymize the design.
- **Scalable isomorphic clustering** that is secure-by-construction and can preserve security while withholding as few as 9% of wires from foundry, depending on design and required security level.

II. Related Work

A. Hardware Trojans

Hardware Trojans are a threat arising from untrusted entities in the design or manufacturing process, which often involves a contract foundry, EDA tools [6], and 3rd-party IP [7]. The use of contract foundries that reside in foreign countries can increase concern about exposure to Trojan insertion [8]. When triggered, a Trojan can carry out attacks such as snooping data [9], inducing or enhancing side channels [10], injecting a fault [11], or retrieving cryptographic keys [12]. The focus of this work is based on countermeasures which aim at preventing Trojan insertion by preventing the attacker from knowing the functionality of the design well enough to insert stealthy Trojans.

B. Split Manufacturing Implementations

Split fabrication techniques to hide parts of design have been studied in literature. Wires have been lifted to top layers on the basis of k security [4] and VLSI testing principles [13]. Sengupta et al. leverage a greedy graph coloring strategy to hide connectivity information [14]. Circuit similarity based partitioning [15] and MILP formulation with capability to add dummy logic [16] have also been proposed. Proximity [17] and Simulated Annealing [18] based attacks have been mounted to guess hidden wire connections and reconstruct netlists. The lifted wires can be susceptible to probing [19]. Patnaik et al. perform partition the original design into sub-modules with at least k-instances of each type, but limit their technology mapping to only a few predefined cells [20].

C. Alternatives to Split Fabrication

An alternative to split manufacturing is to use reconfigurable logic to hide critical design information [21]–[23], but these solutions come with significant area/power cost compared to an ASIC design. Previous works have used LUTs [21], [24], coarse

grained reconfigurable architectures [25] and programmable transistor fabrics [26] to withhold design information from foundries.

III. Methodology

A. Threat Model

Our threat model considers the adversary to be an untrusted foundry that intends to modify circuitry in a targeted manner to change its behavior. We assume the attacker has access to the design netlist obtained through reverse engineering, public knowledge of the implemented algorithm or through insider knowledge from a malicious entity in the design company. Example hardware Trojans in open literature that fit our attack model have enabled leaking of cryptographic keys [27], privilege escalation in processors [28] and bypassing of critical security functions [29].

B. Primer on k Security

We adopt the k-security metric first formalized by Imeson et al. [4] to quantify resistance to targeted Trojans. The output of a primitive is k-secure if there are at least k instances of it that are indistinguishable from each other. The entire design is k-secure if each primitive type is k-secure. The motivation for k-security is that a targeted attack will have only a $1/k$ chance of modifying the correct node, while indiscriminately attacking multiple nodes raises the chances of being caught. This metric has been widely adopted and used by various others in literature [16], [18], [20].

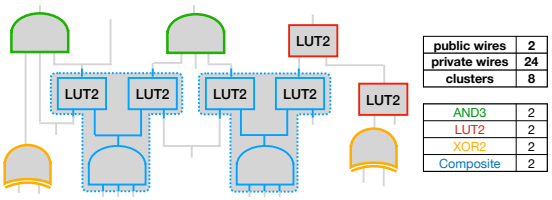


Fig. 1: Circuit with security level of k = 2

The netlist presented in Fig 1 has a security level of k = 2, meaning that there are at least 2 indistinguishable instances of every primitive in the circuit. The circuit contains two composite clusters in blue each with a single publicly visible wire. If the lookup tables in the clusters were replaced with different fixed-function cells, then the clusters would not be indistinguishable, and so their associated wires would need to remain hidden to preserve the security level of k = 2.

C. Our Secure Synthesis Approach

If the foundry can associate a logic signal in a design to a specific net in the layout, then an attack might target that signal by modifying its layout. Conversely, if the foundry can only know that a signal is implemented by one of k layout nets, which is referred to as being k-secure (Sec. III-B), then the ambiguity is a barrier against targeted tampering. Ensuring ambiguity to prevent targeted attacks is our goal in this work.

Our approach, illustrated in Fig. 2, uses two phases to create deployable chips. Phase 1 is the design process that generates a layout for fabrication at an untrusted foundry, and also generates configuration information that will later be used to complete it. Phase 2, after untrusted fabrication, completes the circuit in a trusted facility by applying its configuration. The deployed circuit, after phase 2, is free from targeted manipulations.

