Covert Computation in Self-Assembled Circuits

Angel A. Cantu
Department of Computer Science, University of Texas - Rio Grande Valley
angel.cantu01@utrgv.edu

Austin Luchsinger
Department of Computer Science, University of Texas - Rio Grande Valley
austin.luchsinger01@utrgv.edu

Robert Schweller
Department of Computer Science, University of Texas - Rio Grande Valley
robert.schweller@utrgv.edu

Tim Wylie
Department of Computer Science, University of Texas - Rio Grande Valley
timothy.wylie@utrgv.edu

Abstract

Traditionally, computation within self-assembly models is hard to conceal because the self-assembly process generates a crystalline assembly whose computational history is inherently part of the structure itself. With no way to remove information from the computation, this computational model offers a unique problem: how can computational input and computation be hidden while still computing and reporting the final output? Designing such systems is inherently motivated by privacy concerns in biomedical computing and applications in cryptography.

In this paper we propose the problem of performing "covert computation" within tile self-assembly that seeks to design self-assembly systems that "conceal" both the input and computational history of performed computations. We achieve these results within the growth-only restricted abstract tile assembly model (aTAM) with positive and negative interactions. We show that general-case covert computation is possible by implementing a set of basic covert logic gates capable of simulating any circuit (functionally complete). To further motivate the study of covert computation, we apply our new framework to resolve an outstanding complexity question; we use our covert circuitry to show that the unique assembly verification problem within the growth-only aTAM with negative interactions is coNP-complete.

2012 ACM Subject Classification General and reference → General literature; General and reference

Keywords and phrases self-assembly, covert circuits

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.26

Funding This research was supported in part by National Science Foundation Grant CCF-1817602.

1 Introduction

Since the discovery of DNA over half a century ago, humans have been continually working to understand and harness the vast amount of information it contains. The Human Genome Project [17], which began in 1990 and took a decade, was the first major attempt to fully sequence the human genome. In the years since, sequencing has become extremely cheap and easy, and our ability to manipulate DNA has emerged as a central tool for many applications related to nanotechnology and biomedical engineering.

Although this progress has many benefits, as we learn more about the information, we also must be careful with the shared data. There are databases of anonymous DNA sequences, which can sometimes be deanonymized with only small amounts of information such as a surname [14], or by reconstructing physical features from the DNA [7]. In order to address these issues, there has been work on cryptographic schemes aimed at obscuring results related to DNA or the input/output [8, 12, 15, 27].

In this work we take the first steps in addressing some of these issues within self-assembling systems by proposing a new style of computation termed *covert computation* with important motivations for private biomedical computing and cryptography. Self-assembly is the process by which systems of simple objects autonomously organize themselves through local interactions into larger, more complex objects. Understanding how to design and efficiently program molecular self-assembly systems is fundamental for the future of nanotechnology. The Abstract Tile Self-Assembly Model (aTAM) [9, 19], motivated by a DNA implementaiton [13], has become the premiere model for the study of the computational power of self-assembling systems. In the aTAM, system monomers are modeled by four-sided Wang tiles which randomly combine and attach if the respective bonding domains on tile edges are sufficiently strong. The aTAM is known to be computationally universal [26] and intrinsically universal [11].

Covert Computation. As a computational model, tile self-assembly differs from traditional models of computation in that computational steps are defined by permanently placing particular tile types at specific locations in geometric space. A history of each computational step is thereby recorded in the final assembled structure. This presents a unique problem to this type of computation: is it possible to conceal the input and history of a computation within the final assembly while still computing and reporting the output of the computation? Concealing the computational histories of the self-assembly process in this way requires designing a computational system which encodes computational steps in the *order* of tile placement, rather than the type and location of tile placements. We use the term *covert*[1] to describe this concealment of inputs and computational histories. This method of computing is different than previous tile self-assembly computing methods and requires novel techniques.

Also, while the reader may notice many parallels between our work and traditional secure multiparty computation [5], it should be made clear that our main result is the secure computation of a function with only a single party. The challenges presented above make this an interesting problem for tile self-assembly.

Motivation. The concept of covert computation within self-assembly has many potential applications. We briefly outline a few biomedical computing applications. Consider a set of diagnostic tiles sent to a patient as a droplet of DNA to which the patient adds some biological input such as a blood sample. From this the diagnostic system could compute some desired function that outputs specific diagnostic statistics. The patient sends the combined product to a medical facility for interpretation. With covert computation, only the results can be read by the lab and the user's biological input is obscured ensuring privacy.

Another potential use involves implementing a cryptography system within a molecular computing framework. The ability to covertly compute allows users to provide a personal key input that may be combined with a publicly available covert system where the combination verifies some computable property of the input key without revealing any additional details of the key. This style of cryptographic scheme fits well when the input keys are biological based inputs.

A final potential biological application might be engineering a system for unlocking key biological properties within bio-engineered crops. For example, by releasing a hidden "key"

Model	Negative Glues	Detachment	Complexity	Theorem
aTAM	No	No	$O(\lvert A\rvert^2 + \lvert A\rvert\lvert T\rvert)$	Thm. 3.2 in [1]
aTAM	Yes	No	coNP-complete	Thm. 3

[1] It is important to note that the term *covert* has specific meaning in cryptography which does not apply here.

aTAM	Yes	Yes	Undecidable	[10]

Table 1 The complexity of Unique Assembly Verification in the aTAM in relation to negative glues. $|A|$ refers to the size of an assembly and $|T|$ is the number of tile types.

input, covert computation might allow a field of crops to become fertile. A company owning the patent on this type of activation might desire the security of ensuring that the release key cannot be deciphered from the activated crop based on a covert molecular computation.

The final motivation of covert computation is within algorithmic self-assembly. We believe the concept of covert computation is fundamental and hope that our novel design techniques will be applicable to a number of future problems in the area. As evidence towards this, we apply our techniques to resolve the complexity of the fundamental question of verifying whether a tile system uniquely assembles a given assembly within the growth-only negative-glue aTAM.

Contributions. After formally defining the concept of covert computation in tile self-assembly, we implement several covert logic gates within the negative-glue growth-only abstract Tile Assembly Model (this growth-only restriction to negative glues has been seen in the 2HAM [4], and negative glues in tile assembly have received extensive study [3, 10, 18, 21, 20, 22, 23]), and show these gates may be combined to create general circuits, thereby showing that general covert computation is possible. Finally, we apply our techniques and framework to address the fundamental problem of deciding if a negative-glue aTAM system uniquely produces a given assembly. We show this problem is coNP-complete. Table 1 outlines how our result compares to what was previously known.

