Attacks on Covert Computation in the Abstract Tile Assembly Model

Emma Fu, Timothy Gomez, Vihan Lakshman

May 13, 2026

Abstract

We study attacks (and potential defenses) of *covert computation* in the Abstract Tile Assembly Model (aTAM). In a traditional aTAM computation, which proceeds by sequentially gluing tiles together according to a set of rules, both the input, the output, and the entire history of the computation are revealed. This property is of course concerning for users looking to cryptographically hide the inputs of their computation from untrusted observers. Motivated by this challenge, a line of research developed the notion of covert computation for aTAM, providing a method of constructing tile assembly circuits such that the final assembly reveals no information about the input other than what can be learned from the output value. We make the observation that covert computation, despite its influence in the aTAM literature, harbors a potentially significant shortcoming: the security guarantee only holds for observers viewing the final tile assembly, and does not protect against attackers who view snapshots of the computation in process. In this paper, we formalize this observation by presenting attacks on covert computation in settings where adversaries may glean information about the tile assembly process while in flight. Along the way, we also introduce a new formal definition of *probabilistic leakage functions* which may be of independent interest.

1 Introduction

One of the next frontiers of computing is using biochemical molecules as physical hardware in place of traditional silicon-based architectures to take advantage of the energy efficiency and inherent parallelism of these materials. DNA computing [7] has emerged as a particularly promising candidate for molecular computation, beginning with Leonard Adleman's famous 1994 demonstration solving a small instance of the Hamiltonian path problem with a DNA computer [1]. On the theoretical side, molecular computation is often modeled via *tile assembly* processes that define computation as a sequence of gluing geometric tiles together. A particularly influential instantiation of tile assembly is the Abstract Tile Assembly Model (aTAM) defined in [8] and shown to be Turing-complete (and thus capable of executing any

computable function). However, naive instantiations of aTAM reveal not just the output, but the input and the entire computation history to any observer who views the final assembly state. This property implies that the traditional computation methods in the aTAM are not suitable for cryptographic applications.

To overcome this limitation, Cantu, et al. [4] proposed the idea of *covert computation* which transforms a Boolean function into an equivalent tile assembly system such that the final assembly does not reveal any additional information about the input beyond what can be inferred from the output value[1]. This was also used to show that the Unique Assembly Verification (UAV) problem is coNP-hard. This led to a relationship between covert constructions and algorithmic hardness. A covert circuit was shown in [3] to solve the complexity of UAV in the *staged assembly model* with constant stages. The construction in [4] utilized negative glues, which means some tiles "repel" each other and can negatively contribute to attachment. Without this power the model is conjectured to be not capable of covert. In [2] if we allow for 3D tiles, or allow our assembly to be exponentially sized, then we can simulate any Boolean circuit. In the same paper a relationship between functions computable in the aTAM and P/poly is shown.

However, despite the influence of covert computation in the aTAM literature, we make the observation that the covert security guarantee only holds against adversaries who view the final assembly state: it does not protect against attackers who may view partial assemblies while the computation is in progress. To our knowledge, this potential vulnerability of covert computation has not been studied in the literature. In this project, we ask the following question: *Can we design attacks against covert computation that reveal information about the input given partial access to the tile assembly state while in the computation is in progress*?

In summary, we make the following contributions in this project:

- We provide a randomized leakage model motivated by covert computation based on revealing wires of a boolean circuit.
- We study how many samples are needed to reconstruct the input with non-negligible probability. We give two attacks, one based on the coupon collector, and another based on a generalization where we grab multiple coupons at the same time.
- We finish by analyzing a simple example of a covert circuit which mimics the dynamics of the construction [4]. We show that there are two natural leakage functions which can reveal the input with non-negligible probability. We briefly cover ways to defend each one.

[1]The power of the covert computation guarantee is conditional on the function f being computed. If f is efficiently invertible, then the output trivially reveals the input and covert computation provides no value. On the other hand, if f is a one-way function, then covert computation provides a meaningful notion of security.

2 Background

2.1 aTAM

Let G be a set of glues. A tile is a unit square with a tuple of glues (n, e, w, s), one for each side. A glue function $g : G^2 \to \mathbb{Z}$.