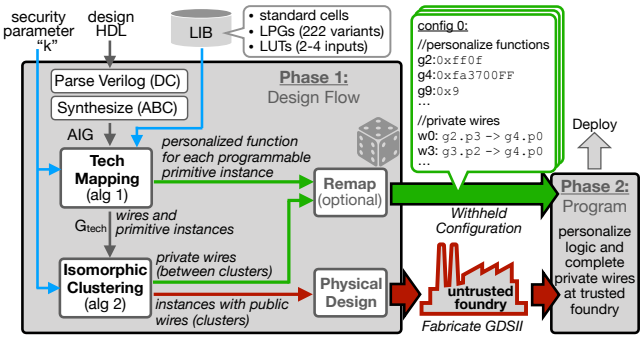


Fig. 2: Proposed Methodology

In split manufacturing, security against tampering requires that information about logic and connectivity which is known to the foundry will not deanonymize how the design maps on to it. The foundry is allowed to know the set of one-or-more functions that each primitive can realize, but does not know which specific function from the set will be selected in phase 2. Likewise, the foundry is allowed to know a subset of connections that are denoted as **public** wires, but does not know how to connect the **private** wires which will be added in phase 2. The upper metal connections made by the trusted BEOL facility in phase 2 configure the logic functions of the primitives and physically connect private wires.

Our work in this paper proposes a synthesis flow for phase 1. The flow comprises technology mapping and isomorphic clustering, and is compatible with any cell library that has programmable or fixed-function standard cells. The flow decides what cell types to use, and which wires must remain private, in order to preserve k-security against targeted attacks from the foundry. Due to page limits, the physical design steps of metal programming and randomized placement are considered as out of scope for the current paper, but can be addressed in a subsequent extension.

We propose and incorporate two techniques into the traditional split manufacturing process to reduce design overheads while maintaining k security:

1) Reducing Area - Limited Programmability Gates: Fixed-function standard cells and fully-programmable lookup tables represent the two extremes on a spectrum of representing a logic function, in terms of area and number of functions which can be implemented. We additionally explore a set of primitives that are a compromise between the two extremes, which we denote as Limited Programmability Gates (LPGs). LPGs are larger than ASIC standard cells and each can implement multiple functions, but they are smaller than lookup tables and cannot implement all possible 4-input functions. Our flow can accommodate LPGs that implement arbitrary subsets of the 65,536 4-input Boolean functions. We design our LPGs such that each one can implement all of the functions in an NPN-equivalence class. Two Boolean functions are NPN-equivalent if one of them can be derived from the other by any combination of negating inputs (N), permuting inputs (P), and negating output (N). The set of 65,536 4-input Boolean functions map to 222 disjoint NPN-equivalence classes [30]. We generate a cell for each of

the 222 LPG classes using synthesis and PnR, which provides the area cost of each LPG.

2) Mitigating Delay Overhead - Isomorphic Clustering: As noted in Fig. 1, replacing fixed-function standard cells with more generic lookup table logic can support creating composite clusters. The internal wires of the cluster are public, and can be known to the foundry while maintaining required security level. Increasing the number private wires will result in more randomized cell placement in phase 2, which will negatively affect timing and power of the fabricated design [31]. Thus, it is essential to reduce the number of wires to be withheld from the untrusted foundry. In our approach, because the clusters are public, they can be placed together using short within-cluster wires for improvements in speed or power after phase 2.

D. Evaluation and Implementation

We evaluate phase 1 of our synthesis flow using the 15 nm FreePDK standard cell library provided by the Silicon Integration Initiative [32]. Area costs of the standard cells are directly from the library, and area costs of LPG and LUT primitives are from implementing them using combinations of the 15 nm library cells. Our primary benchmarks are four design blocks that deal with sensitive data, namely ECC point scalar multiplication [33], a Reed-Solomon decoder [34], Keccak [35] hashing algorithm, and an SDRAM controller [36]. Design Compiler is used for parsing the benchmarks, and technology mapping is performed iteratively by ABC [37]. Isomorphic clustering is performed on the mapped netlists. Our clustering algorithm is written in C++. The source code for Algorithms 1 and 2 is publicly available at [38].