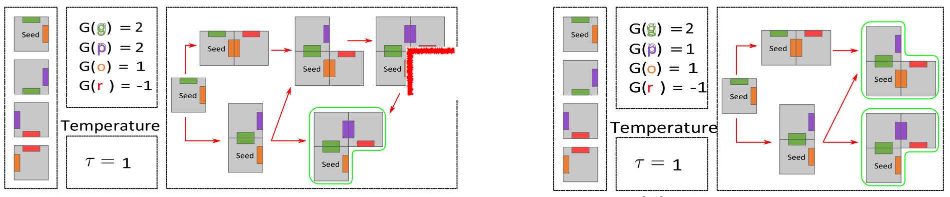
2 Definitions

We begin with an overview of the Abstract Tile-Assembly Model (aTAM) and then give the new definitions introducing covert computation. Due to space constraints, we only give a high-level overview of the aTAM. Formal definitions for all the concepts are in Appendix 8.

2.1 Abstract Tile Assembly Model

Figure 1 gives a high-level overview of the models with a couple of example systems. Essentially, we have non-rotating square *tiles* that have a *glue* label on each edge. The tile with its labels is a *tile type*. The *tile set* is all the tile types. A glue function determines the strength of matching glue labels. An *assembly* is a single tile or a finite set of tiles that have combined via the glues. If the combined strength of the glue labels of a single attaching tile to an assembly is greater than or equal to the *temperature* τ, the tile may attach. A *producible* assembly is any assembly that might be achieved by beginning with the *seed* (the specified starting assembly) and attaching tiles. A producible assembly is further said to be *terminal* if no further tile attachment is possible. A tile system is said to *uniquely produce* a (terminal) assembly A if all producible assemblies will eventually grow into A. A tile system is formally represented as an ordered triplet $\gamma = (T,s,\tau)$ representing the tile set, seed assembly, and temperature parameter of the system respectively.

Tile Set Glues Producible Assemblies Tile Set Glues Producible Assemblies



(a) Negative aTAM **(b)** Growth aTAM

Figure 1 High-level overview of the aTAM with repulsive forces. Both systems have tiles that can attach to the seed tile given they can attach with τ strength. The arrows show the possible assembly paths from the seed tile with the terminal assembly being outlined. (a) A negative aTAM system that has a possible assembly path causing disassembly. One path is growth-only, but the other path can attach the tile with the purple/red glues, which causes the orange/red tile to become unstable and detach. (b) A growth only aTAM system where negative glues are used to block, but never cause disassembly. The only difference is that the purple glue attaches with strength 1, $G(p) = 1$. This yields two possible terminal assemblies, neither of which include disassembly.

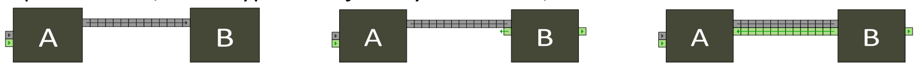
In a standard aTAM system, all glues are positive integral values, but here we look at the negative aTAM where the glues may be negative/repulsive. Such repulsive forces may be used to block the attachment of tiles despite the presence of strong attractive glues. Moreover, the inclusion of repulsive forces may yield unstable producible assemblies where a subassembly could detach because it no longer has enough binding strength. While this type of detachment has been studied in the literature [10, 23], we avoid this feature in this work as it's inclusion drastically changes the complexity of the model by making most types of verification problem undecidable, and may require more sophisticated techniques for experimental implementation. Thus, we consider a system to be a *valid growth-only* system if all producible assemblies are τ-stable. In this paper we restrict our consideration to valid growth-only systems.

2.2 Covert Computation

Here, we provide formal definitions for computing a function with a tile system, and the further requirement for covert computation of a function. Our formulation of computing functions is based on that of [16] but modified to allow for each bit to be represented by a sub-assembly potentially larger than a single tile.

Informally, a Tile Assembly Computer (TAC) for a function f consists of a set of tiles, along with a format for both input and output. The input format is a specification for how to build an input seed to the system that encodes the desired input bit-string for function f. We require that each bit of the input be mapped to one of two assemblies for the respective bit position: a sub-assembly representing "0", or a sub-assembly representing "1". The input seed for the entire string is the union of all these sub-assemblies. This seed, along with the tile set of the TAC, forms a tile system. The output of the computation is the final terminal assembly this system builds. To interpret what bit-string is represented by the output, a second *output* format specifies a pair of sub-assemblies for each bit. The bitstring represented by the union of these subassemblies within the constructed assembly is the output of the system.

For a TAC to *covertly* compute f, the TAC must compute f and produce a unique assembly for each possible output of f. We note that our formulation for providing input and interpreting output is quite rigid and may prohibit more exotic forms of computation. We acknowledge this, but caution that any formulation must take care to prevent "cheating" that could allow the output of a function to be partially or completely encoded within the input, for example. To prevent this, some type of *uniformity* constraint, similar to what



(a) **(b)** **(c)**

Figure 2 Backfilling in covert computation. Given two gadgets A and B. (a) If true is output from Gadget A, that wire assembles to the next gadget. (b) Gadget B builds, and based on its function, outputs the true or false wire (false in this case). Once it received the input, it backfills the false wire towards A. (c) The false wire finishes assembling and both Gadget A and B have true and false paths filled. The true output wire of Gadget B will be backfilled from the next gadget. In this way, the input to B/output from A is "hidden."

is considered in circuit complexity [25], should be enforced. We now provide the formal definitions of function computing and covert computation.

Input/Output Templates. An n-bit input/output template over tile set T is a sequence of ordered pairs of assemblies over T: $A = (A_{0,0}, A_{0,1}), \ldots, (A_{n-1,0}, A_{n-1,1})$. For a given n-bit string $b = b_0, \ldots, b_{n-1}$ and n-bit input/output template A, the *representation* of b with respect to A is the assembly $A(b) = \bigcup_i A_{i,b_i}$. A template is valid for a temperature τ if this union never contains overlaps for any choice of b, and is always τ-stable. An assembly $B \supseteq A(b)$, which contains $A(b)$ as a subassembly, is said to represent b as long as $A(d) * B$ for any $d 6= b$.

Function Computing Problem. A *tile assembly computer* (TAC) is an ordered quadruple = = (T,I,O,τ) where T is a tile set, I is an n-bit input template, and O is a k-bit output template. A TAC is said to compute function f : [redacted] if for any b [redacted] and c [redacted] such that $f(b) = c$, then the tile system $\Gamma_{=,b} = (T,I(b),\tau)$ uniquely assembles a set of assemblies which all represent c with respect to template O.

Covert Computation. A TAC *covertly* computes a function $f(b) = c$ if 1) it computes f, and 2) for each c, there exists a unique assembly A_c such that for all b, where $f(b) = c$, the system $\Gamma_{=,b} = (T,I(b),\tau)$ uniquely produces A_c. In other words, A_c is determined by c, and every b where $f(b) = c$ has the exact same final assemby.