Definition 1 (Assembly)**.** *An assembly is a set of non-overlapping tiles. An assembly is τ-stable if all cuts have at least weight τ.*

Growth starts from a seed assembly usually denoted as σ. Growth begins from the seed by attaching a tile if the matching glues are at least τ.

Definition 2 (Assembly Sequence)**.** *An assembly sequence is a sequence of tile placements $\alpha = (\sigma, A_1, A_2, \dots)$ starting from the seed assembly σ. Where each A_i is τ-stable and each A_i and A_{i+1} only differ by one tile.*

The set of assemblies which can be built by some assembly sequence starting from a seed assembly σ, denoted as $PROD(\mathcal{T}, \sigma)$ is called the *producible assemblies*. The subset of the producible assemblies which can no longer grow are called the *terminal assemblies*. We can also think of the set of producible assemblies after k steps denoted by $PROD_t(\mathcal{T}, \sigma)$, these are assemblies produced using assembly sequences of length k.

Definition 3 (Tile Assembly System)**.** *A tile assembly system (TAS) is a tuple $\mathcal{T} = (T, \tau)$ where T is a finite set of tile types, and $\tau \in \mathbb{N}$ is the temperature.*

Definition 4 (Directed TAS)**.** *A TAS is directed if it produces a unique terminal assembly A.*

2.2 Tile Automata

Tile automata is a generalization of the aTAM where the main addition is that tiles can change states/tile type once they attach. This is done similar to Asynchronous Cellular Automata. Rather than define our attachments through glues, we use an affinity function Π which takes two states and a direction $d \in \{H, V\}$ and outputs their attachment strength in the either vertical or horizontal direction. We also have a transition function which is a partial function Δ which takes two states and a direction, and outputs two new states.

Definition 5 (Tile Automata system)**.** *A Tile Automata system is a tuple $\Gamma = (\Sigma, \Pi, \Delta, \tau)$ where Σ is a set of tile states, Π is an affinity function, Δ is a transition function, and τ is the temperature.*

Now our producible assemblies and assembly sequences are defined by what we can attach, and what tile transitions we can apply. Terminal assemblies are defined as assemblies which don't have any possible attachments or transitions.

2.3 Covert Computation

Here we formally define Tile Assembly Computers and the requirements for Covert Computation.

Definition 6 (Tile Assembly Computer)**.** *A* tile assembly computer (TAC) *for a function* $(f : \mathcal{D} \to \mathcal{R})$ *is a tuple*

$$(\mathcal{T}, I, O)$$

where:

- $\mathcal{T}$ *is an aTAM system*
- $I : \mathcal{D} \to \mathcal{A}$ *is an input function that maps each input* $x \in \mathcal{D}$ *to a seed assembly* $I(x)$*,*
- $O : \mathcal{A} \to \mathcal{R}$ *is an output function that maps assemblies to outputs in the range of* f*.*[2]

The TAC is valid *for* f *if for every input* $x \in \mathcal{D}$*, the tile assembly system*

$$(T, I(x), \tau)$$

produces a unique terminal assembly A_x*, and*

$$O(A_x) = f(x).$$

Similarly define this for Tile Automata systems but with Γ instead of $\mathcal{T}$. We now provide a definition for covert. We continue by re-framing covert as a cryptographic leakage problem.

Definition 7 (Covert Computation)**.** *A TAC covertly computes a function* $f(x)$ *if for every pair of inputs* x, y *such that* $f(x) = f(y)$*,* $A_x = A_y$*. Informally a TAC is covert if for each output there is exactly one assembly representing that output.*

2.4 Leakage Resilient Cryptography

Our approach to attacking covert computation is inspired by the cryptographic literature on *leakage-resilient functions* [6]. Traditional cryptographic models assume secret state is perfectly hidden from the adversary; leakage-resilient cryptography weakens this assumption to model side channels — cache attacks, timing attacks, power analysis — that reveal partial information about secrets through observable side effects of computation. Formal definitions typically involve a *leakage oracle* the adversary can query. Some cryptographic models (e.g., noisy leakage [5]) treat the oracle's output as probabilistic, capturing measurement noise on an otherwise deterministic computation. In aTAM, however, the computation *itself* is randomized: re-running the assembly on the same input produces different attachment orders and different partial assemblies. The natural leakage model in this setting treats the oracle's output as a sample from a substrate-induced distribution, with randomness arising from the computation rather than from added noise. We formalize this below.

