3D Hardware Canaries

Sébastien Briais[4], **Stéphane Caron**[1], **Jean-Michel Cioranesco**[2,3], **Jean-Luc Danger**[5], **Sylvain Guilley**[5], **Jacques-Henri Jourdan**[1], **Arthur Milchior**[1], **David Naccache**[1,3], **Thibault Porteboeuf**[4]

[1] École normale supérieure, Département d'informatique
`surname.name@ens.fr`
[2] Altis Semiconductor
`jean-michel.cioranesco@altissemiconductor.com`
[3] Sorbonne Universités – Université Paris II
`jean-michel.cioranesco@etudiants.u-paris2.fr`
[4] Secure-IC
`surname.name@secure-ic.com`
[5] Département Communications et Electronique
Télécom-ParisTech
`surname.name@telecom-paristech.fr`

Abstract. 3D integration is a promising advanced manufacturing process offering a variety of new hardware security protection opportunities. This paper presents a way of securing 3D ICs using Hamiltonian paths as hardware integrity verification sensors. As 3D integration consists in the stacking of many metal layers, one can consider surrounding a security-sensitive circuit part by a wire cage.

After exploring and comparing different cage construction strategies (and reporting preliminary implementation results on silicon), we introduce a "hardware canary". The canary is a spatially distributed chain of functions F_i positioned at the vertices of a 3D cage surrounding a protected circuit. A correct answer $(F_n \circ \ldots \circ F_1)(m)$ to a challenge m attests the canary's integrity.

1 Introduction

3D integration is a promising advanced manufacturing process offering a variety of new hardware security protection opportunities. This paper presents a way of securing 3D ICs using Hamiltonian paths[1] as integrity verification sensors. 3D integration consists in the stacking of many metal layers. Hence, one can consider surrounding a security-sensitive circuit part by a wire cage, for instance a Hamiltonian path connecting the vertices of a cube (Fig. 1). In this paper, different algorithms to construct cubical Hamiltonian structures are studied; those ideas can be extended to other forms of sufficiently dense lattices.
Since 3D integration is based on the vertical stacking of different dies, a Hamiltonian cage can surround the whole target and protect its content from physical attacks. 3D ICs are relatively hard to probe due to the tight bonding between layers [11]. Moreover, the 3D path can even penetrate the protected circuit and connect points in space between the protected circuit's transistors.

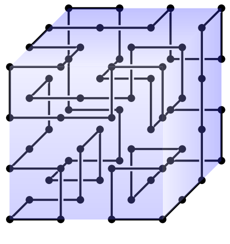


Fig. 1: Hamiltonian cycle passing through the vertices of a $4 \times 4 \times 4$ cube

[1] A Hamiltonian circuit (hereafter "cage" or simply "path" for the sake of conciseness) is an undirected path passing once through all the vertices of a graph.

A path running through different metal layers and different dies can thus serve as a *digital* integrity verification sensor allowing the sending and the collecting of signals. In addition, the wire can be used to fill gaps in empty circuit parts to increase design compactness and make reverse-engineering harder.

Such a protection proves challenging in terms of design as it requires devising new manufacturing and synthesis tools to fit the technology used [1,2]. However the resulting structures prove very helpful in protecting against active probing (*cf.* Appendix A).

Throughout this paper n will represent the number of points forming the edge of a cubical Hamiltonian structure. We will focus our study on cubical structures, but the algorithms and concepts that are presented hereafter can in principle be extended to many types of sufficiently dense lattices of points.

2 Generating Random 3D Hamiltonian Paths

2.1 General Considerations

The problem of finding a Hamiltonian path in arbitrary graphs (HAMPATH) is NP-complete. Membership in NP is easy to see (given a candidate solution, the solution's correctness can be verified in quasi-linear time). We refer the reader to [3] for more information on HAMPATH.

A quick glance reveals that a cube's n^3 vertices, potentially connectable by a mesh of $3n^2(n-1)$ edges, break-down into four categories, illustrated in Fig. 2[2]:

- $(n-2)^3$ vertices corresponding to the cube's innermost edges (*i.e.* not facing the outside) can be potentially connected in any of the possible 3D directions (right, left, up, down, front, rear).
- $6(n-2)^2$ vertices, facing the cube's outside in exactly one direction, can be potentially connected in five possible directions.
- $12(n-2)$ vertices, facing the cube's outside in exactly two directions, can be potentially connected in four possible directions.
- 8 extreme corner vertices can be connected in only three possible manners.

Indeed: $(n-2)^3 + 6(n-2)^2 + 12(n-2) + 8 = ((n-2)+2)^3 = n^3$

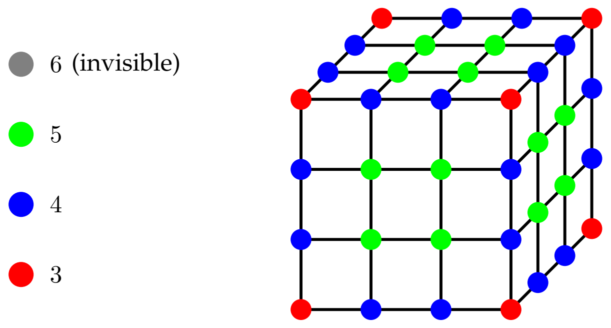


Fig. 2: Potential edge connectivity

[2] The depicted cube is shown as a solid opaque object for the sake of clarity.

We observe that for HAMPATH to be solvable in a cube, n must be even. If we depart from point the $(0, 0, 0)$ and reach a point of coordinates (x, y, z) after visiting i vertices, then $x + y + z$ and i have the same parity. Given that the path must collect all the cube's vertices, the cube size must necessarily be even.

2.2 Odd Size Cubes

The above observation excludes the existence of odd-size cubes unless one skips in such cubes an edge (x, y, z) such that $x + y + z \equiv 1 \bmod 2$. To extend the construction to odd $n = 2k + 1$ while preserving symmetry, we *arbitrarily* decide to exclude the central vertex (*i.e.* at coordinate (k, k, k)) when n is odd.

Assume that we color vertices in black and white alternatingly (the cube's 8 extreme vertices being black) with black corresponding to even-parity $x + y + z$ and white corresponding to odd parity $x+y+z$. Here $0 \leq x, y, z \leq 2k$. In other words, a $(2k+1)$-cube has $4k^3 + 6k^2 + 3k$ white vertices and $4k^3 + 6k^2 + 3k + 1$ black vertices.

The coordinate of the cube's central vertex is (k, k, k) which parity is identical to the parity of k. When k is even, vertex (k, k, k) is black and when k is odd vertex (k, k, k) is white. If we remove vertex (k, k, k) it appears that:

– When k is even, (*i.e.* $n = 2k + 1 = 4\ell + 1$) we have as many black and white vertices (namely $4k^3 + 6k^2 + 3k$).

– When k is odd, we have $4k^3 + 6k^2 + 3k + 1$ black vertices and $4k^3 + 6k^2 + 3k - 1$ white vertices.

Noting that each edge causes a color switch, we see that Hamiltonian paths in cubes of size $4\ell+3$ cannot exist. Note that if one extra black vertex is removed[3] then (the now asymmetric) construction becomes possible for all k.

It remains to prove that cubes of size $n = 4\ell + 1$ exist for all $\ell \neq 0$. This is seen to be true given the extensible structure shown in Appendix B. If ℓ is increased, the structure can be re-scaled by enlarging each floor by four units and piling up four additional floors (two at the top and two at the bottom).

As a purely theoretical side-note, although we have not fully analyzed the constructibility problem in higher dimensions, it seems that 4D cubes of all sizes are "constructible". A hypercube of dimension d has n^d vertices with a central vertex at coordinate $(dk, \ldots, dk)$. Hence when d is even the parity issue *seems* to vanish.

[3] *e.g.* one of the cube's extreme edges which is necessarily black.

3 A Toolbox for Generating 3D Hamiltonian Cycles

3.1 From Two to Three Dimensions

We start by presenting a first algorithm for constructing random[4] Hamiltonian cycles in graphs having a minimum degree equal to at least half the number of their vertices.