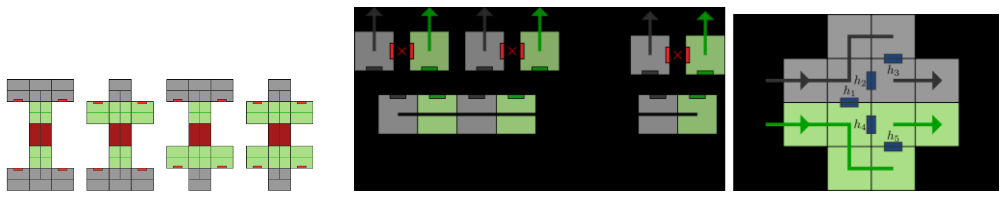
3 Covert Circuits

Here we cover the machinery for making covert gadgets and the covert gadgets needed for functional completeness in circuits based on a dual-rail logic implementation: variables, wires, fanouts, and NANDs. We cover a NOT gate as a primitive used in the NAND construction. Traditionally, a crossover is also given, and we discuss why this is unnecessary in Section 4. For simplicity, we give some other common gates in Section 5.

Some Conventions. All solid lines through two neighboring tiles indicate strength-2 glues between them. The arrows indicate the build order (which may branch). Blue single glues are strength 1, and red are strength -1. Following the variable gadget (Figure 3b), all variables have a true and false path adjacent to each other (dual-rail logic), but only one may be traversed at a time until the next gadget. The true value is always to the left or on top of the false value, and for most gadgets, the true input is colored grey while the false input is colored green. Once a variable wire, true or false, reaches the next gadget, the unused variable wire is *backfilled* so that both wires are present. This is a key concept used in all constructions and is further explained in Figure 2.

3.1 Variables and Wires

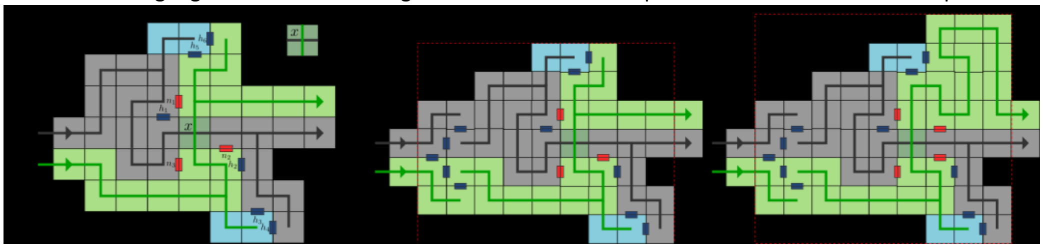
A variable in our system is represented by two lines of connected tiles where only one exists at a time when the wire is in use (dual rail). Figure 3a shows an example of the possible input seeds on 2-bits used in a half-adder. Figure 3b demonstrates how the variables might



(a) Possible Input Seeds **(b)** Variable **(c)** Logic Diode

Figure 3 (a) Example of the 4 possible input seeds for a half-adder from Section 5.1. (a) Variables are represented by a true and a false line where only one may exist. The variables build off the seed, but only the t_i or the f_i tile may attach due to the negative glue between the two tiles. (b)

A gadget referred to as a logic diode. This ensures input from one direction and stops tiles from



(a) Basic NOT **(b)** NOT **(c)** H-NOT

Figure 4 (a) Basic NOT gate (b) NOT gate with the logic diode on the input (c) A covert NOT gate with an additional negative horizontal glue on the output to prevent incorrect backfilling. This modification is needed when using this gate for the construction of the NAND gate.

be set nondeterministically, although generally the specific bits desired would already be attached as part of the input seed (as in Figure 3a). Each variable v_i has a sequence of tiles t_i representing a true setting and f_i a false setting. The first tiles have a negative glue of strength −1 meaning only the t_i or the f_i tile can attach. The other shown glues are strength 2. Once the variable is set, the setting travels to the gadget as a *wire*.

The variable setup in Figure 3b is used in one of two ways: In the case of providing an input to a covert computation, this variable setup defines the *input template* for the computation, with the seed for a given binary input being the seed assembly with either a true or false tile (but not both) placed at each bit position. An example system (a half-adder) with a big seed input is shown in Section 5. Alternatively, the seed begins as a single seed tile that nondeterministically creates a valid input over all possible n-bit inputs. This approach is used in Section 4 to show coNP-completeness for unique assembly verification.

Figure 3c shows what we refer to as a *logic diode*, and prevents timing issues. These appear in every gadget and serves two purposes: if backfilling, this stops the filling at the gadget level so it does not backfill a wire that has not been set, and second it ensures that a gadget must have input from the wire. All shown glues are strength 1 and the lines are strength 2. This gadget is important for later constructions. The properties of the gadget are in Appendix 9.1.

3.2 Covert NOT Gadget

The first covert gadget we introduce is a NOT gadget. This gadget displays some of the key insights needed for covert computation, such as how blocking with negative glue adds power to the system. The NOT gadget is also used as a submodule within our NAND gadget. The NOT gadget in Figure 4a is the basic gadget with 4b only adding the logic diode on the input to ensure no backfill happens past the gadget and that the gadget had input.

Given the variables and wires work as shown, the difficulty in a dual-rail NOT is that there must be at least one crossing tile that both the true and false paths place. This tile can

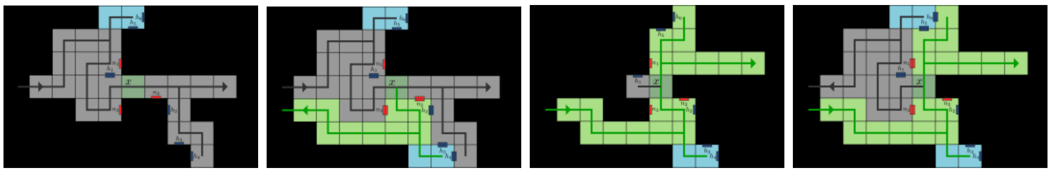
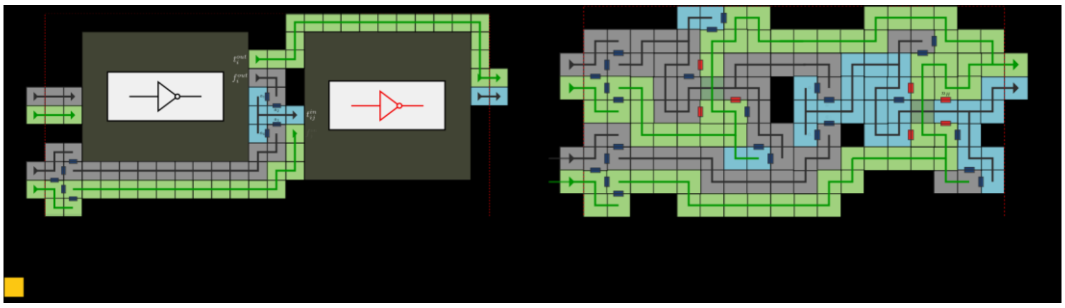

(a) True Input 1 **(b)** True Input 2 **(c)** False Input 1 **(d)** False Input 2

Figure 5 (a) A NOT gadget with true input t_i^{in} and output f_i^{out}. The true output can not place from tile x due to the negative glues n_1 and n_3 of strength −1. (b) Once the NOT gadget passes the false output, glues h_3,h_4 cooperatively allow the false portion and wire to backfill. Glue h_2 is needed to fill in the tile with n_2. (c) A NOT gadget with false input f_i^{in} and output $t^{out}{}_i$. The false output can not attach due to the negative glue n_2. The tile to the west of x may attach, but due to glue n_3, no other tile can attach. (d) Once the NOT gadget passes the true output, glues h_5,h_6 allow the true portion and wire to backfill. Glue h_1 is needed to counteract the n_1 glue when backfilling that tile.