[2]Usually I and O are very easy-to-compute functions based on placing/reading specific bits at fixed locations

Definition 8 (Probabilistic Leakage Oracle)**.** *Let T be a tile assembly construction (TAC) computing a function $f : \mathcal{X} \to \mathcal{Y}$, let A_x denote the random variable describing the assembly process induced by T on input $x \in \mathcal{X}$, and fix a timestep t. A* probabilistic leakage oracle *L_T^t is a randomized procedure that on each call:*

1. *samples a fresh run of the assembly process on x, producing a partial assembly $A_x^{(t)}$ drawn from the distribution induced by the aTAM's intrinsic nondeterminism;*
2. *flips fresh internal coins to select and return an observation (e.g., the value of a uniformly random wire structurally exposed in $A_x^{(t)}$).*

Definition 9 (Probabilistic Leakage Resilience)**.** *Let T be a TAC computing $f : \{0,1\}^n \to \mathcal{Y}$, and let L be a probabilistic leakage oracle for T. The TAC T is* L-leakage-resilient *if for every PPT algorithm P making* $\mathrm{poly}(n)$ *adaptive queries to L,*

$$\Pr_{x \leftarrow \{0,1\}^n} \left[P^L(f(x)) = x \right] \;\leq\; \mathsf{negl}(n),$$

where each oracle call returns an independent sample as in Definition 8.

Remark 1. *A TAC is* covert *if it is leakage-resilient against an oracle that reveals only the terminal assembly.*

3 Circuit Leakage Attack

Using our model of probabilistic leakage, we can define an attacker who can rerun an aTAM computation many times (e.g. run many DNA test tubes executing the computation in parallel)

This can be modelled as an attacker has access to a probabilistic leakage oracle. Let the attack be as follows. Fix a timestep t, and rerun the aTAM computation k times up to timestep t. Query the leakage oracle for the value of the bit on a uniformly sampled wire returned by the oracle.

Given this, we want to find out how many samples we need to reveal the full computation.

Theorem 1. *For a circuit S and history w, given $O(|S| \log |S|)$ samples of wires ℓ, one can construct the input with high probability.*

Proof. In order to construct the input with high probability, it is more than sufficient to see the whole computation history. We can bound the probability to see every single input bit via the Coupon Collector Problem, which states we need $O(|w| \log |w|)$ to see all the wires. Because of the Coupon Collector Problem, we know that if we sample $O(|S| \log |S|)$, we are expected to get one of each "coupon", so we are expected to see every single input bit.

The number of wires is $|S| + n$, however, in most cases we need at least 1 gate for each input bit, which gives us $n \leq O(|S|)$ so $|w| = O(|S|)$. □

This is an attack where we fix a timestep t, and rerun the aTAM computation k times up to timestep t, each time getting exactly one wire per query. Using the coupon collector method, we can say that we can determine all of the inputs with high probability.

However, consider the model where we can get multiple wires per query. Then, the k wires we select per query are selected "without replacement", and are replaced after each query.

Theorem 2. *For a circuit S and history w, given $O(\log\log|S|)$ samples each of size $\alpha|S|$, with $0 < \alpha < 1$, one can construct the input with non-negligible probability.*

Proof. First, we find the expected number of inputs we see within h samples. Let n be the number of non-input wires, and m be the number of input wires, such that $n + m = |S|$. We can calculate the expected number of input wires we see by calculating the probability that a wire has been selected sometime within the h samples. The probability that a given wire is not in a sample is $\binom{n+m-1}{k}/\binom{n+m}{k} = \frac{n+m-k}{n+m}$. Then, since we draw each sample with replacement, the probability that this input wire gets selected in none of our h samples is $(\frac{n+m-k}{n+m})^h$, which means that the probability that the input wire is selected at least once within our h samples is $1 - (\frac{n+m-k}{n+m})^h$.