Our application requires an efficient algorithm that outputs cycles passing through a very large number of vertices. The first algorithm reduces the problem's complexity by using smaller cycles that we will progressively merge to form the final bigger cycle. Consider the elementary Hamiltonian cycle forming a simple 2×2 square. To combine two such squares all we need are two parallel edges. Merging (denoted by the operator $\leftrightsquigarrow$) can be done in two ways as shown Fig. 3. Note that this association not only preserves Hamiltonicity but also extends it.

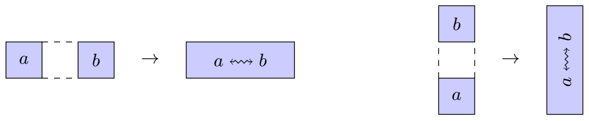


Fig. 3: Association of squares along the x axis (leftmost figure), or the y axis (rightmost figure)

In other words, at each step two different Hamiltonian cycles in adjacent graphs are merged, and a new Hamiltonian cycle is created. The process is repeated until only one Hamiltonian cycle remains. We implemented this process in C. As explained previously, our program cannot find Hamiltonian cycles for odd cardinality values simply because such cycles do not exist (see Algorithm 1). The code starts by filling the lattice with 2×2 squares, and then associates them randomly. The program ends when only one cycle is left (Fig. 4).

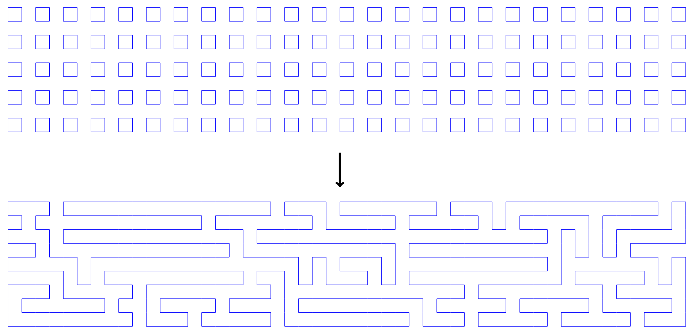


Fig. 4: Rewriting 125 squares filling a 50×10 lattice as a Hamiltonian cycle using Algorithm 1

Algorithm 1 Cycle Merging

```
1:  Input $p, q \in 2\mathbb{N}$.
2:  let $Q = Q_1, ..., Q_v$ be the $v = \frac{pq}{4}$ squares of size 2 filling the lattice of $p \times q$ points.
3:  while Card$(Q) \neq 1$ do
4:      choose randomly $\{a, b\} \in Q^2$ with $a \neq b$.
5:      if $a$ and $b$ have at least one couple of neighbouring parallel edges then
6:          Break a randomly chosen couple of parallel neighbouring edges, verify that they form a single
            Hamiltonian circuit and merge $c = a \leftrightsquigarrow b$.
7:          let $Q = Q \cup \{c\} - \{a, b\}$
8:      else
9:          goto line 4
10:     end if
11: end while
```

[4] As explained in Appendix C , the entropy of our structure generators seems very complex to estimate.

The algorithm is pretty fast, and we were able to build Hamiltonian cycles of 10^5 points using a laptop[5] within few seconds. For some p and q values, we observed some runtime spikes in single measurements due to convergence issues. Fig. 5 shows the average runtime over 100 measurements as well as the standard deviation at each point in red.

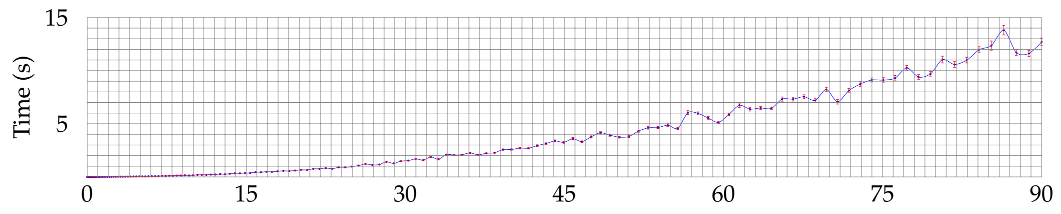


Fig. 5: Cycle Merging runtime as a function of the number of points $\times 10^3$ (average over 100 measurements)

To transform a rectangular 2D Hamiltonian cycle into a 3D one, we run Algorithm 1 for $\{p, q\} = \{p, p^2\}$ to get a $p \times p^2$ rectangle $\mathcal{L}$ similar in nature to the one shown in Fig. 4.

Then, letting (x_i, y_i) denote the Cartesian coordinates of points in $\mathcal{L}$, with the first point being $(0, 0)$, we fold $\mathcal{L}$ into a 3D structure of coordinates (x'_i, y'_i, z'_i) using the following transform where $j = \lfloor \frac{x_i}{p} \rfloor$ and $\ell \equiv j \bmod 2$:

$$\varphi = \begin{cases} x'_i = (-1)^\ell (x_i - jp) + \ell(p-1) \\ y'_i = y_i \\ z'_i = j \end{cases}$$

The result is shown in Fig. 26 (Appendix D).

It remains to destroy the folded nature of the construction while preserving Hamiltonicity. This is done as follow: Identify anywhere in the generated structure the red pattern shown at the leftmost part of Fig. 6 where at positions a, b, c, d edges take any of the blue positions. Iteratively apply this rewriting rule *along any desired axis* until the resulting structure gets "mixed enough" to the designer's taste. Evidently, this is only one possible rewriting rule amongst several.

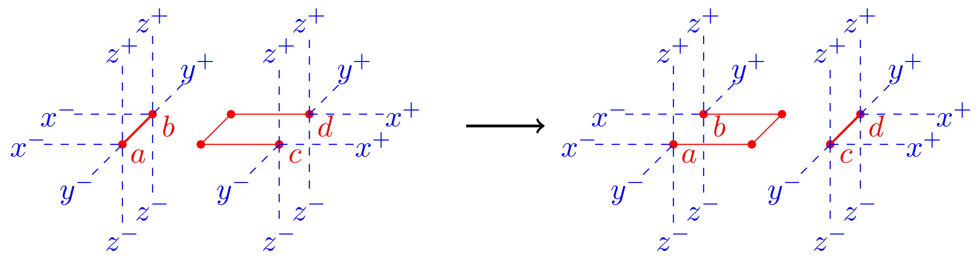


Fig. 6: Rewriting rule

Note that the zig-zag folding φ is only one among many possible folding options as φ may be replaced by any 2D (preferably random) plane-filling curve of size $p \times p$ (*e.g.* a Peano curve [8]).

[5] MacBook Air 1.8 GHz Intel Core i7.

3.2 Random Cube Association

Another approach consists in generalizing Algorithm 1 to the associating of elementary 3D cubes. As shown in Fig. 28, one can fill the target lattice by a random sampling of six elementary Hamiltonian cubes (Fig. 27), associate them randomly and further randomize the resulting structure by rewriting.

The algorithm proves very efficient (Fig. 7) and takes a few seconds[6] to compute a random Hamiltonian cube of size 50 (125 000 points).

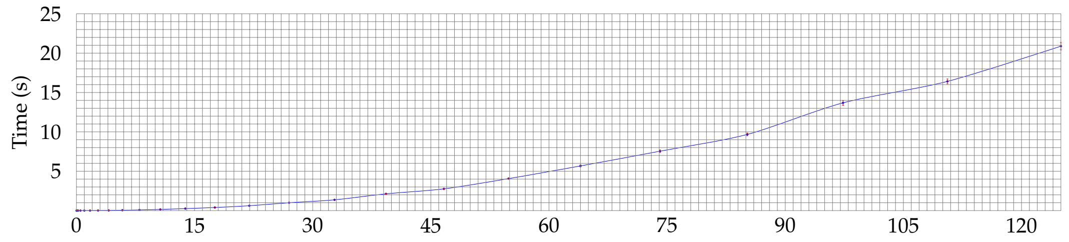


Fig. 7: Random Cube Association runtime as a function of the number of points $\times 10^3$ (average over 100 measurements)

The algorithm picks random parallel edges from different Hamiltonian cycles and attempts to associate them in one new structure. By opposition of the 2D case, the 3D case presents a new difficulty which is that in some cases associable parallel edges suddenly cease to exist. To force termination we abort and restart from scratch if the number of iterations executed without finding a new association exceeds the upper bound p^3. To compute structures over huge lattices (*e.g.* $n = 100$), one might need to introduce additional association rules (*e.g.* the rule shown in Fig. 8) to avoid such deadlocks.