NOT blocks are shown outlined in Figures 4b and 4c. The left box is the standard NOT gadget and the right box is the H-NOT gadget. (b) The full NAND gate with the two NOT gadgets filled in and compacted.

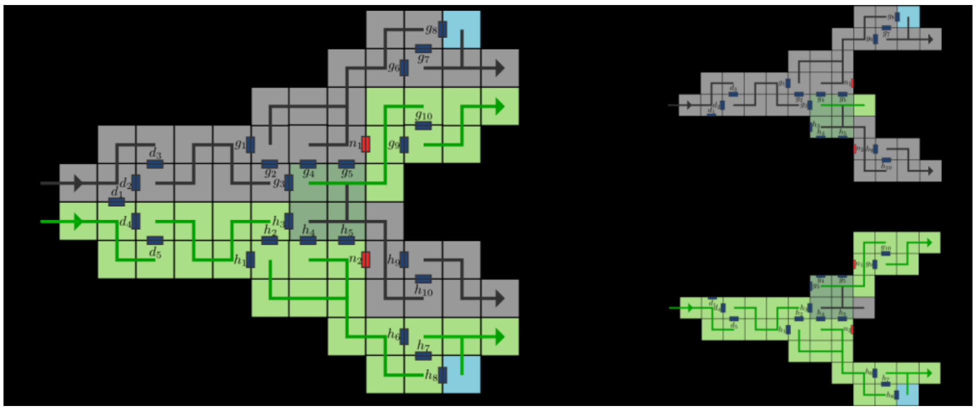
be thought of as where the signals cross or switch. Figure 4a shows the basic NOT gadgets, and the tile shared by both paths is labelled x. The negative glues allow blocking around this tile so that only one path is possible once x is placed.

The specific properties that must hold are covered in Appendix 9.2. The conditions guarantee that the gadget works correctly and that the gadget is covert (the gadget looks indistinguishable before the output regardless of the input), and that the backfill works correctly. Figure 5 discusses these elements and walks through how the true/false inputs block and crossover correctly. The Figure does not show the logic diode though.

3.3 Covert NAND Gadget

The basic idea for the NAND gadget is to flip one of the inputs using a covert NOT, and then we can compare the two true input lines to see if both inputs were true. Since a NAND is false only when both inputs are true, this is the only path that should result in a false output. The basic idea for the gadget is shown in Figure 6a with a representative block for the NOT gadget already discussed. The second NOT block is the modified NOT gadget (H-NOT) from Figure 4c. Both false inputs are routed to the true output. One must go through another NOT in order to flip to the top output position, while the other false line skips this NOT and ties directly to the true output.

For clarity, we discuss the block diagram first, then address a few technical issues in the full version. The full list of properties necessary and sufficient for the covert NAND is given in Appendix 9.3. Once we flip the top input, we can use cooperative binding to compare the two true inputs, and only if both are true do we send it as true into the second NOT block (so the gadget outputs false). All other input combinations output true.



(a) FANOUT **(c)** False Input

Figure 7 (a) FANOUT gadget. (b) True input wire for the FANOUT gadget t^{in}_i results in output wires $t^{out}_{i}{}^{1}$ and $t^{out}_{i}{}^{2}$. (c) False input wire for the FANOUT gadget f_i^{in} results in output wires $f_i^{out_1}$ and f_{iout_2}.

We will show why NOT and H-NOT are both necessary. Looking at Figure 6b, the negative glue n_H is necessary in H-NOT to ensure that t^{out}_i, which skips the second NOT gadget, does not set the output t^{out}_{ij}, and then also set f_{ij}^{out} based on the assembly order. Essentially, this protects from incorrect backfilling and setting both outputs. However, the n_H glue should not exist in the standard NOT gadget, or it may backfill and could cause a tile to break off depending on build order. Given we want a purely growth model, this would not be allowed. It is possible to create a single NOT that incorporates these properties, but we prefer to avoid the added complexity.

Finally, the logic diodes on the inputs (Figure 3c) ensure that if we only have one input, the gadget does not backfill down the other input wire. Even if the gadget has already been set, that input will wait until either the true or false wire comes before backfilling the wire.

3.4 Covert FANOUT Gadget

The FANOUT gadget needs to duplicate the geometric wire, and also needs to only backfill once both outgoing wires have backfilled. Figure 7a shows the FANOUT gadget. Similar to the NOT, there is a shared set of tiles placed by both the true and the false path. Figures 7b and 7c show the true and false paths without any backfilling, respectively. The necessary conditions decscribing the gadget are in Appendex 9.4.

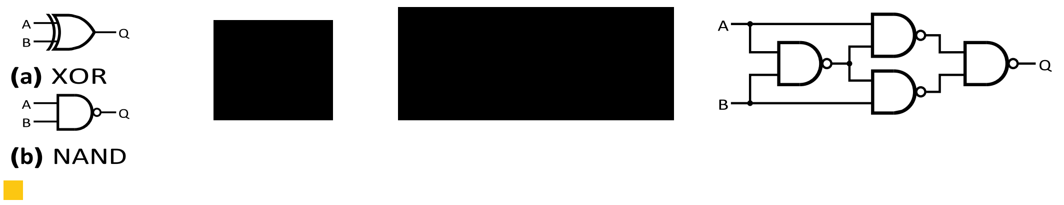
4 Covert Computation and Unique Assembly Verification

In this section we establish our main results related to covert computation in self-assembly systems. We first utilize our covert circuitry to show that any function is covertly computable (Thm. 1). We then apply covert circuitry to show that the open problem of Unique Assembly Verification within the growth-only negative glue aTAM is coNP-complete (Thm. 3).