IV. Technology Mapping

The technology mapped circuit must have at least k instances of each primitive. Alg. 1 repeatedly maps the synthesized design while pruning the cell library until reaching the security goal. At each iteration, if the mapped design (line 3) is not yet secure, the rarest primitive is pruned from the library (line 11) for the next iteration. The pruning continues until the technology mapped circuit uses at least k of each primitive (line 13). Tab. I shows the size of the library that is generated by the algorithm, for different values of k on each benchmark. Our flow can choose cells from libraries containing standard cells or programmable elements such as LPGs and LUTs.

Fig. 3 shows data from iterations of the technology mapping algorithm for SDRC_top with standard cells for security level of k = 500. The first technology mapping minimizes the area cost (1,023 μm^2), but over 2000 gates have security less than k. As the library is pruned across iterations, the logic covered by those gates gets re-implemented at higher area cost using more common cells, until the design becomes k-secure after 10 iterations at an area cost of 1,207 μm^2.

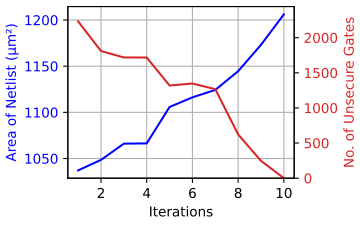


Fig. 3: Impact of pruning in Alg 1

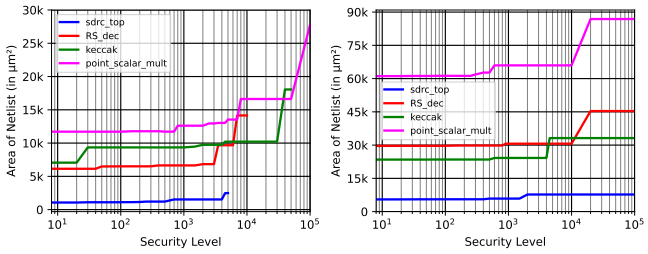
TABLE I: Sizes of pruned libraries from Alg. 1

Design	No. of Primitives		
	k = 10	k = 100	k = 1000
Point Scal.	18	17	9
Keccak	11	6	6
RS Dec.	13	8	6
SDRC	15	10	4

Algorithm 1: Technology map AIG to primitive cells such that each primitive is either instantiated 0 times, or at least k times, to satisfy the k-security requirement

```
   Input: G                          // And-Inverter-Graph of function
   Input: k                          // Required level of security
   Input: P                          // Library of primitive cells
   Output: G_tech                    // Technology mapped design
1  P' = P                            // Working copy of cell library
2  while iterations++ do
3      G_tech = tech_map(G, P')      // ABC tech. map
4      instCnt = {}                  // instance count of each primitive
5      foreach primitive p ∈ P' do
6          n = numOccurrences(G_tech, p)
7          if n ≠ 0 then             // ignore unused primitives
8              instCnt[p] = n
       /* evaluate whether mapped design is k-secure   */
9      p_min = argmin(instCnt)       // rarest prim. in G_tech
10     if instCnt[p_min] < k then    // p_min not secure
11         P' = P' \ p_min           // remove from lib for next try
12     else                          // success: all primitives are k-secure
13         return G_tech             // netlist mapped to library cells
```

As explained in the prior section, higher values of k cause more pruning, and thus tend to incur higher area cost. On all four designs, the trend of increasing area cost for higher security is observed when the target library contains standard cells (Fig. 4a), or contains LUT/LPGs (Fig. 4b). Comparing across the two library types, the costs are higher for LUT/LPGs due to the additional hardware required for programmability.



(a) Std cell / LPG / LUT Mapping (b) LPG / LUT mapped

Fig. 4: Comparison of logic area vs security level (k) for the four benchmark circuits.

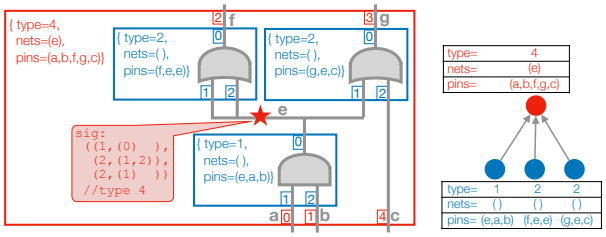
Our synthesis approach saves area relative to LUT-mapped versions of the same designs, which is an existing solution for design protection. Table II shows, for the point scalar multiplier, savings of 21.9% to 86.1% compared to LUT-based secure circuits, depending on the library and value of k.