Let X_i be an indicator variable describing whether we have seen input wire i in the h samples. Then, $E[X_i] = 1 - (\frac{n+m-k}{n+m})^h$. By linearity of expectation, we get that $E[X]$, which is the expected number of input wires we see after h samples, is $m(1 - (\frac{n+m-k}{n+m})^h)$.

Then, if each sample is of size $\alpha|S|$, we can bound the expected number of input wires needed to see a sufficient number of input wires. We can afford to not see $\log m$ wires, as we can guess this quantity in a reasonable amount of time. Then, we can bound the expected number of wires as follows, in order to find how many samples of size $\alpha|S|$ we need to obtain:

$$
\begin{aligned}
m(1 - (\frac{n+m-k}{n+m})^h) &\geq m - \log m \\
m(1 - (\frac{|S| - \alpha|S|}{|S|})^h) &\geq m - \log m \\
m(1 - (1-\alpha)^h) &\geq m - \log m \\
1 - (1-\alpha)^h &\geq 1 - \frac{\log m}{m} \\
(1-\alpha)^h &\leq \frac{\log m}{m} \\
h(1-\alpha) &\leq \log(\frac{\log m}{m}) \\
h &\leq \frac{\log(\frac{\log m}{m})}{1-\alpha} \\
h &= O(\log\log m)
\end{aligned}
$$

□

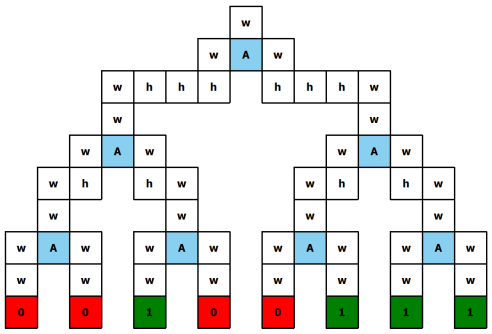
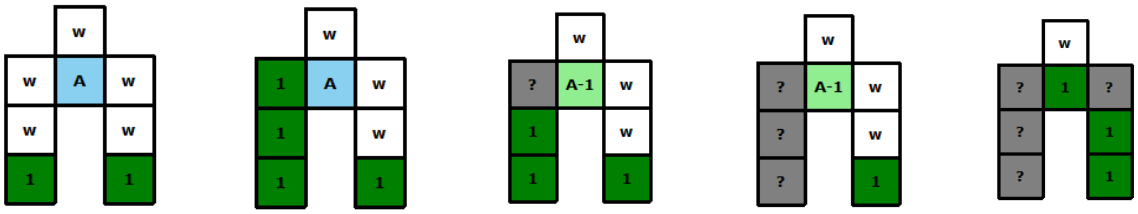


Figure 1: Seed Assembly



(a) Input (b) First signal (c) Read Signal (d) Hidden input (e) Second input

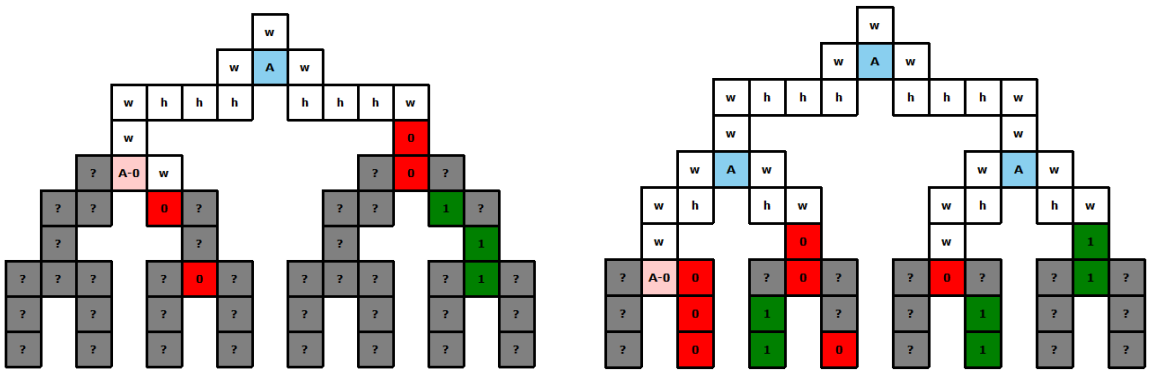
Figure 2: Walk through of how wires and an AND tile interact.