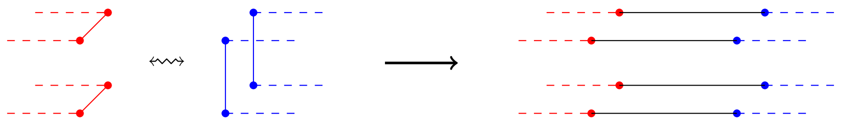


Fig. 8: An additional association rule (example)

3.3 Cycle Stretching

Our third algorithm maintains and extends a set of edges E initialized with the four edges defined by the square of vertices $(0,0,0)$, $(0,1,0)$, $(1,1,0)$ and $(1,0,0)$. At each iteration, the algorithm selects a random edge $e \in E$ and one of the four extension directions shown in Fig. 9. If such an extension is possible (in other words, by doing so we do not bump into an edge already in E) then E is extended by replacing e by three new edges (one parallel to e and two orthogonal to e in the chosen extension direction). If e cannot be replaced, *i.e.* none of the four extensions is possible, we pick a new $e' \in E$ and try again.

[6] MacBook Air 1.8 GHz Intel Core i7.

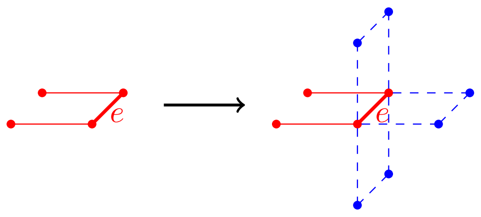


Fig. 9: Extension options

The algorithm keeps track of a subset of E, denoted B, interpreted as the set of potentially stretchable edges of E. B avoids trying to stretch the same e over and over again.

At each stretching attempt the algorithm picks a random $e \in B$. As the algorithm tries to stretch e, e is removed from B (no matter if the stretching attempt is successful or not). If stretching succeeded, e is also removed from E and three new edges replacing e are added to B and E.

The algorithm halts when $B = \varnothing$. If upon halting $|E| = n^3 - (n \bmod 2)$ then the algorithm succeeds, otherwise the algorithm fails and has to be re-launched. Since at most $3n^2(n-1)$ vertices can be added to B, the algorithm will eventually halt.

A non-optimized implementation running on a typical PC found a solution for $n = 6$ in about a minute and a solution for $n = 8$ in 30 hours. The same code was unable to find a solution for $n = 10$ in three weeks. An empirical human inspection of the obtained cubes shows that the resulting structures seem very irregular. Hence, an interesting strategy consists in generating a core cube of size $n = 8$ by cycle stretching, surrounding it by elementary size 2 cubes and proceeding by random cube association and rewriting.

Algorithm 2 Edge Stretching

```
1:  let $E$ = the four vertices defined by the square $(0,0,0)$, $(0,1,0)$, $(1,1,0)$, $(1,0,0)$.
2:  let $B = E$.
3:  while $B \neq \varnothing$ do
4:      let $e \in_R B$, we denote the vertices of $e$ by $e = [e_1, e_2]$.
5:      let $B = B - \{e\}$
6:      let dir $= \{\leftarrow, \rightarrow, \uparrow, \downarrow, \nearrow, \swarrow\}$
7:      while dir $\neq \varnothing$ do
8:          let $d \in_R$ dir
9:          let dir = dir $- \{d\}$
10:         if $d$ and $e$ are not aligned and stretching is possible then
11:             $E = E - \{e\}$.
12:             $E = E \cup \{[e_1, v_1], [v_1, v_2], [v_2, e_2]\}$.
13:             break
14:         end if
15:     end while
16: end while
```

In the above algorithm the sentence "*stretching is possible*" is formally defined as the fact that no edges in E pass through the two vertices v_1,v_2 such that the segment $[v_1, v_2]$ is parallel to e in direction d. Arrows represent right, left, up, down, front and backwards directions, *i.e.* $\begin{array}{ccc} & \uparrow & \nearrow \\ \leftarrow & & \rightarrow \\ \swarrow & \downarrow & \end{array}$

3.4 Constraining Existing Hamiltonian Pathfinding Algorithms

A fourth experimented approach consisted in adapting existing HAMPATH solving strategies. (Dharwadker) [4] presents a polynomial time algorithm for finding Hamiltonian paths in certain classes of graphs. Assuming that the graphs that we are interested in are in such a class, we tweaked [4]'s C++ code to find Hamiltonian cycles in cubes. The resulting code succeeded in finding solutions, but these had a too regular appearance and had to be post-processed by re-writing.

We hence constrained the algorithm by working in a randomly chosen subgraph E of the full n^3 cube. We define a *density factor* $\gamma \leq 1$ allowing to control the number of edges in E to which we apply [4]. The ratio of edges in E and n^3 is expected to be approximately γ. Note that because of the heuristic corrective step (9), meant to reduce the odds that certain points remain unreachable, E's density is expected to be slightly higher than γ. The corresponding algorithm is:

Algorithm 3 Edges Selection Routine

```
1: $E = \varnothing$
2: **for** each vertex $v = (x, y, z)$ of the full cube **do**
3:     **for** each move $\mathrm{d}v = (\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)$ in $\{(1,0,0), (0,1,0), (0,0,1)\}$ **do**
4:         **generate** a random $r \in [0, 1]$
5:         **if** $r < \gamma$ and $(0,0,0) \leq v + \mathrm{d}v \leq (n-1, n-1, n-1)$ **then**
6:             **add** edge $[v, v + \mathrm{d}v]$ to $E$
7:         **end if**
8:     **end for**
9:     **if** loop 3 didn't add to $E$ any edge having $v$ as en extremity **then**
10:        **goto** line 3
11:    **end if**
12: **end for**
```

Practical experiments show indeed that as γ diminishes, the generated Hamiltonian cycles seem increasingly irregular (for high (*i.e.* $\simeq 1$) γ values the algorithm fills the cube by successive "slices"). Finding solutions becomes computationally harder as γ diminishes, but using a standard PC, it takes about a second to generate an instance for $\{\gamma = 0.8, n = 6\}$ and an hour to generate a $\{\gamma = 0.86, n = 10\}$ one. The reader is referred to Appendix F for several experimental results.

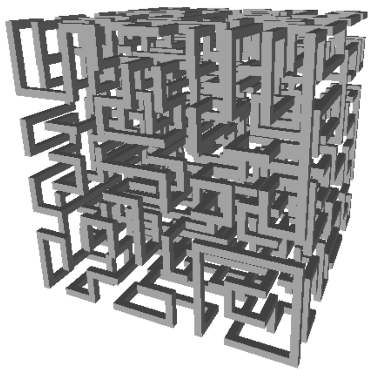


Fig. 10: A $n = 10$ Hamiltonian cycle obtained by a modified version of Dharwadker's algorithm [4]

3.5 Branch-and-Bound

Another experimented approach was the use of branch-and-bound: Using a recursive function, we can try all different cycles. Given a connected portion of a potential Hamiltonian path, this function tries to add all the possible new edges and calls itself recursively. If the function is called with a complete path, the job is done.

We added several heuristic improvements to this method:

1. If the set of vertices unlinked by the current path is disconnected, it is clear that we won't be able to find any Hamiltonian path, and thus we can stop searching.
2. If this set is not connected to the extremities of the current path, we can also halt.
3. The existence of an Hamiltonian path containing a given sub-path only depends on the extremities and on the set of vertices in the path. We can hence use a dynamic programming approach to avoid redundant computations.
4. We tried multiple heuristics to chose the order of recursive calls.

However, those approaches proved much slower than cycle stretching: it appears that the branch-and-bound algorithm makes decisive choices at the beginning of the path without being able to re-consider them quickly. We tried to count all the Hamiltonian cycles when $n = 4$ using this algorithm, but the code proved too slow to complete this task in a reasonable time.