Theorem 1. *For any function f computed by a boolean circuit, there exists a tile assembly computer (TAC) that covertly computes f.*

Proof. The proof of this theorem consists of a direct simulation of boolean circuits by way of a series of covert gadget implementations for various logic gates and how to connect them. The proof follows from the gadgets and machinery given in Section 3. J

We now prove that Unique Assembly Verification (UAV) in a growth-only negative glue aTAM system is coNP-complete by utilizing our covert gadgets. Without the growth-



(c) Crossover **(d)** XOR Crossover **(e)** XOR with NANDs

Figure 8 Constructing planar crossover gadgets with NAND gates. (a) XOR symbol. (b) NAND symbol. (c) Two wires in a circuit that cross making it non-planar. (d) A planar circuit using XOR gates that act as a crossover. (e) A planar circuit using only NAND gates that implement an XOR gate.

only constraint, UAV in the atam with negative glues is undecidable as a Turing machine simulation could use negative interactions to break down produced assemblies into a final unique terminal assembly exactly when the Turing machine halts [10]. With no negative glues however, the problem is in P [1]. We prove that with the ability to temporarily block, the problem becomes coNP-complete. This result is achieved with a reduction from Circuit SAT. Unique Assembly Verification in our model is formally defined as follows:

B Problem (Unique Assembly Verification (growth only)). Given a tile-system $\Gamma = (T,S,\tau)$ with the promise that it is a growth-only system, and an assembly A. Does Γ uniquely assemble A?

A reduction from Circuit SAT generally requires a functionally universal set of gates and variable, wire, fanout, and crossover gadgets. Both NAND and NOR are functionally complete gates, so given either, all gates can be made. A crossover gadget is redundant since it can be made with XOR gates and XOR gates can be made with NAND gates [24]. Figure 8 shows this derivation. Finally, Circuit SAT requires a DAG, and thus there are no cycles, and so the gadgets can be topologically sorted so that there are no crossovers that cause a loop (the output of a gadget can not crossover one of its input lines). Thus, a reduction from Planar Circuit SAT is equivalent to a reduction from Circuit SAT.

I **Definition 2** (Planar Circuit SAT)**.** *Instance: A planar directed acyclic graph (DAG) $G = (V,E)$ with n boolean inputs, one output, and all gates are NAND gates (or NOR gates). Every $v \in V$ is either a NAND gate ($deg^-(v) = 2$, $deg^+(v) = 1$) or a fanout ($deg^-(v) = 1$, $deg^+(v) = 2$). The source vertices, $v_i \in V$ s.t. $deg^-(v_i) = 0$ and $1 \leq i \leq n$, are the variables.*

The sink vertex, $s \in V$ s.t. $deg^+(v_i) = 0$ is the "output" of the boolean circuit.

Question: Does there exist a setting of the inputs such that the output to the circuit is 1?

I **Theorem 3.** *Unique Assembly Verification in the aTAM with repulsive forces in a growth only system is coNP-complete.*

Proof. We first observe that Unique Assembly Verification with repulsive forces is in coNP as any failure to uniquely assemble a target assembly A comes in the form of a polynomially sized assembly that is inconsistent with A. The producibility of this assembly can be verified in polynomial time, and thus serves as a certificate for "no" instances to the UAV problem.

We now show coNP-hardness by a reduction from Planar Circuit SAT. Given an instance of planar Circuit SAT C with inputs $i_1,...,i_n$ where $i \in \{0,1\}$, i.e., a boolean circuit. By our definition we assume there are only NAND gates, fanouts, input variables and an ouput variable in the planar DAG.

For our reduction, we build a tileset T by adding tiles corresponding to the covert gadgets and connections described in Section 3. Replace each NAND gate with a unique set of tiles implementing a NAND gadget, and each FANOUT gate with a unique set of tiles implementing a FANOUT gadget. For each edge, a unique sequence of tiles is added to *T* that connects the two gadgets representing the two gates the edge connected.

This yields a tile assembly computer (TAC), = = (*T,I,O,τ*), for covertly computing the circuit *C*. The key modification to show coNP-hardness is the utilization of a seed that non-deterministically grows any one of the possible *n*-bit input seeds for this TAC, and then evaluates the circuit. If the circuit is not-satisfiable, then the final computation will be false regardless of the guessed input, and therefore will yield the unique "no" assembly of the TAC based on the fact that the circuit is computed covertly. On the other hand, if there exists some satisfying *n*-bit input, there will be at least one final assembly that differs from the "no" assembly. Thus, the "no" assembly is uniquely produced if and only if the circuit *C* is not satisfiable, thereby showing coNP-hardness.

Non-deterministic input selection. To non-deterministically form the possible input bits, we include the tile types and seed tile described in Figure 3b. The seed grows a length *O*(*n*) line with each bit being encoded by a pair of adjacent locations which expose a glue on the north edge. For each pair of positions, the presence of the left tile denotes a "1" for the respective bit, and the placement of the right tile denotes a "0". The "1" and "0" tiles share a negative strength 1 glue, making their mutual placement impossible until the covert gadgets have passed on the computed signal and backfilled. J

Given that UAV is coNP-complete with negative glues by way of covert circuitry, yet UAV is in *P* without negative glues [1], it is reasonable to conjecture that the use of negative interactions is needed to perform covert computation.

B Conjecture. For some function *f* computed by a boolean circuit, there does not exist a tile assembly computer (TAC) that covertly computes *f* in the aTAM without negative glues.

5 Further Motivation

Here, we give a few more motivating examples and some simplified gadgets. There is a lot of future work in this vein of research that is extremely relevant to modern society. We first cover the covert AND and OR gadgets.

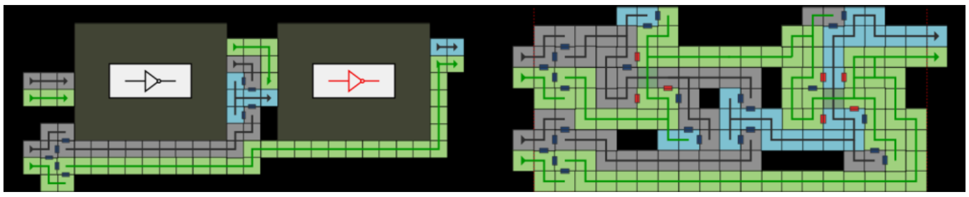
Simplified Gadgets Even though NAND gates alone are functionally complete, for some gates the circuit is larger than desired. Here, we give compact direct versions of some other useful gadgets and gates. This does not affect the complexity, but does help build a more efficient covert computation toolkit.

Covert AND Gadget. The covert AND gadget is nearly identical to the NAND gadget. The only real difference is which two inputs the second NOT takes in. Also, similar to the H-NOT needed for the NAND, we create a V-NOT, which is a NOT with one additional vertically aligned negative glue. Figure 9a shows the AND gadget with the blocks in place of NOTs for clarity, and Figure 9b shows the full gadget.

Covert OR Gadget. The covert OR gadget still uses a NOT to flip one of the inputs, but does several checks on the second flip to the point of drastically differing from a NOT. Figure 10a shows the AND gadget with the blocks in place of NOTs for clarity, and Figure 10b shows the full gadget.