4 Covert Computation Attacks

Now we will cover the ways that standard covert constructions might reveal information. We start with a simple covert circuit in the Tile Automata model. This process somewhat mimics the behavior of the circuit in [4] where wires grow out toward gates, then once the signal reaches a gate back fills toward the previous gate. Next we will provide 2 methods to attack covert, one based on sampling at a specific time step, another by viewing the first tile attachment / transition rule. These proofs are written for the Tile Automata construction we provide, but these theorems carry over to the circuit from [4].

4.1 Covert Tile Automata Circuit Rules

As an example we give a high level covert construction in Tile Automata. For ease of simulation we start with the seed assembly already in the shape of the circuit as in Figure 1. We show how an AND gate works in Figure 2. The wire states are $w, 0, 1, ?$ indicating empty, false, true, and hidden. States for 0 and 1 turn w into the same state. Once a gate state A, is reached it reads the left bit to go to state $A-0$ or $A-1$ and hides the left input wire tile. When the other signal arrives, the previous states turn into 0 or 1 based on its inputs and hides the right input wire tile. This process is repeated. In the larger construction we also have turn states h to wire up everything. When h reads a tile, it also hides it.



(a) (b)

Figure 3: Two randomly selected assemblies, the one on the right is taken at an earlier time step than the left, so less wires are computed but less are hidden.

4.2 Partial Assembly Leakage

Each partial assembly reveals at least one bit. This idea is seen in partial assemblies in Figure 3. If we can sample assemblies right before the assembly process finishes, at time step $t-1$, then we can simulate the coupon collector attack. This is motivated by the fact that in practice, tile assemblies do not always finish growing by the time the chemist checks the test tube, these partial assemblies might reveal information.

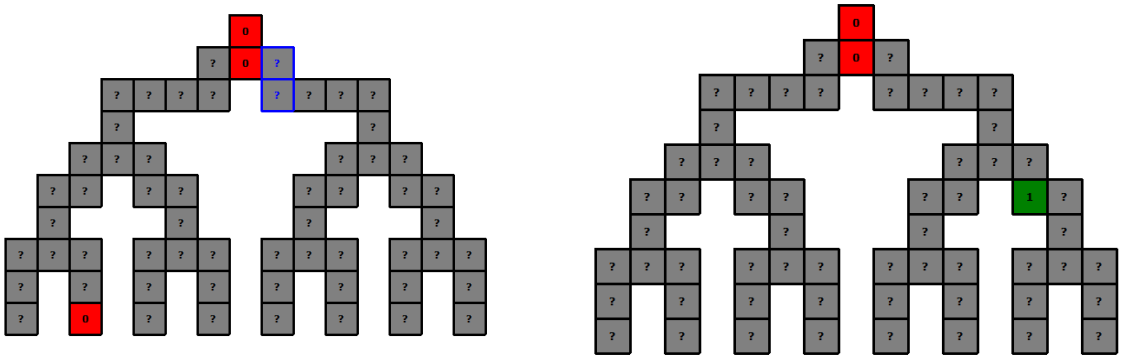
Lemma 1. *There is a one-to-one mapping between wires and* $PROD(T, t-1)$.

Proof. On each wire there is some tile gets hidden last from the other tiles on the wire. Each of these last tiles gets hidden independently, so any of the last wires could be the last to chosen to be hidden. Each assembly in $PROD(\mathcal{T}, t-1)$ reveals some wire as shown in Figure 4 and thus maps to that wire. For any assembly sequence leading to A_X we can move the last transition that hides some wire w to the end of the sequence. This lets us map any wire back to the same assembly. □

With this Lemma, sampling assemblies simulates sampling wires, so we can simulate the coupon collector attack and get an upper bound

Theorem 3. *When L reveals one assembly from $PROD(\mathcal{T}, t-1)$ uniformly at random, the covert circuit presented is not L-resilient even against $O(|S| \log |S|)$ samples for circuit size $|S|$.*

Proof. Since there is a mapping between wires and assemblies in $PROD(\mathcal{T}, t-1)$, our sampled assemblies maps to some bit that it reveals. Thus using $O(|S| \log |S|)$ assemblies, we also get the same number of bits and we can simulate the attack from 1 and reveal the input with high probability. □



(a) Revealing Input (b) Revealing Wire

Figure 4: Two assemblies sampled from $PROD(\mathcal{T}, t-1)$. Both have exactly one wire not yet hidden.