TABLE II: Comparison against LUT-based design for point scalar multiplier benchmark. Reported area is units of μm^2

Benchmark	Std. Cell Lib		LPG/LUT Lib		LUT
	k=10	k=1000	k=10	k=1000	$k \approx \infty$
Point Scalar Mult	11,713	12,611	61,106	65,958	84,486

V. Isomorphic Clustering

Technology mapped netlist G_{tech} contains various types of primitives and their connections (see Fig. 1). The security guarantee from Alg. 1 assumes that all wires are unknown to the foundry, so each primitive instance is its own cluster with single output, and there are at least k instances of each cluster type. Local wires within each cluster are public and known to the foundry. Because connections between clusters are private, the overall design is k-secure if the clusters are k-secure.



(a) Circuit view of clustering step (b) Type relations

Fig. 5: Illustration of signature-based clustering

A. Secure-by-Construction Approach

Our algorithm to iteratively merge clusters into larger clusters is given in Alg. 2. The algorithm begins with the k-secure design from technology mapping, and preserves k-security through each clustering iteration, making the resulting circuit k-secure by construction. Taking this approach avoids the expensive step of finding arbitrary isomorphic subgraphs in a design [4].

We explain Alg. 2 using the annotated example circuit in Fig. 5. Initially, a cluster is created for each primitive instance (lines 3-5); the primitive clusters in Fig. 5a are shown in blue.

Each iteration of the main while loop (line 6) finds the most prominent cluster types that could be created through a simple merge at each node (lines 8-10) and chooses the best feasible cluster type as the one to create (line 11).

Then, nodes with that type each become an internal node of a new cluster (lines 12-13), and all of the clusters attached to that node get absorbed into the new cluster. In Fig. 5a, the clustering pass performs a merge at node e which has signature type 4. Three blue clusters get absorbed into the red cluster. This cluster would only be created if there are at least $k-1$ other occurrences of the cluster created at the same time using the same signature. Node e becomes an internal node of the new cluster (line 18), which then derives its internal nets and pin connections by inheriting the signals from the clusters it absorbs (lines 21 and 23).

When copying pins and internal nets from the absorbed clusters, all processing happens in the order of the signature, which is canonical (line 34), meaning that any two nodes that connect to the same pins of the same cluster types will always have the same signature and be created in exactly the same manner to ensure that they are isomorphic. Merging clusters at node e changes e from private to public, and the number of private wires is reduced.

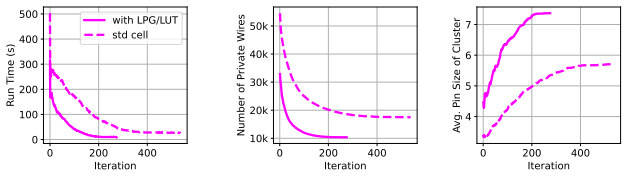
The process continues to iterate through typically 10s to 100s of iterations until no new clusters can be created without compromising k security. The algorithm then returns the set of private wires, which are the inter-cluster wires at the termination

Algorithm 2: Cluster components to reduce the number of private wires while preserving security level k.

```
   Input: G_tech                          // technology mapped design
   Input: k                               // Required level of security
   Output: W_priv                         // private edges
1  Insts = ()                             // cluster instances
2  Types = ()                             // cluster types
   // The initial clusters are the primitives
3  foreach primitive p ∈ G_tech do
4      c1 = new Cluster(type=p, nets=(), pins=(p.pins))
5      Insts.append(c1)
   /* Merge clusters while preserving security          */
6  while iterations++ do
7      sCount = {}
8      foreach net n ∈ W_priv do
9          sig = findType(n)
10         sCount[sig]++
11     if ∃N ⊂ W_Priv, where N is a set of at least k nets
         with same signature, that can be made public
         without violating k security of any types then
           // Create new clusters from existing ones
12         foreach n ∈ N do
13             MergeClusters(n)
14     else // No more clusters can be created
15         return W_priv  // inter-cluster wires are private

16 MergeClusters(net n) begin
17     idx = findType(n)
18     c1 = new Cluster(type=idx, nets=(n), pins=())
       // absorb clusters in canonical signature order
19     foreach {c2|c2 ∈ Insts & n ∈ c2.pins} do
           // pins/nets of c2 become pins/nets of c1
20         foreach {n2|n2 ∈ c2.pins & n2 ≠ n} do
21             c1.pins.append(n2)          // absorb pins
22         foreach n2 ∈ c2.nets do
23             c1.nets.append(n2)  // absorb internal nets
24         Insts.delete(c2)            // remove old cluster
25     Insts.append(c1)                // add new cluster
26     return

27 findType(net n) begin
       /* Generate canonical cluster signature for node n.
          Add sig to Types if new, and return its index  */
28     sig = ()     // sig of neighbors, in canonical order
29     foreach c2 ∈ Insts such that n ∈ c2.pins do
30         pins = ()    // pins of c2 that are attached to n
31         for i = 0 to size(c2.pins) do
32             if c2.pins[i] == n then // pin i attached to n
33                 pins.append(i)
34         sig.add((c2.type, (pins)))  // in canonical order
35     if Types.findIndexOf(sig) == ∅ then
36         Types.append(sig)      // sig previously unseen
37     return Types.findIndexOf(sig)
```