Those results suggest a meta-heuristic approach that would be intermediate between branch-and-bound and stretching: we can make a cycle evolve using meta-heuristics until we obtain an Hamiltonian cycle. Using this method (that we did not implement) we should be able to re-consider any previous choice without restarting the search process.

3.6 Rewriting 3D Moore Curves

Finally, one can depart from a know regular 3D cycle (*e.g.* a 3D Moore curve as shown in Fig.11) and rewrite it. Moore curves are particularly adapted to such a strategy given that the maze entrance and exit are two adjacent edges. However, as shown in Fig.11c (a top-down view of Fig.11b), Moore curves are inherently regular and must be re-rewritten to gain randomness.

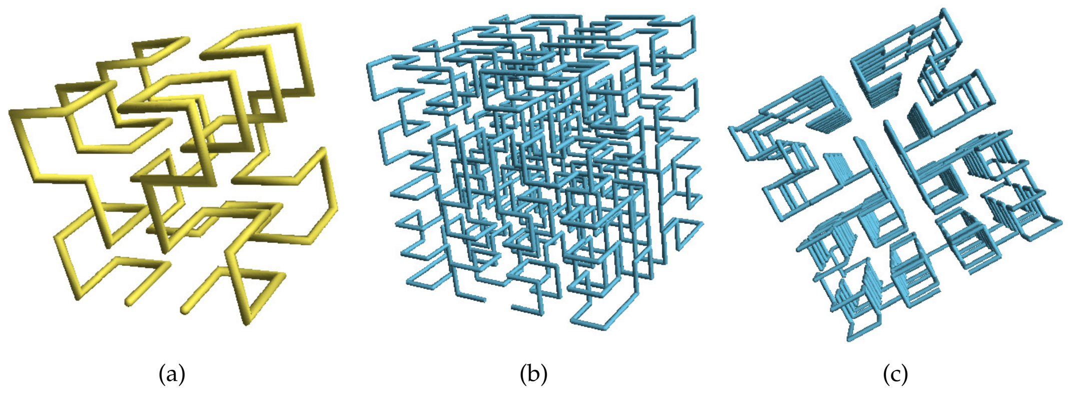

(a) (b) (c)

Fig. 11: Example of Moore Curves [5]

4 Silicon Experiments

To test manufacturability in silicon we created a first passive cage meant to protect an 8-bit register. We notice that the compactness of the cage provides a very good reverse-engineering protection.

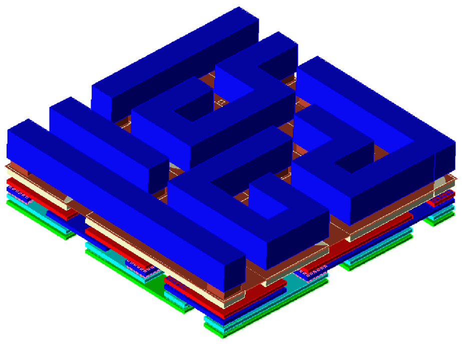
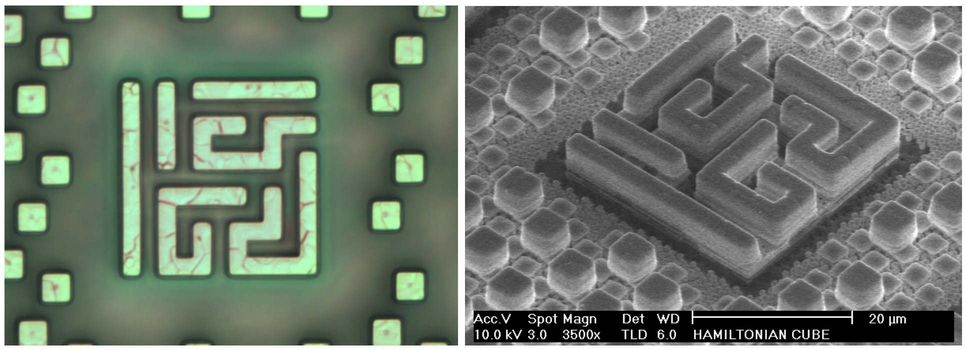


Fig. 12: 3D layout of a cage of size 6 (130nm, 6 Metal Layers Technology)

The implemented structure (Fig. 12) is a $6 \times 6 \times 6$ Hamiltonian cube stretching over six metal layers, the first four metal layers are copper ones, and the last two metal layers are thicker and made of aluminum (130nm RF technology, Fig. 13). The cube is 26μm wide and covers an 8 bit register.

As will be explained in the next section, this first prototype is not dynamic, the Hamiltonian path is not connected to transistors. The implementation of a simplified dynamic structure as described in section 5 is underway and does not seem to pose insurmountable technological challenges. Moreover, all layers of the prototype are processed in one side of the silicon, so this implementation does not prevent backside attack. Backside metal deposit and back to back wafer stacking must thus be investigated to thwart backside attacks as well.



(a) (b)

Fig. 13: Top layer view (a) and tilted SEM view (b) of a 26μm wide $6 \times 6 \times 6$ cage implemented in a 130nm technology ($\times$2500)[7]

[7] The structure implemented in silicon is surrounded by fill shapes used as a gaps filler, due to manufacturing constraints (polishing).

5 Dynamically Reconfigurable 3D Hamiltonian Paths

A canary is a binary constant placed between a buffer and stack data to detect buffer overflows. Upon buffer overflow, the canary gets corrupted and an overflow exception is thrown. The term "canary" is inherited from the historic practice of using canaries in coal mines as toxic gas biological alarms. The dynamic structures presented in this section are hardware equivalents of biologic canaries: our "hardware canary" is formed of a spatially distributed chain of functions F_i positioned at the vertices of a 3D cage surrounding a protected circuit. In essence, a correct answer $(F_n \circ \ldots \circ F_1)(m)$ to a challenge m will attest the canary's integrity. The device described in this section relies on a library of paths precomputed using the toolbox of algorithms described in the previous section.

5.1 Reconfigurable 3D Mazes

The construction of a 3D dynamic grid begins with the description of a Network On Silicon (NOS) with speed, power and cost constraints [7,12]. As described in [6,9], metal wires are shared, or made programmable, by introducing *switch-boxes*, that serve as the skeleton of the dynamic Hamiltonian path. Each switch-box is an independent cryptographic cell that corresponds to a vertex of the graph. The switch-boxes are reconfigurable and receive reconfiguration information as messages flow through the Hamiltonian path during each session c. All boxes are clocked[8], and able to perform basic cryptographic operations. Six cell-level parameters are used to define each switch-box:

– A *coordinate identifier* i is a positive integer representing the ordinal number of the box's Cartesians coordinates: *i.e.* $i = x + ny + n^2z$.

– A *session identifier* c is an integer representing the box's configuration: this value is incremented at each new reconfiguration session.

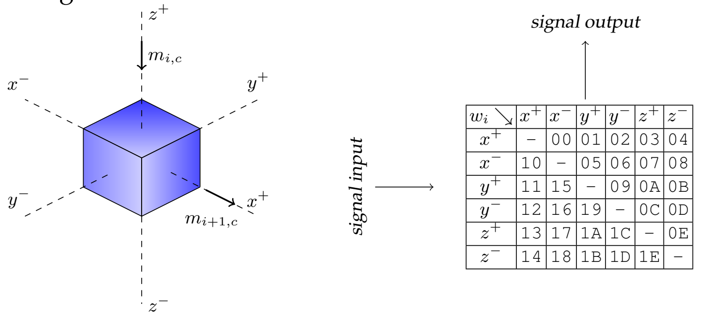
– A *key* k_i shared with the protected processor located inside the cage.

– A *routing configuration* $w_{i,c}$ chosen between the thirty possible routing positions of a 3D bi-directional switch (Fig. 14)[9].

– A *state variable* $s_{i,c}$ computed at each clock cycle from the incoming data $m_{i,c}$ (see hereafter) and the preceding state, $s_{i,c-1}$. The state $s_{i,c}$ is stored in the switch-box's internal memory[10].

$$\begin{cases} m_{i+1,c} = F(m_{i,c}, k_i, w_{i,c}, s_{i,c}) \\ s_{i,c+1} = G(m_{i,c}, k_i, w_{i,c}, s_{i,c}) \end{cases} \tag{1}$$

The output data $m_{i+1,c}$ is computed within box i using the input data $m_{i,c}$ and an integrated cryptographic function F, serving as a lightweight MAC. The final output $m_{n^3,c}$ attests the cage's integrity during session c.



