Verifying Differential Privacy in TLA$^+$ via Self-Products

Uğur Y. Yavuz

Boston University

TLA$^+$ Community Event
April 12, 2026

The privacy problem

Suppose a hospital wants to release statistics about its patients, such as average age and disease prevalence.

The challenge: the released statistics should be useful for research, but should not reveal whether any specific individual was a patient.

Naive approaches fail:

- Removing names is not enough: auxiliary information can re-identify individuals.
- Releasing exact aggregates leaks information through differencing attacks.

Differential privacy (Dwork et al., 2006)
provides a rigorous mathematical guarantee against this.

What is differential privacy?

A randomized algorithm M takes an input d (e.g., a database) and produces a randomized output. We say M is ϵ-**differentially private** w.r.t. an adjacency relation Φ on inputs if for all d, d' with $\Phi(d, d')$ and all output sets A:

$$\Pr[M(d) \in A] \;\leq\; e^{\epsilon} \cdot \Pr[M(d') \in A]$$

The output distribution changes by at most a factor of e^{ϵ} when the input changes in a way deemed "negligible" by Φ.

- Φ captures what "neighboring" inputs means: e.g., databases differing in one row.
- ϵ is the *privacy budget*: smaller ϵ means stronger privacy.
- The bound holds worst-case over all adjacent inputs and all output events.

DP is a probabilistic relational property

DP is a probabilistic relational property:
it relates the output distributions of two runs on adjacent inputs.

- In TLA$^+$, we reason about *trace properties*, i.e. properties of individual executions.
- *Hyperproperties* are properties of *sets* of executions, e.g. non-interference.
- Lamport & Schneider (2021) treat hyperproperties in TLA$^+$,
  but only **non-probabilistic** ones.
- As a **probabilistic hyperproperty**, DP falls outside existing TLA$^+$ approaches.

Building differentially private algorithms

A DP algorithm is typically a program consisting of a sequence of deterministic computation interleaved with calls to differentially private mechanisms, e.g. **the Laplace mechanism**.

The Laplace mechanism

$\mathrm{Lap}_\epsilon(t)$ releases $t +$ noise, where noise is drawn from a Laplace distribution with scale $1/\epsilon$.

If $|t - t'| \leq k$ whenever the program runs on adjacent inputs d, d', then releasing $\mathrm{Lap}_\epsilon(t)$ is $k\epsilon$-DP.

The pWhile language (Barthe et al., 2014)

A simple imperative language for reasoning about DP.

Grammar.

$$\begin{aligned}\mathcal{C} ::= \ & \texttt{skip} \mid \mathcal{C}; \mathcal{C} \mid x \leftarrow e \mid \texttt{if } e \texttt{ then } \mathcal{C} \texttt{ else } \mathcal{C} \mid \texttt{while } e \texttt{ do } \mathcal{C} \\ & \mid x \xleftarrow{\$} \mathrm{Lap}_{\epsilon}(e) \\ & \mid \texttt{return } e\end{aligned}$$

- Imperative core with Booleans and integers, plus probabilistic assn. from Laplace.
- We restrict to programs of the form c; `return` e.
- Programs denote functions from input memories to *distributions* over outputs.

The self-product construction (Barthe et al., 2014)

Barthe et al. proposed the **self-product** that replaces sampling with non-determinism. Every variable x is duplicated into x_1 and x_2, and **ghost variable** v_ϵ is added.

Selected transformation rules.