(a) runtime (b) private wires (c) pins

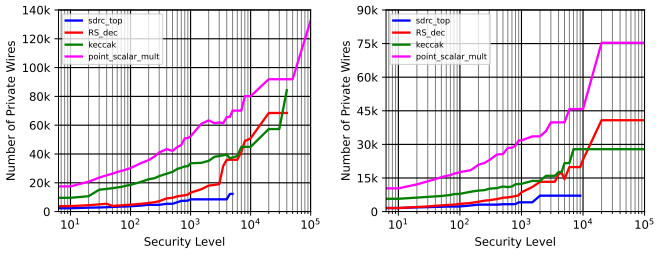
Fig. 6: Run time, number of private wires and average pin size of clusters across iterations of Alg. 2

of the clustering (line 15). At this point the clustering is complete, and the public (within-cluster) wires can be revealed to the foundry while the private wires are withheld. Fig. 6 shows, for each iteration of Alg. 2, the increase in cluster size (in terms of number of pins) and a decrease in the number of private connections.

The algorithm is efficient because isomorphism of clusters is preserved at each step through type signatures, and it avoids having to find and check arbitrary isomorphisms, which is infeasible for large designs. Fig. 6a shows improving run time of Alg. 2 as number of iterations increase. The number of nets considered at each clustering iteration keeps reducing. Additionally, nets that cannot be merged at one iteration can be removed from further consideration as viability for merging is non-increasing across iterations.

B. Security vs. Cost

Next, we look at relationship between number of private wires and the required security level for the benchmarks, as depicted in Fig. 7.



(a) Std cell / LPG / LUT mapping (b) LPG / LUT mapping

Fig. 7: Regardless of library type used, the number of wires that must remain private increases with security level k.

As k increases relative to the size of the circuit, it becomes harder to create clusters while preserving security. Therefore, as k increases there is an increase in the required number of private wires to meet security constraints. Our clustering approach allows us to achieve k security while making public up to 74% of wires with standard cell mapped designs, and up to 91% of wires with LPG/LUT mapped designs. Because our approach is general, we show its results for a comprehensive gate library containing standard cells (Subfig. 7a) and LPGs and LUTs (Subfig. 7b). In each case, Alg. 1 precedes Alg. 2.

C. Comparison against Related Work

We compare our clustering approach to the seminal 2013 work by Imeson et al. [4] which clusters by solving subgraph isomorphism. Their approach can produce optimal results on small designs, but scales poorly.

TABLE III: Comparing the percentage of private wires to Imeson et al. [4] for benchmark c432. The larger benchmarks, at k=48, are only solved by our clustering approach.

Design	Percentage of Private Wires				
	Security Level (k)				
	10	**20**	**30**	**40**	**48**
c432, Imeson et al. [4]	63.7	69.3	76.9	80.2	90.1
c432, Alg. 2	74.8	89.0	100.0	100.0	100.0
Point Scalar, Alg. 2	18.3	21.7	24.1	25.9	26.8
Keccak, Alg. 2	21.2	23.0	24.3	25.3	26.0
RS Decoder, Alg. 2	7.1	8.0	8.6	9.1	9.7
SDRC, Alg. 2	27.5	32.0	35.7	38.4	38.1

First we compare the proportion of edges that must be protected in the ISCAS '85 benchmark circuit c432 at various security levels (k=10,20,30,40,48) used in their paper. For a fair comparison, we use their same three gate library comprising inverter, NAND and NOR gates. The results are compared in the first two rows of Table III. For a circuit such as c432 with only a few hundred edges, which can be handled by either approach, Imeson's work outperforms ours. However, as shown in the next four rows of the table, our approach is demonstrated on much larger circuits that go far beyond what Imeson's formulation can solve; these cases yield excellent results with up 92.9% of edges being public. Note that the low number of private edges in these circuits is a property of the circuit itself, but only our method is able to handle circuits large enough to find the result.