$w_i \searrow$	x^+	x^-	y^+	y^-	z^+	z^-
x^+	–	00	01	02	03	04
x^-	10	–	05	06	07	08
y^+	11	15	–	09	0A	0B
y^-	12	16	19	–	0C	0D
z^+	13	17	1A	1C	–	0E
z^-	14	18	1B	1D	1E	–

Fig. 14: Example of a 3D switch-box programmed with a routing configuration $w_i = \texttt{0x13}$

[8] We denote by t the clock counter.

[9] For switch-boxes depicted in red, blue and green (Fig. 2) the number of possible configurations drops to (respectively) 6, 12 and 20.

[10] Upon reset $s_{i,0} = 0$ for all i.

Each switch-box comprises five logic parts (Fig. 15) that serve to route the integrity attestation signal through the box's six IOs and successively MAC the input values $m_{i,c}$:

– Two multiplexers routing IOs, with three state output buffers to avoid short-circuits during re-configuration.

– A controller commanding the two multiplexers' configuration.

– A MAC cell for processing data and a register for storing results.

– A register storing the state variable $s_{i,c}$, the key k_i, the present configuration $w_{i,c}$, the next box configuration $w_{i,c+1}$ and the clock counter t.

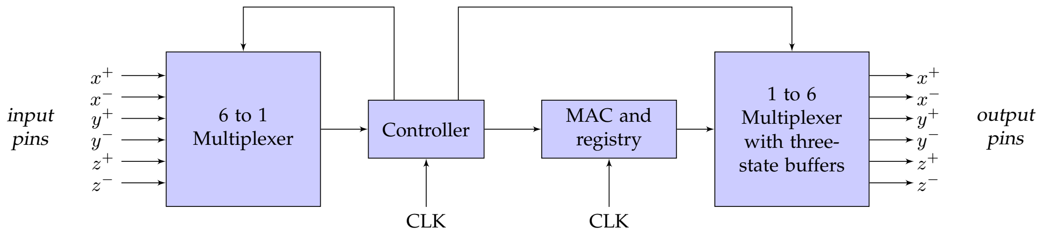


Fig. 15: Logic diagram of a 3D switch-box

The input message $m_{0,c}$, sent through the Hamiltonian path, is composed of two parts serving different goals (Fig. 16):

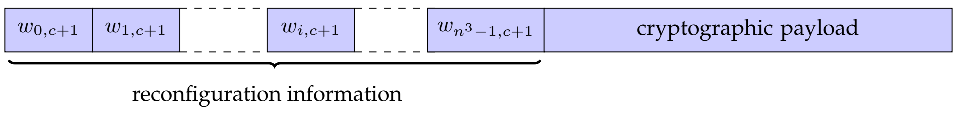


Fig. 16: Structure of message $m_{0,c}$

– The first message part is dedicated to reconfiguring the grid. For a cube of size n, the reconfiguration information has n^3 parts, each containing the next routing configuration $w_{i,c+1}$ of switch-box i. As the routing information of each switch-box can be coded on 5 bits, the reconfiguration information is initially $5n^3$ bits long[11]. Basically, this message part carries the position of all switches for the next Hamiltonian path of session $c+1$.

– The second message part (cryptographic payload) is used to attest the circuit's integrity, the 64-bits payload will be successively MACed by all switch-boxes and eventually compared to a digest computed by the protected circuit. If possible, one should select a function F that simplifies after being composed with itself to reduce the protected circuit's computational burden.

[11] Note that the reconfiguration information part of the $m_{i,c}$'s gets shorter and shorter as i increases, *i.e.* as the message approaches the last switch-box.

5.2 Description of the Dynamic Grid and the Integrity Verification Scheme

Upon reset, each switch-box is in a default configuration $w_{i,0}$ corresponding to an initial predefined hardwired Hamiltonian path for session $c = 0$. The input and the output boxes (S_0 and S_{n^3-1}) are only partially reconfigurable; namely, the routing of S_0's input and the routing of S_{n^3-1}'s output cannot be changed. To clarify the reconfiguration dynamics, we denote by t the number of clock ticks elapsed since system reset assuming a one bit per clock tick throughput; given that 5 bits are dropped at each "station", a full reconfiguration route (session) claims

$$5\sum_{j=0}^{n^3-1}(n^3-j) = \frac{5}{2}n^3(n^3+1)$$

clock ticks, which is the time needed for the reconfiguration information to flow through all n^3 switch-boxes *i.e.* the number of clock ticks elapsed between the entry of the first bit of $m_{0,c}$ into S_0 and the exit of the last bit of $m_{n^3,c}$ from S_{n^3-1}. Note that this figure does not account for the time necessary for payload transit[12].

At $t = 0$**:** A new session c starts and the first bit of $m_{0,c}$ is received by S_0 form the protected processor.

For $5\sum_{j=0}^{i-1}(n^3-j) = \frac{5}{2}i(2n^3+1-i) \le t \le 5\sum_{j=0}^{i}(n^3-j) - 1 = \frac{5}{2}(i+1)(2n^3-i) - 1$**:** All switch-boxes except S_{i-1} and S_i are inactive (dormant). S_{i-1} sends the message $m_{i-1,c}$ to S_i which performs the following operations:

- Store the reconfiguration information $w_{i,c+1}$, for the next Hamiltonian route of session $c+1$.
- Compute $m_{i+1,c}$ and update $s_{i,c+1}$ as defined in formula (1).

At $t = 5\sum_{j=0}^{n^3-1}(n^3-j) = \frac{5}{2}n^3(n^3+1)$**:** The first bit of message $m_{n^3,c}$ emerges from the grid (from S_{n^3-1}) and all switch-boxes re-configure themselves to the new Hamiltonian path $c+1$. $m_{n^3,c}$ is received by the protected processor who compares it to a value computed by its own means. At the next clock tick a new message $m_{0,c+1}$ is sent in, and the process starts all over again for a new route representing session number $c+1$.

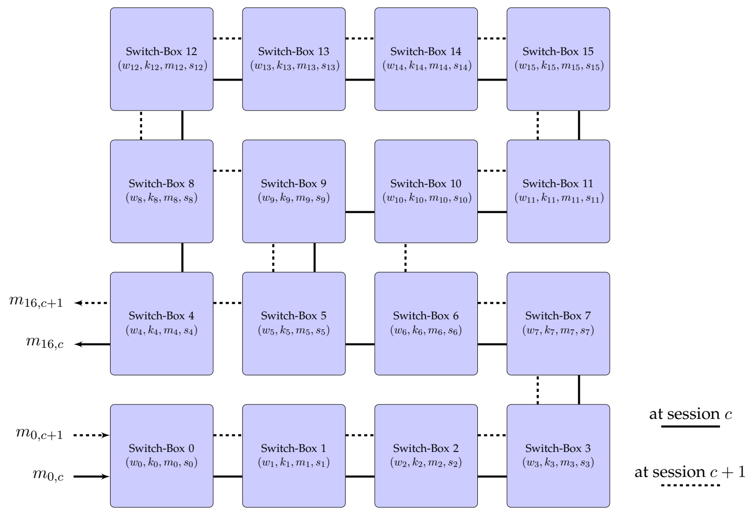

Switch-Box 12 $(w_{12}, k_{12}, m_{12}, s_{12})$
Switch-Box 13 $(w_{13}, k_{13}, m_{13}, s_{13})$
Switch-Box 14 $(w_{14}, k_{14}, m_{14}, s_{14})$
Switch-Box 15 $(w_{15}, k_{15}, m_{15}, s_{15})$
Switch-Box 8 (w_8, k_8, m_8, s_8)
Switch-Box 9 (w_9, k_9, m_9, s_9)
Switch-Box 10 $(w_{10}, k_{10}, m_{10}, s_{10})$
Switch-Box 11 $(w_{11}, k_{11}, m_{11}, s_{11})$
$m_{16,c+1}$
$m_{16,c}$
Switch-Box 4 (w_4, k_4, m_4, s_4)
Switch-Box 5 (w_5, k_5, m_5, s_5)
Switch-Box 6 (w_6, k_6, m_6, s_6)
Switch-Box 7 (w_7, k_7, m_7, s_7)
$m_{0,c+1}$
$m_{0,c}$
Switch-Box 0 (w_0, k_0, m_0, s_0)
Switch-Box 1 (w_1, k_1, m_1, s_1)
Switch-Box 2 (w_2, k_2, m_2, s_2)
Switch-Box 3 (w_3, k_3, m_3, s_3)
at session c
at session $c+1$

Fig. 17: 4×4 dynamic switch-box grid routed at c and $c+1$ (illustration)

[12] $p(n^3+1)$ where p is the payload size in bits.