Source (c)	$\longrightarrow$	**Self-product ($\lceil c \rceil$)**
$x \leftarrow e$	$\longrightarrow$	$x_1 \leftarrow e_1;\ x_2 \leftarrow e_2$
$\texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ d$	$\longrightarrow$	$\texttt{assert}(b_1 = b_2);\ \texttt{if}\ b_1\ \texttt{then}\ \lceil c \rceil\ \texttt{else}\ \lceil d \rceil$
$\texttt{while}\ b\ \texttt{do}\ c$	$\longrightarrow$	$\texttt{while}\ b_1\ \texttt{do}\ (\texttt{assert}(b_1 = b_2);\ \lceil c \rceil); \texttt{assert}(b_1 = b_2)$
$x \xleftarrow{\$} \mathrm{Lap}_\epsilon(e)$	$\longrightarrow$	$(x_1, x_2) \leftarrow \mathrm{Lap}^\diamond_\epsilon(e_1, e_2)$

$\mathrm{Lap}^\diamond_\epsilon(e_1, e_2)$: non-det. pick a value v, set $x_1 = x_2 = v$, increment v_ϵ by $\epsilon \cdot |e_1 - e_2|$.

From self-product to DP

Main theorem (Barthe et al., 2014)

Let c; `return out` be a pWhile program. Let Φ be an adjacency relation on its inputs.

If the self-product satisfies the Hoare triple

$$\{\Phi(d_1, d_2) \wedge v_\epsilon = 0\} \ \lceil c;\ \texttt{return out} \rceil \ \{\texttt{out}_1 = \texttt{out}_2 \wedge v_\epsilon \leq \epsilon\}$$

where d_1, d_2 are the program inputs in the two simulated executions, then c; `return out` is ϵ-DP with respect to Φ.

- The probabilistic content of DP is hidden inside the main theorem proof, which the user does not have to redo.
- Verifying ϵ-DP reduces to a Hoare-logic proof of a bound on the ghost variable v_ϵ, and an equality of outputs, i.e. a standard safety property of the self-product.

Self-products in PlusCal

We adapt Barthe's construction to the TLA$^+$ ecosystem: the source language is **PlusCal**, and the self-product is generated automatically.

- The user writes a PlusCal algorithm with Lap_ϵ calls expressed as uninterpreted operators, e.g. `Lap(Epsilon, e)`, and returns as writing to a special variable `out`.
- A Python script (`transform.py`, built with TLA$^+$'s tree-sitter grammar) rewrites the source into its self-product.
- The result is then translated to TLA$^+$ via the standard PlusCal translator.

PlusCal already has a non-deterministic choice construct (`with`), which we use to model abstract Laplace calls in the self-product.

Self-products in PlusCal: transformation rules

The deterministic constructs translate as in Barthe's pWhile rules, expressed in PlusCal syntax.

Source	$\longrightarrow$	**Self-product**
`x := e`	$\longrightarrow$	`x1 := e1 || x2 := e2`
`if (b) {...} else {...}`	$\longrightarrow$	`await (b1 = b2); if (b1) {...} else {...}`
`while (b) {...}`	$\longrightarrow$	`while (b1) {await (b1 = b2); ...}; await (b1 = b2)`

Self-products in PlusCal: transformation rules

A Laplace call:

```
y := Lap(Epsilon, e);
```

becomes, in the self-product:

```
with (res \in AbsLap(Epsilon, e1, e2, v_eps)) {
  y1 := res[1] || y2 := res[2] || v_eps := res[3]
}
```

`AbsLap` is an uninterpreted operator from a reusable `DP` module, instantiated to a set of tuples encoding the Hoare specification of $\mathrm{Lap}_\epsilon^\diamond$:

$$\texttt{AbsLap}(\epsilon, e_1, e_2, v_\epsilon) = \{\langle n, n, v_\epsilon + \epsilon \cdot |e_1 - e_2| \rangle : n \in \mathit{Value}\}$$

The proof obligation

After translation, the self-product becomes a standard TLA$^+$ specification. The Hoare triple from Barthe et al.'s main theorem becomes an **invariant**:

$$pc = \texttt{"Done"} \;\Rightarrow\; (out_1 = out_2 \;\wedge\; v_\epsilon \leq \epsilon)$$

assuming $\Phi(d_1, d_2)$ at the initial state, where d is the input to the original program.