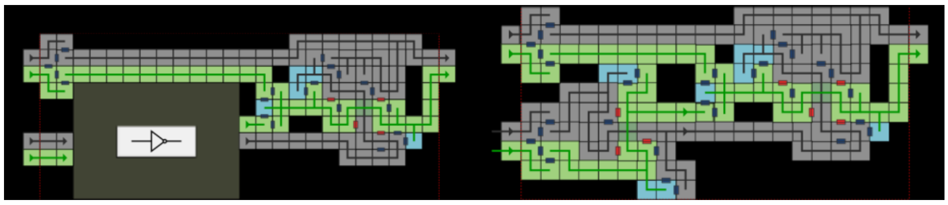
5.1 Encryption and Cryptography

Several encryption methods are based off problems that we believe to be "hard" computationally. One of the most common is factoring the product of large prime numbers, which is the basis for several encryption schemes. Although factoring may be difficult, the function



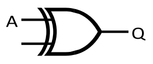
(a) AND Block Diagram **(b)** AND

Figure 9 (a) Diagram of the covert AND gate with NOTs shown as blocks. The left box is the standard NOT gadget and the right box is the V-NOT gadget (has an additional vertical glue). (b) The full AND gate with the two NOT gadgets filled in and some simplification for space.

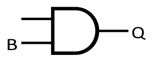


(a) Block OR **(b)** OR

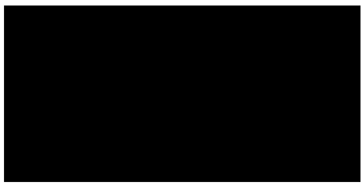
Figure 10 (a) Block diagram for the OR gadget. (b) The covert OR gadget with the NOT gadget filled in.



(a) XOR



(b) AND

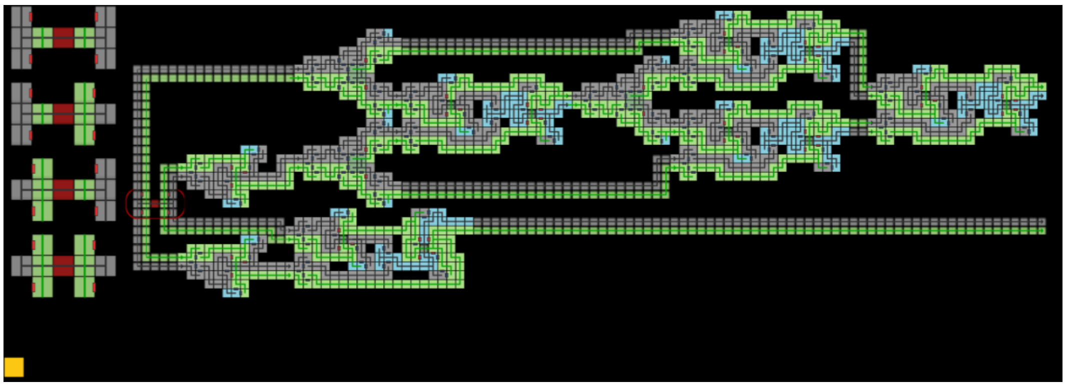


(c) Half-Adder **(d)** 2-Bit Multiplier

Figure 11 Constructing covert circuits for arithmetic building up to cryptography examples. (a) XOR symbol. (b) AND symbol. (c) A half-adder, which has two 1-bit numbers as input and a 2-bit number as output. (d) A 2-bit multiplier which has two 2-bit numbers as input and outputs a 4-bit number that is their product. This can be expanded to use two large primes resulting in a large number that would be hard to factor.

to generate the number is simple multiplication, which can be accomplished with simple circuits. Figure 11d shows a simple 6-gate circuit implementing a 2-bit number multiplier resulting in a 4-bit output number. An *n*-bit multiplier scales linearly (in the number of bits) with additional AND gates and full and half adders.

Implementing the multiplier with covert gates is not difficult, but the resulting assembly is large due to the inefficient crossover gadget used. Instead, we demonstrate a simple half-adder. The schematic for a half-adder is in Figure 11c. A covert half-adder as a TAC is shown in Figure 12b. The XOR has been replaced by the 4 NAND gates as shown in Figure 8e. Further, 3 FANOUTs were needed, an AND gadget as shown above in Section 5, and 2 NOT gadgets were used to flip the input for the gadgets. Figure 12a shows the four possible input seeds to build the assembly. A half-adder is simple enough to know which seed was used if 00 or 10 are output, but if 01 is output there is no way to know.


seed input is highlighted and all 4 possible seeds are shown in (a). Regardless of the seed, the final assembly will look identical except the final T/F representing the bits of the numbers added. This implements the schematic shown in Figure 11c and the XOR is implemented with NANDs as shown in Figure 8e.

6 Conclusions and Future Work

We have introduced the concept of covert computation in self-assembly and provided a general scheme to implement any boolean circuit under this restriction. Beyond potential applications to biomedical privacy, cryptography, and intellectual property, our techniques and framework promise to impact self-assembly theory itself. As a first example we have applied our techniques to the fundamental problem of Unique Assembly Verification in the negative glue aTAM, and shown it to be coNP-complete with growth-only systems, essentially as a corollary of our covert computation theory.

A number of future directions stem from our work. Having established the general computation power of covert computation, a natural next step is the consideration of efficiency for computing classes of functions. The *time* complexity of self-assembly computation has been studied [2, 16] and shown to allow for a substantial amount of parallelism. Can similar results be achieved under the covert constraint? What general connections exists between the time complexity for unrestricted self-assembly computation versus that of covert computation? Other natural metrics include minimizing the number of distinct tile types, along with the space taken up by the final assembly of the computation.

7 Acknowledgements

We would like to thank the anonymous reviewers for their careful review of our work and for their constructive feedback.

References

1 Leonard M. Adleman, Qi Cheng, Ashish Goel, Ming-Deh A. Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.

2 Yuriy Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Comp. Sci.*, 378:17–31, 2007.

3 Cameron Chalk, Erik D. Demiane, Martin L. Demaine, Eric Martinez, Robert Schweller, Luis Vega, and Tim Wylie. Universal shape replicators via self-assembly with attractive and repulsive forces. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, 2017.

4 Cameron Chalk, Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of any shape with constant tile types using high temperature. In *Proc. of the 26th Annual European Symposium on Algorithms*, ESA'18, 2018.

5 David Chaum, Claude Crépeau, and Ivan Bjerre Damgård. Multiparty unconditionally secure protocols (abstract). In *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 11–19, 1988.

6 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

7 Peter Claes, Denise K. Liberton, and Katleen et al. Daniels. Modeling 3d facial shape from dna. *PLOS Genetics*, 10(3):1–14, 03 2014. doi:10.1371/journal.pgen.1004224.

8 Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In *Proc. of the 12th ACM Workshop on Privacy in the Electronic Society*, WPES'13, pages 107–118. ACM, 2013.