If one of the switch-boxes is compromised then the digest output by the path will be altered with high probability and the fault will be detected by the mirror verification routine implemented in the protected processor (Fig. 18). The device could then revert to a safe mode, and sanitize sensitive data.

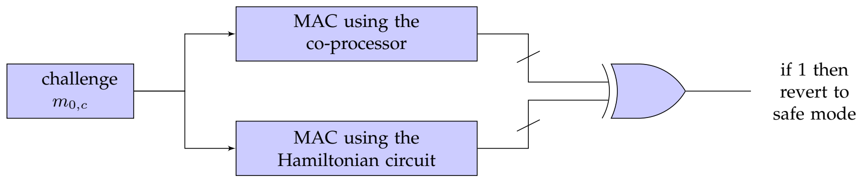

challenge
$m_{0,c}$
MAC using the co-processor
MAC using the Hamiltonian circuit
if 1 then revert to safe mode

Fig. 18: Device integrity verification scheme

The verification circuit's size essentially depends on the MAC's size and complexity. Note that the XOR gate is a weak point: if it is bypassed the entire canary becomes pointless. Luckily, the XOR is spatially protected by the Hamiltonian path that surrounds it.

5.3 Vulnerability to Focused Ion Beam (FIB) Attacks

The proposed dynamic structure complies with the Read-Proof Hardware requirements described in [10]: the structure is easy to evaluate, relatively cheap (in some case no additional masks would be required) and can't be easily removed without damaging the chip.

Even though an attacker might modify some switch-box interconnections using FIB equipment, one cannot bypass a switch-box without modifying the digest computation logic and thus triggering the canary. In theory, an attacker may microprobe the input of the first switch-box to get the reconfiguration path, feed it into an FPGA simulating the grid and re-feed the MAC into the target, thus bypassing the canary. The state function s_i implemented in each switch-box should prevent such attacks by keeping state information. Moreover, switch-boxes are defined at transistor level (first metal level): to microprobe each cell the attacker has to bypass many interconnections, making such an attack very complex. Fig. 19 describes schematically the dynamic grid concept.

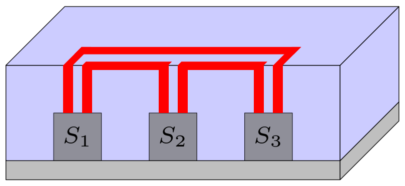


Fig. 19: Three switch-boxes embedded at substrate level with interconnections over the top layers

The successive grid configurations are precomputed by an external Hamiltonian path generator using the strategies described in Section 3. This configuration data should be stored in a non-volatile memory located under the cage.

6 Perspectives and Open Problems

Hardware canaries present an advantage with respect to analog integrity protection such as PUFs and sensors: being purely digital, hardware canaries can be integrated at the HDL-level design phase be portable across technologies. The proposed solution would, indeed, increase manufacturing and testing complexity but, being purely digital, would also increase reliability in unstable physical conditions, a common problem encountered when implementing analog sensors and PUFs.

The previous sections raise several sophistication ideas. For instance, instead of having the processor simply pick a reconfiguration route in a pre-stored table, the processor may *also re-write* the chosen route before configuring the canary with it. Devising more rewriting rules and developing lightweight heuristics to efficiently identify where to apply such rules is an interesting research direction.

Another interesting generalization is the interleaving in space of several disjoint Hamiltonian circuits. Interleaved canaries will force the attacker to overcome several spatial barriers. It is always possible to interleave a cube of size $n-1$ in a cube of size n without having the two cubes intersect each other[13] as illustrated in Fig. 20.

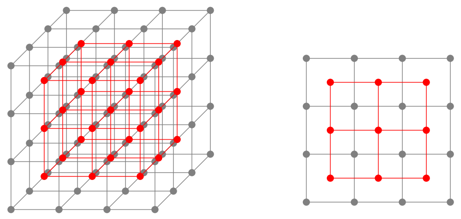


Fig. 20: A size 4 cube interleaved with a size 3 cube (3D and front view)

Fig. 21 shows the result of such a (laborious!) physical interleaving for a cube of size 4 and a cube of size 5. Note that interleaving remains compatible with a dynamic evolution of *both cubes* as canaries do not touch each other nor share any hardware (edges or vertices).

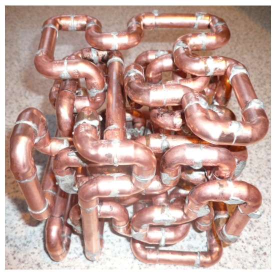


Fig. 21: Interleaving a Hamiltonian cube of size 4 and a Hamiltonian cube of size 5

Finally, functions F for which the evaluation of $F(x) = (F_{n^3-1} \circ \ldots \circ F_0)(x)$ is faster than n^3 individual evaluations of F_i are desirable for efficiency reasons. XOR, bit permutation, addition, multiplication and exponentiation (*e.g.* modulo 251) all fall into this category[14]. Note that $F_i(x) = k_i \times x^{k'_i} \bmod p$ works as well.

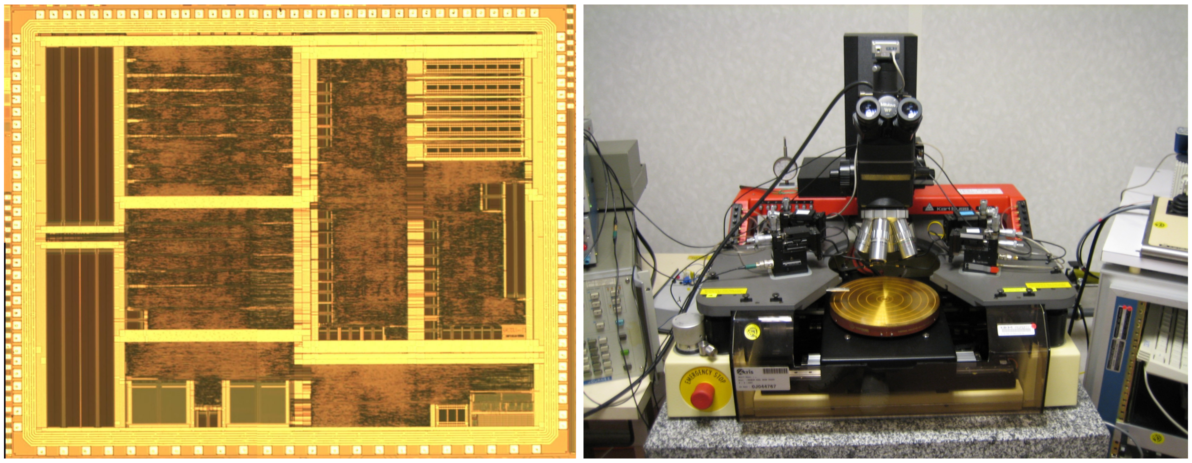
In the first dynamic prototype the F_i's will be formed of XORs and bit permutations. Devising computational shortcuts taking into account an evolving internal state $s_{i,c}$ are also desirable.

[13] Remove the (k,k,k) point from the center of the odd cube as explained before.

[14] Evidently, input should be nonzero for multiplication, nonzero and $\neq 1$ for exponentiation etc.