VI. Design Space Tradeoff

For a given security level, our flow can also be used to tradeoff between the competing objectives of minimizing area and minimizing the number of wires that must be kept private from the foundry. The tradeoff exists when the target library comprises only standard cells, and also when it contains programmable cells. The design points in the space of area vs private wires that are not dominated by any others form a set of Pareto-optimal solutions for a designer to select from. The mechanism to reduce the number of private wires but increase area at security level k is to use larger values of k' ($k' > k$) in Alg. 1. The larger value k' more strictly constrains

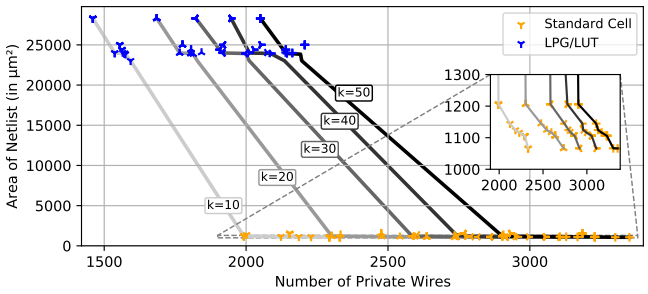


Fig. 8: Tradeoff between number of private wires and netlist area for SDRAM benchmark, for a given k.

the cell types in the mapped design, which increases the area; the cell types appearing in the mapped design are then each

more plentiful, which creates more opportunities for clustering in Alg. 2, leading to the reduction in private wires while still meeting the k security requirement. The tradeoffs for the SDRAM benchmark are shown in Fig. 8.

Fig. 8 shows that the tradeoff between area and private wires exists regardless of whether standard cells or programmable gates are included in the target library. Furthermore, the choice between standard cells and programmable gates is itself a tradeoff between area and number of private wires. Regardless of target library, reducing the number of private wires for phase 1 will tend to reduce the total wirelength and average delays when the circuit is later completed at phase 2, because wires added at phase 2 cannot exploit placement locality, and therefore will be the dominant contributor to interconnect delay.

VII. Conclusion

In this paper we have demonstrated a flow for secure fabrication in untrusted foundries, such that the foundry cannot make targeted manipulations. The approach combines standard cells, LPGs and LUTs, to ensure enough the design has enough ambiguity to meet security requirements while deploying its programmability selectively so as to limit area overheads. The approach provides an assurance of k security, and can trade off area against the number of private connections that must be withheld from the foundry. Our clustering algorithm is scalable to large circuits with over 10k gates, and found solutions that reduce private wires by up to 74.81% for the standard cell approach, and up to 91.33% in the LPG/LUT approach in our k secure netlist, depending on the design and the value of k.

Acknowledgment: This work has been supported in part by National Science Foundation (NSF) grant CNS-1749845.

References

[1] S. Adee, "The hunt for the kill switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[2] G. Miller, "The intelligence coup of the century." *Wash. Post*, 2020.

[3] S. Mitra, H.-S. P. Wong, and S. Wong, "The trojan-proof chip," *IEEE Spectrum*, vol. 52, no. 2, pp. 46–51, 2015.

[4] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation," in *22nd USENIX Security Symposium (USENIX Security 13)*, Aug. 2013, pp. 495–510. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/imeson

[5] T. Nigussie, J. C. Schabel, S. Lipa, L. McIlrath, R. Patti, and P. Franzon, "Design obfuscation through 3-D split fabrication with smart partitioning," *IEEE Trans. on Very Large Scale Integration (VLSI)*, pp. 1–14, 2022.

[6] M. Potkonjak, "Synthesis of trustable ICs using untrusted CAD tools," in *Design Automation Conference*, 2010, pp. 633–634.

[7] T. Reece and W. H. Robinson, "Detection of hardware trojans in third-party intellectual property using untrusted modules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 357–366, 2016.

[8] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," vol. 22, no. 1, May 2016. [Online]. Available: https://doi.org/10.1145/2906147

[9] M. N. I. Khan, A. De, and S. Ghosh, "Cache-out: Leaking cache memory using hardware trojan," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1461–1470, 2020.