Possible Defense: We can synchronize the hiding of the inputs so that the growth doesn't end until the inputs are all hidden. This makes it so that no input wire is revealed in $PROD(\mathcal{T}, t-1)$. This doesn't defend against attacking earlier time steps $t-k$ for some k, but if we know when the adversary is sampling we can add additional synchronizing tiles to push the hiding of the input deeper into the assembly process, although this increases the amount of time it takes to build and increases the assembly size.

4.3 Build Sequence Leakage

The previous attack focuses on viewing the end of the assembly process, we can actually attack covert if we learn the first tile attachment / first transition rule applied. This is motivated by the fact that attachments and transition rules in practice leave some "junk" molecule that if revealed might reveal information.

Lemma 2. *When L reveals the first attachment or transition of some assembly sequence at uniformly at random*[3]*, the covert circuit presented is not resilient even against $O(n \log n)$ samples over functions on n-bits.*

Proof. This result does not reduce to the coupon collector attack, but mirrors the proof. The first transition changes a w state to 0 or 1. By sampling the rule used and the position, we learn the value of some uniformly random input wire since these are the only transitions that can occur. Each sample gives us a random input bit, so after $O(n \log n)$ samples we can learn the input with high probability. □

Defenses We can add additional dummy tiles that do not compute or read any bits. This adds noise to the samples of the first attachment and lowers the chance of finding the input bits.

[3]Uniformly random over attachments and rules.

5 Conclusion & Future Work

In this project, we studied attacks on the covert computation model in the Abstract Tile Assembly Model (aTAM). While covert computation provides guarantees that the final assembly state does not provide any additional information about the input beyond what can be gleaned from the output value, we make the observation that this does not preclude attacks that can observe partial information as the computation proceeds. In particular, through a new definition of probabilistic leakage oracles tailored to the aTAM model, we demonstrate that an attacker capable of observing partial tile assemblies can learn non-negligible information about the input even in the covert setting. We also discuss a possible defense to our covert attack, but note that our defense only partially mitigates the threat and more work is needed in this setting. We close by highlighting a few directions for future work:

- Is there a time step we can sample to simulate the handfuls attack?
- What if we revealed assemblies weighted by the likelihood they appear instead of uniformly. In this case we cannot simulate the coupon collector attack.
- What other ways can we defend against these attacks?
- Are there other settings where our randomized leakage model can be useful?
- Can we show a lower bound on number of samples to break covert leakage resilience for any leakage function?

References

[1] Adleman, L. M. Molecular computation of solutions to combinatorial problems. *science 266*, 5187 (1994), 1021–1024.

[2] Alaniz, R. M., Gomez, T., Rodriguez, A., Wylie, T., Caballero, D., Grizzell, E., and Schweller, R. Covert computation in the abstract tile-assembly model.

[3] Caballero, D., Gomez, T., Schweller, R., and Wylie, T. Covert computation in staged self-assembly: Verification is pspace-complete. In *European Symposium on Algorithms* (2021).

[4] Cantu, A. A., Luchsinger, A., Schweller, R., and Wylie, T. Covert computation in self-assembled circuits. *Algorithmica 83*, 2 (2021), 531–552.

[5] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference* (1999), Springer, pp. 398–412.

[6] Kalai, Y. T., and Reyzin, L. A survey of leakage-resilient cryptography. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 727–794.

[7] Paun, G., Rozenberg, G., and Salomaa, A. *DNA computing: new computing paradigms*. Springer Science & Business Media, 2005.

[8] Winfree, E., Liu, F., Wenzler, L. A., and Seeman, N. C. Design and self-assembly of two-dimensional dna crystals. *Nature 394*, 6693 (1998), 539–544.