A Chip Probing

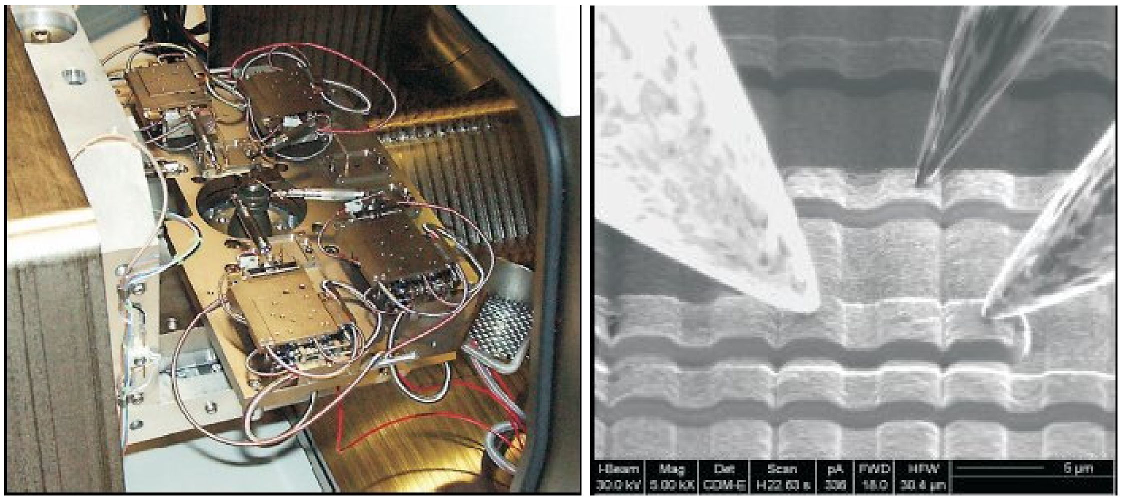
An optical probing station (Fig. 22) allow an attacker to feed and collect signals from a target chip through its input and output pins.



(a) (b)

Fig. 22: A decapsulated chip with an exposed pad ring (a) and an optical probing station (b).

To probe directly metal lines (Fig. 23b), an attacker must use more sophisticated tools. Using a probing station installed in a SEM (Fig. 23a), it is possible to probe apparent metal lines. To probe hidden metal lines, a FIB must be used to remove layers and create new connections.



(a) (b)

Fig. 23: A probing station mounted in a SEM and a tilted SEM view of circuit lines being probed.

B Constructing Cubes of Size $n = 4\ell + 1$

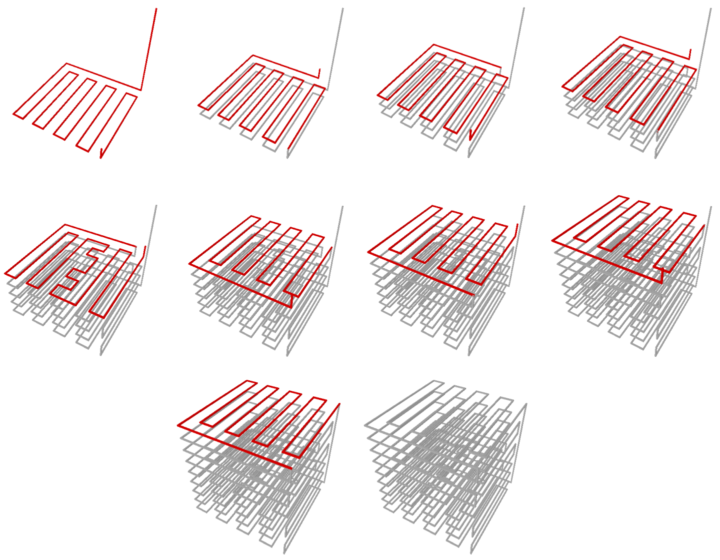


Fig. 24: Constructive proof that cubes of size $n = 4\ell + 1$ exist for all $\ell \neq 0$

C The Entropy of a Random Hamiltonian Path

The entropy of a random Hamiltonian path generator $\mathcal{G}(n)$ for cubes of size n is simply:

$$H(\mathcal{G}(n)) = -\sum_{i=1}^{u_n} p_i \log_2(p_i)$$

Where u_n denotes the number of distinct paths constructible within a cube of size n and p_i is the probability that, when queried, $\mathcal{G}(n)$ will output path number i. This definition is however of little use given that we know of no estimates of u_n in the literature (let alone estimates of the p_i's for the algorithms proposed in this paper).

D Circuit Folding

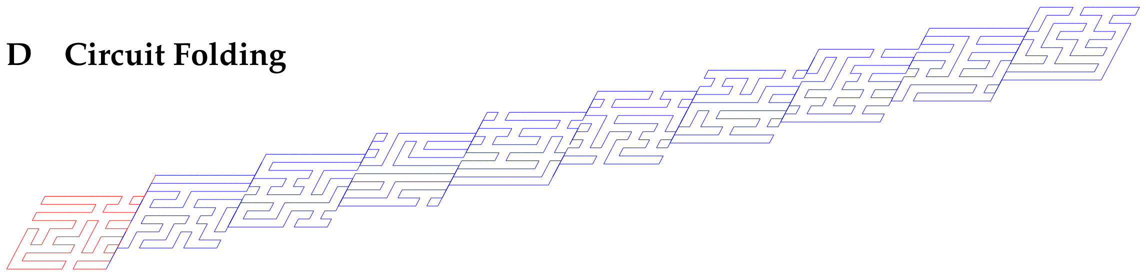


Fig. 25: 10×100 Hamiltonian rectangle $\mathcal{L}$ prepared to be folded

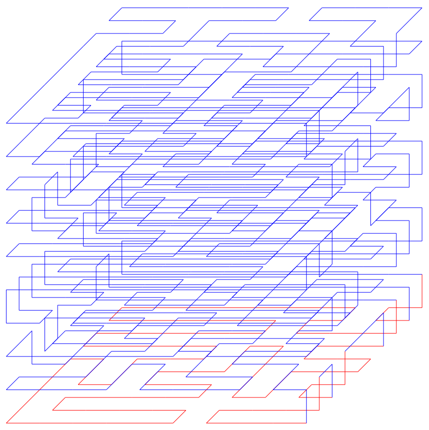


Fig. 26: $10 \times 10 \times 10$ Hamiltonian cube $\varphi(\mathcal{L})$ obtained by folding Fig. 25

E Random Cube Association

Five elementary cubes in Fig. 28 are shown in red to underline that all cubes forming Fig. 28 are still disjoint.

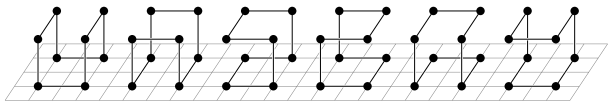


Fig. 27: The six elementary Hamiltonian cubes of size 2

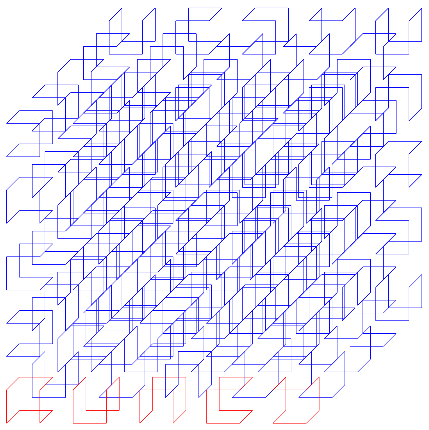


Fig. 28: Elementary 2×2 cubes filling the lattice of points forming a cube of size $n = 10$