[10] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware trojans through side-channel engineering," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, 2009, pp. 382–395.

[11] J. Breier and W. He, "Multiple fault attack on PRESENT with a hardware trojan implementation in FPGA," in *2015 International Workshop on Secure Internet of Things (SIoT)*, 2015, pp. 58–64.

[12] T. Reece and W. H. Robinson, "Analysis of data-leak hardware trojans in AES cryptographic circuits," in *IEEE International Conference on Technologies for Homeland Security (HST)*, 2013, pp. 467–472.

[13] Y. Wang, P. Chen, J. Hu, and J. J. V. Rajendran, "Routing perturbation for enhanced security in split manufacturing," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 605–510.

[14] A. Sengupta, S. Patnaik, J. Knechtel, M. Ashraf, S. Garg, and O. Sinanoglu, "Rethinking split manufacturing: An information-theoretic approach with secure layout techniques," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 329–326.

[15] Y. Cheng, Y. Wang, H. Li, and X. Li, "A similarity based circuit partitioning and trimming method to defend against hardware trojans," in *IEEE Annual Symposium on VLSI*, 2015, pp. 368–373.

[16] M. Li, B. Yu, Y. Lin, X. Xu, W. Li, and D. Z. Pan, "A practical split manufacturing framework for trojan prevention via simultaneous wire lifting and cell insertion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1585–1598, 2019.

[17] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *Design, Automation Test in Europe*, 2013, pp. 1259–1264.

[18] Z. Chen, P. Zhou, T.-Y. Ho, and Y. Jin, "How secure is split manufacturing in preventing hardware trojan?" in *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, 2016, pp. 1–6.

[19] IEEE Electronics Packaging Society, "Chapter 19: Security," *Heterogeneous Integration Roadmap*, 2019.

[20] S. Patnaik, M. Ashraf, O. Sinanoglu, and J. Knechtel, "A modern approach to ip protection and trojan prevention: Split manufacturing for 3d ics and obfuscation of vertical interconnects," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1815–1834, 2021.

[21] P. Mohan, O. Atli, J. Sweeney, O. Kibar, L. Pileggi, and K. Mai, "Hardware redaction via designer-directed fine-grained eFPGA insertion," in *Design, Automation Test in Europe*, 2021, pp. 1186–1191.

[22] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Des & Test of Comp.*, vol. 27, no. 1, pp. 66–75, 2010.

[23] J. Bhandari, A. K. Thalakkattu Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri, "Exploring eFPGA-based redaction for IP protection," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[24] Z. U. Abideen, T. D. Perez, and S. Pagliarini, "From FPGAs to obfuscated eASICs: Design and security trade-offs," *arXiv preprint arXiv:2110.05335*, 2021.

[25] J. Chen and B. C. Schafer, "Area efficient functional locking through coarse grained runtime reconfigurable architectures," in *Proc. of the 26th Asia and South Pacific Design Automation Conf.*, ser. ASPDAC '21, 2021, pp. 542–547.

[26] M. M. Shihab, J. Tian, G. R. Reddy, B. Hu, W. Swartz, B. Carrion Schaefer, C. Sechen, and Y. Makris, "Design obfuscation through selective post-fabrication transistor-level programming," in *Design, Automation Test in Europe Conf Exhibition (DATE)*, 2019, pp. 528–533.

[27] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris, "Silicon demonstration of hardware trojan design and detection in wireless cryptographic ICs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1506–1519, 2017.

[28] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware," in *Proc. of 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[29] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, 2013, pp. 197–214.

[30] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 532–535.

[31] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, pp. 1–23, 2016.

[32] S. Inc., "Open-Cell 15 nm FreePDK," https://si2.org/open-cell-library/.

[33] H. Hsing, "Elliptic Curve Group," https://opencores.org/projects/ecg, (module: point_scalar_mult).

[34] V. semiconductors, "Reed Solomon Decoder (204,188)," https://opencores.org/projects/reed_solomon_decoder.

[35] H. Hsing, "SHA3 (KECCAK)," https://opencores.org/projects/sha3.

[36] D. Annayya, "8/16/32 bit SDRAM Controller," https://opencores.org/projects/sdr_ctrl.

[37] R. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Comp. Aided Verification*, 2010, pp. 24–40.

[38] "Supply Chain Security," https://github.com/danholcomb/supply-chain-security.