9 David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.

10 David Doty, Lila Kari, and Benoît Masson. Negative interactions in irreversible self-assembly. *Algorithmica*, 66(1):153–172, 2013.

11 David Doty, Jack H. Lutz, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *Proc. of the 53rd IEEE Conf. on Foun. of Comp. Sci.*, FOCS '12, 2012.

12 N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Manual for using homomorphic encryption for bioinformatics. *Proceedings of the IEEE*, 105(3):552–567, March 2017.

13 Constantine Evans. *Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly*. PhD thesis, California Inst. of Tech., 2014.

14 Melissa Gymrek, Amy L. McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, 2013.

15 Z. Huang, E. Ayday, J. Fellay, J. Hubaux, and A. Juels. Genoguard: Protecting genomic data against brute-force attacks. In *2015 IEEE Symposium on Security and Privacy*, pages 447–462, May 2015. doi:10.1109/SP.2015.34.

16 Alexandra Keenan, Robert Schweller, Michael Sherman, and Xingsi Zhong. Fast arithmetic in algorithmic self-assembly. *Natural Computing*, 15(1):115–128, Mar 2016.

17 Eric S. Lander, Lauren M. Linton, and Bruce Birren et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2 2001.

18 Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, Aug 2018. doi:10.1007/s11047-018-9707-9.

19 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, Jun 2014.

20 Matthew J. Patitz, Trent A. Rogers, Robert Schweller, Scott M. Summers, and Andrew Winslow. Resiliency to multiple nucleation in temperature-1 self-assembly. In *Proc. of DNA Computing and Molecular Programming*, DNA'16, pages 98–113, 2016.

21 Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and turing universality at temperature 1 with a single negative glue. In *DNA Comp. and Molecular Prog.*, volume 6937 of *LNCS*, pages 175–189. 2011.

22 John H. Reif, Sudheer Sahu, and Peng Yin. Complexity of graph self-assembly in accretive systems and self-destructible systems. *Theoretical Comp. Sci.*, 412(17):1592–1605, 2011.

23 Robert Schweller and Michael Sherman. Fuel efficient computation in passive self-assembly. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'13, pages 1513–1525. SIAM, 2013.

24 Allan Scott, Ulrike Stege, and Iris van Rooij. Minesweeper may not be np-complete but is hard nonetheless. *The Mathematical Intelligencer*, 33(4):5–17, Dec 2011.

25 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag, Berlin, Heidelberg, 1999.

26 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

27 Jing Yang, Jingjing Ma, Shi Liu, and Cheng Zhang. A molecular cryptography model based on structures of dna self-assembly. *Chinese Science Bulletin*, 59(11):1192–1198, Apr 2014. doi:10.1007/s11434-014-0170-4.

8 Formal Definitions

Tiles. Let Π be an alphabet of symbols called the *glue types*. A tile is a finite edge polygon with some finite subset of border points each assigned a glue type from Π. Each glue type $g \in \Pi$ also has some integer strength $str(g)$. Here, we consider unit square tiles of the same orientation with at most one glue type per face, and the *location* to be the center of the tile located at integer coordinates.

Assemblies. An assembly is a finite set of tiles whose interiors do not overlap. If each tile in A is a translation of some tile in a set of tiles T, we say that A is an assembly over tile set T. For a given assembly Υ, define the *bond graph* G_Υ to be the weighted graph in which each element of Υ is a vertex, and the weight of an edge between two tiles is the strength of the overlapping matching glue points between the two tiles. Only overlapping glues of the same type contribute a non-zero weight, whereas overlapping, non-equal glues contribute zero weight to the bond graph. The property that only equal glue types interact with each other is referred to as the *diagonal glue function* property and is perhaps more feasible than more general glue functions for experimental implementation (see [6] for the theoretical impact of relaxing this constraint). An assembly Υ is said to be *τ-stable* for an integer τ if the min-cut of G_Υ is at least τ.

Tile Attachment. Given a tile t, an integer τ, and an assembly A, we say that t may attach to A at temperature τ to form A^0 if there exists a translation t^0 of t such that $A^0 = A^{S}\{t^0\}$, and the sum of newly bonded glues between t^0 and A meets or exceeds τ. For a tile set T we use notation $A \rightarrow_{T,\tau} A^0$ to denote there exists some $t \in T$ that may attach to A to form A^0 at temperature τ. When T and τ are implied, we simply say $A \rightarrow A^0$. Further, we say that $A \rightarrow^* A^0$ if either $A = A^0$, or there exists a finite sequence of assemblies $hA_1 \ldots A_k i$ such that $A \rightarrow A_1 \rightarrow \ldots \rightarrow A_k \rightarrow A^0$.

Tile Systems. A tile system $\Gamma = (T,S,\tau)$ is an ordered triplet consisting of a set of tiles T called the system's *tile set*, a τ-stable assembly S called the system's *seed* assembly, and a positive integer τ referred to as the system's *temperature*. A tile system $\Gamma = (T,S,\tau)$ has an associated set of *producible* assemblies, PROD_Γ, which define what assemblies can grow from the initial seed S by any sequence of temperature τ tile attachments from T. Formally, $S \in \text{PROD}_\Gamma$ is a base case producible assembly. Further, for every $A \in \text{PROD}_\Gamma$, if $A \rightarrow_{T,\tau} A^0$, then $A^0 \in \text{PROD}_\Gamma$. That is, assembly S is producible, and for every producible assembly A, if A can grow into A^0, then A^0 is also producible. We further denote a producible assembly A to be *terminal* if A has no attachable tile from T at temperature τ. We say a system $\Gamma = (T,S,\tau)$ *uniquely produces* an assembly A if all producible assemblies can grow into A through some sequence of tile attachments. More formally, Γ *uniquely produces* an assembly $A \in \text{PROD}_\Gamma$ if for every $A^0 \in \text{PROD}_\Gamma$ it is the case that $A^0 \rightarrow^* A$. Systems that uniquely produce one assembly are said to be *deterministic*.

Finally, we consider a system to be a *valid growth-only* system if all assemblies in PROD_Γ are τ-stable. The existence of negative strength glues allows for the possibility that unstable assemblies are produced.

9 Gadget Properties

Here, we describe some of the gadgets in more detail by specifying the properties that each gadget must have in order to guarantee covert operation.

9.1 Logic Diodes

Logic Diodes are important to prevent timing issues in the dual rail logic related to backfilling of the covert gadgets. They must have these properties.

1. If t^{in}_i enters, then only t^{out}_i leaves. This is guaranteed due to h_2,h_3 cooperatively placing the next tile. Without f_i^{in} present, the only next tile which could be placed is this cooperatively-placed tile from t^{out}_i.