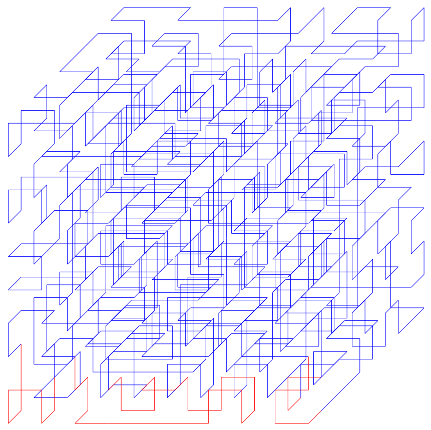


Fig. 29: An $n = 10$ Hamiltonian path obtained by randomly associating Fig. 28

F Constrained Execution for Several γ Values

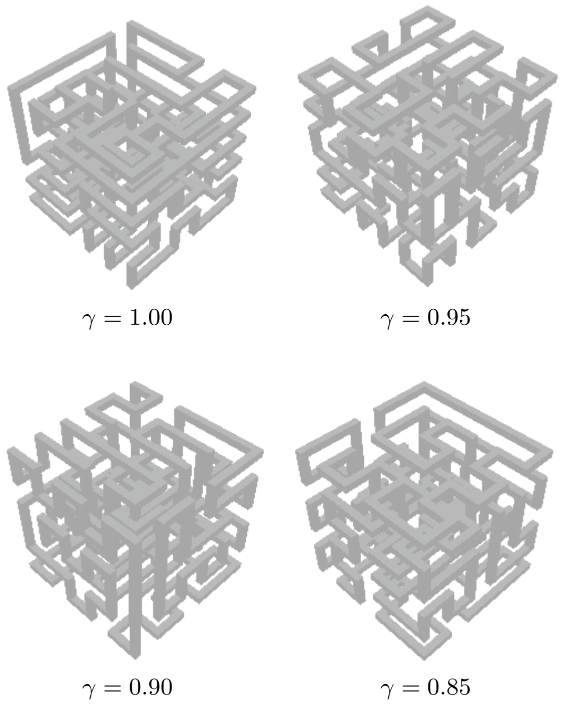


Fig. 30: Structures obtained for several γ values.

G Experimental Pre-Silicon Models

Having obtained several construction plans, we decided to try and construct concrete examples using copper supplies before migrating to silicon. We used an industrial robot to cut 12mm∅ copper segments of various sizes. A measurement of the dimensions of off-the-shelf right angle connectors (Fig. 31) revealed that if a 1-unit segment is h millimeters long, then an i-unit segment has to measure $(h + 16) \times i - 16$ millimeters.

Fig. 31: Angle connector

G.1 Visualizing and Layering

Layering and visualizing the prototypes (and chip metal layers) was done using an *ad-hoc* software suite written in C and in Processing[15]. The software allows decomposing a 3D structure into layers and rotating it for inspection.

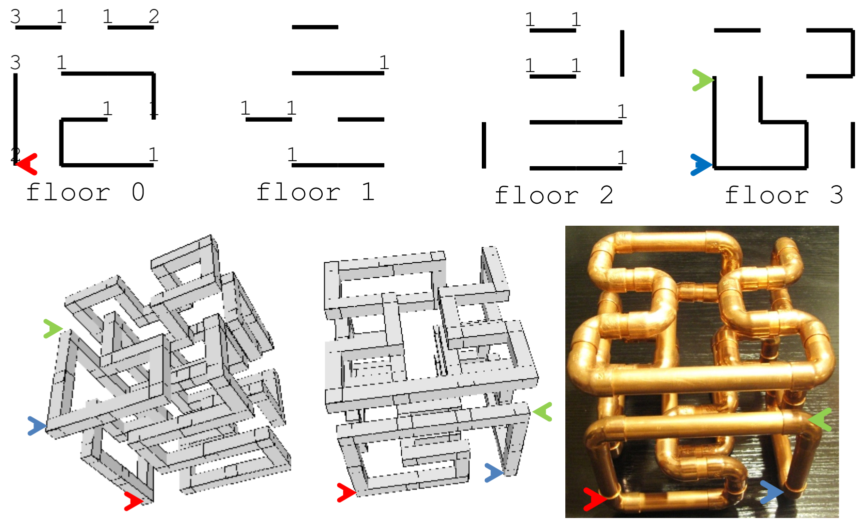


Fig. 32: Layering, visualizing and constructing the prototypes.

G.2 Assembly Options

Segments were assembled using several techniques ranging from soldering to super-glue. The disadvantage of welding was the risk of unsoldering an angle connector while soldering the nearby one (and this indeed happened at times). Super-glue happened to be less risky but called for dexterity as the glue would harden in a couple of seconds and thereby make any further correction impossible. All in all super-glue was preferred and allowed the generation of a variety of experimental pre-silicon cubes shown in Fig. 33. 3D printing using stereolitography or thermoplastic extrusion (fused deposition modeling) were considered as well.

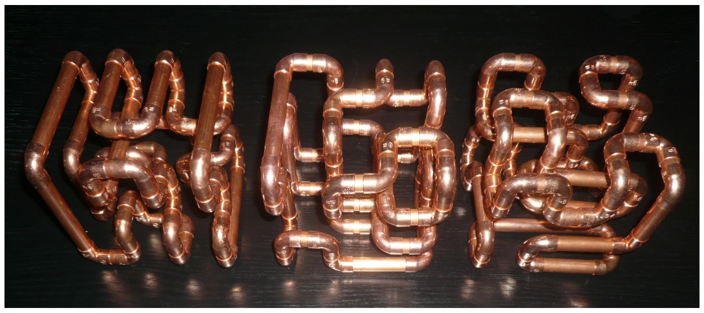


Fig. 33: Experimental pre-silicon cubes

[15] `http://processing.org/`

G.3 Angular Deviation Problems

When assembling a 3D cage with glue (or soldering) it is very easy to make mistakes that add-up. A small angular deviation in the assembly of an angle along the x axis will mix with a small angular deviation along the y axis and quickly result in a distorted cage. To avoid this, we assembled structures using a process that consists in slicing the generated structure along the three axes and identifying the longest planar (2D) parts in the target construction. Each planar part is laid on a table and is hence glued according to two axes only (*i.e.* with a lesser degree of freedom). This makes 2D angular errors avoidable (in theory) or at least much smaller (in practice). As the 2D parts are dry and ready, they are glued to each other to form the final cage. As it turns out, this indeed yields much straighter constructions.

References

[1] C. Ababei, Y. Feng, B. Goplen, H. Mogal, T. Zhang, K. Bazargan and S. Sapatnekar, *Placement and Routing in 3D Integrated Circuits*, IEEE Design and Test of Computers, 22(6), pp. 520-531, Nov/Dec 2005.

[2] A.J. Alexander, J.P. Cohoon, J.L. Colflesh, J. Karro, E. Peters and G. Robins, *Placement and routing for three-dimensional FPGAs*, Fourth Canadian Workshop on Field-Programmable Devices, pp. 11-18, May 1996.

[3] B. Bollobás, *Graph Theory: An Introductory Course*, New York: Springer-Verlag, p. 12, 1979.

[4] A. Dharwadker, *The Hamiltonian Circuit Algorithm*, Proceedings of the Institute of Mathematics, p. 32, 2011.

[5] R. Dickau, *Hilbert and Moore 3D Fractal Curves*, The Wolfram Demonstrations Project, http://demonstrations.wolfram.com/HilbertAndMoore3DFractalCurves

[6] K. Goossens, J. van Meerbergen, A. Peeters and P. Wielage, *Networks on Silicon: Combinig Best-Effort and Guaranteed Services*, Proceedings of Design Automation and Test Conference (DATE), pp. 423-425, 2002.

[7] J. Kim, I. Verbauwhede and M.-C. F. Chang, *Design of an Interconnect Architecture and Signaling Technology for Parallelism in Communication*, IEEE Trans. VLSI Syst. 15(8), pp. 881-894, 2007.

[8] E. H. Moore, *On Certain Crinkly Curves*, Trans. Amer. Math Soc., 1, pp. 72-90, 1900.

[9] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, *Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip*, Proceedings of Design, Automation and Test Conference in Europe (DATE), pp. 350–355, March 2003.

[10] P. Tuyls, G. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, R. Wolters, *Read-Proof Hardware from Protective Coatings*, Cryptographic Hardware and Embedded Systems, CHES 2006, LNCS vol. 4249, pp. 369-383, Springer, 2006.

[11] J. Valamehr, T. Huffmire, C. Irvine, R. Kastner, Ç. Koç, T. Levin, and T. Sherwood, *A Qualitative security Analysis of a New Class of 3-D Integrated Crypto Co-processors*. Festschrift Jean-Jacques Quisquater, LNCS vol. 6805, pp. 364-382, Springer, 2011.

[12] I. Verbauwhede and M.-C. F. Chang, *Reconfigurable interconnect for next generation systems*. The Fourth IEEE/ACM International Workshop on System-Level Interconnect Prediction (SLIP 2002), April 6-7, 2002, Proceedings, pp. 71-74, 2002.