2. If f_i^{in} enters, then only f_i^{out} leaves. This is guaranteed due to h_4,h_5 cooperatively placing the next tile. Without t^{in}_i present, the only next tile which could be placed is this cooperatively-placed tile from f_i^{out}.

3. The f_i^{in} wire will be backfilled if and only if t^{in}_i enters. Without t^{in}_i present, a given backfilled false path will stop at the tile with h_4,h_5. With t^{in}_i present, the tile with h_1,h_4 can cooperatively attach and backfill the false wire.

4. The t^{in}_i wire will be backfilled if and only if f_i^{in} enters. Without f_i^{in} present, a backfilled false path will stop at the tile with h_2,h_3. However, with f_i^{in} present, the h_1,h_2 tile can cooperatively attach and backfill the false wire.

9.2 NOT Gadget

In verifying that the gadget works as intended, we must verify six properties.

1. If t^{in}_i enters a NOT gadget, it results in f_i^{out} and not t^{out}_i. Figure 5a shows the gadget in this case– with true input t^{in}_i and output f_i^{out}. The true output can not place from tile x due to the negative glues n_1 and n_3 of strength −1. Given the build order, we are guaranteed f_i^{out} and that t^{out}_i can not build.

2. If f_i^{in} enters a NOT gadget, it results in t^{out}_i and not f_i^{out}. Figure 5c shows the gadget in this case– with false input f_i^{in} and output t^{out}_i. The false output can not attach due to the negative glue n_2. The tile to the west of x may attach, but due to glue n_3, no other tile can attach. Given the build order, we are guaranteed t^{out}_i and that f_i^{out} can not build.

3. If t^{in}_i enters a NOT gadget and f_i^{out} leaves, the gadget and f_i^{in} is backfilled up to tile x. Figure 5b shows the desired result. Glues h_3,h_4 cooperatively allow the false portion and wire to backfill. Glue h_2 is needed to fill in the tile with n_2.

4. If f_i^{in} enters a NOT gadget and t^{out}_i leaves, the gadget and t^{in}_i is backfilled up to tile x. Figure 5d shows the desired result. Glues h_5,h_6 allow the true portion and wire to backfill. Glue h_1 is needed to counteract the n_1 glue when backfilling that tile.

5. If the gadget resulted in f_i^{out}, a future gadget can backfill t^{out}_i and the gadget will be complete. If the gadget is in the configuration of Figure 5b, the true wire can directly backfill until the tile directly above tile x. The glue n_1 would prevent this tile from placing except x will be there and the tile can cooperatively be placed using the north glue of x and the south glue of the backfilling wire.

6. If the gadget resulted in t^{out}_i, a future gadget can backfill f_i^{out} and the gadget will be complete. If the gadget is in the configuration of Figure 5d, the false wire can directly backfill until the tile directly east of tile x. The glue n_2 would prevent this tile from placing except x is there and the tile can cooperatively be placed using the east glue of x and the west glue of the backfilling wire.

The first two conditions guarantee that the gadget works correctly. The second two conditions guarantee the covertness of the gadget, i.e., the gadget looks indistinguishable before the output regardless of the input. The final two conditions verify that the backfill from future gadgets will work correctly, and no trace of the build path will be evident.

9.3 NAND Gadget

There are many specific issues related to the NAND gadget handled in the text. The properties that it must have are as follows.

1. Given f_i^{in} or f_j^{in}, the wire t^{out}_{ij} leaves the gadget. If wire $f_i^{in} = t_i^{out}$ is set, this is tied directly to the output wire t^{out}_{ij}. If the wire f_j^{in} comes in, it comes in as the false input of the second H-NOT gadget, which means it leaves as t^{out}_{ij} by the validity of the NOT gadget.

2. If wires t^{in}_i and t^{in}_j are set, then the wire f_{ij}^{out} should leave the gadget. Wire t^{in}_i exits the first NOT gadget as f_i^{out}. This wire, f_i^{out}, and t^{in}_j both stop and expose glues a_2 and a_3, respectively. Both are strength 1 glues, and thus the tile with glues a_2 and a_3 can only place if both the glues are exposed. Thus, only if wires t^{in}_i and t^{in}_j are set, will wire t^{in}_{ij} ever enter the H-NOT gadget, which results in the wire f_{ij}^{out} as the gadget output.

3. All components backfill correctly. The NOT gadget backfills correctly. If f_j^{in} was set, then t^{in}_{ij} is backfilled from the H-NOT gate. When the tiles are placed such that glues a_1 and a_2 are placed, the tile can cooperatively place that backfills f_i^{out}. Similarly, when the other tiles place with glues a_3 and a_4, a tile cooperatively attaches to backfill the t^{in}_j.

4. The growth-only constraint is not violated with the negative glues. This can only happen given a stable assembly where a tile attaches with a negative glue that destabilizes part of the assembly. The additional negative glue n_H could do this if the green tile is placed after the blue tile, however, the build path is intentional to ensure this can not happen. If the wire f_{ij}^{out} were placed and t^{out}_{ij} is backfilled, the tile with n_H would be the last tile that could attach and the assembly would never be unstable.

5. The gadget does not behave incorrectly with only one input. The logic diode guarantees the backfilling never goes beyond the gadget. If one input is false, the NAND can send the t^{out}_{ij} wire and backfill the NAND gadget without having yet receieved the second input. When it arrives, that wire is backfilled.

9.4 FANOUT Gadget

The FANOUT gadget needs to duplicate the geometric wire, and also needs to only backfill once both outgoing wires have backfilled. Figure 7a shows the FANOUT gadget. It has the following necessary properties.

1. With input t^{in}_i, the gadget outputs wires t^{out}_{i1} and t^{out}_{i2}, and does not output $f_i^{out_1}$ and $f_i^{out_2}$. Figure 7b shows the true fanout without the backfilling. Due to n_1 and n_2, the false outputs can not assemble. Both settings share the same middle four tiles, but with placement order n_1 is placed first and then cooperative glues are used to place the first tile of the four (with glues g_3,g_4).

2. With input f_i^{in}, the gadget outputs wires $f_i^{out_1}$ and $f_i^{out_2}$, and does not output t^{out}_{i1} and $t_i^{out_2}$. Figure 7c shows the false fanout without the backfilling. Due to n_1 and n_2, the true outputs can not assemble. With placement order n_2 is placed first and then cooperative glues are used to place the first of the four middle tiles (with glues h_3,h_4). **3.** With input wire t^{in}_i, wire f_i^{in} only backfills once $f_i^{out_1}$ or $f_i^{out_2}$ have backfilled. Both wires backfill independently, and only when

$f_i^{out_2}$ is backfilled will wire f_i^{in} backfill. **4.** With input wire f_i^{in}, wire t^{in}_i only backfills once $t^{out}_{i\,1}$ or $t^{out}_{i\,2}$ have backfilled. Both wires backfill independently, and only when $t_i^{out_1}$ is backfilled will wire t^{in}_i backfill.