How it gets checked.

- **TLC.** The invariant is model-checked on a finite domain, with ϵ fixed to a concrete unit value and the value domain restricted to a bounded set of integers.
- **TLAPS.** An adequately strong inductive invariant that implies the above invariant (for arbitrary ϵ) is identified, and proven correct using the TLA$^+$ induction rule.

Case study: SmartSum (Chan et al., 2011)

Goal. Given a sequence $l = [l_1, l_2, \ldots]$, release a noisy estimate of every prefix sum $\sum_{i=1}^{k} l_i$ for $k = 1, 2, \ldots$

Core idea. Naively noising every input independently makes errors grow with the prefix length. SmartSum groups inputs into blocks of size Q and noises the block sums, keeping the error bounded.

Setup.

- Input: $l \in \mathrm{Seq}(\mathrm{Value})$.
- Φ: two equally long sequences differing in exactly one position, by ≤ 1.

Result. SmartSum is (2ϵ)-DP with respect to Φ.

Case study: SmartSum (Chan et al., 2011)

```
while (0 < Len(l)) {
  if (Len(l) % Q = 1) {
    x := Lap(Epsilon, c + Head(l));
    n := x + n; next := n;
    c := 0;
    r := Append(r, next);
  } else {
    x := Lap(Epsilon, Head(l));
    next := next + x;
    c := c + Head(l);
    r := Append(r, next);
  };
  l := Tail(l);
};
out := r;
```

Inductive invariant (select clauses):

$$
\begin{aligned}
l_1 \neq l_2 &\Rightarrow v_\epsilon = 0 \\
c_1 \neq c_2 &\Rightarrow l_1 = l_2 \wedge v_\epsilon \leq \epsilon \\
l_1 = l_2 &\Rightarrow v_\epsilon \leq 2\epsilon
\end{aligned}
$$

The differing entry goes through 3 phases:

- not yet processed,
- sitting in the accumulator c,
- absorbed into a noised block sum.

Each phase transition contributes $\leq \epsilon$.

Approximate differential privacy

Pure ϵ-DP is sometimes too strong. (ϵ, δ)-**DP** relaxes the definition with a small additive failure probability.

(ϵ, δ)-differential privacy

For all d, d' with $\Phi(d, d')$ and all A: $\Pr[M(d) \in A] \leq e^{\epsilon} \cdot \Pr[M(d') \in A] + \delta$

Main theorem for (ϵ, δ)-DP (Barthe et al., 2014)

The self-product framework is extended with a second ghost variable v_δ, also accumulated along the execution. If the self-product satisfies the Hoare triple

$$\{\Phi(d_1, d_2) \wedge v_\epsilon = 0 \wedge v_\delta = 0\} \ \lceil c;\ \texttt{return}\ out \rceil \ \{out_1 = out_2 \wedge v_\epsilon \leq \epsilon \wedge v_\delta \leq \delta\}$$

then c; `return` out is (ϵ, δ)-DP with respect to Φ.

Approximate differential privacy in our framework

Two specifications for the Laplace mechanism. The user picks one per call.

Standard: costs ϵ, no δ-cost.

$$\texttt{AbsLapHoareSpec}(\epsilon, e_1, e_2, v_\epsilon, v_\delta) = \{\langle n, n,\ v_\epsilon + \epsilon \cdot |e_1 - e_2|,\ v_\delta \rangle : n \in \mathit{Value}\}$$

Accuracy: requires $e_1 = e_2$, no ϵ-cost, δ-cost is the Laplace tail bound.

$$\begin{aligned}&\texttt{AbsLapAccSpec}(\epsilon, T, e_1, e_2, v_\epsilon, v_\delta)\\ &\qquad = \{\langle n, n,\ v_\epsilon,\ v_\delta + \texttt{LapTail}[\epsilon, T] \rangle : n \in \mathit{Value} \wedge |n - e_1| \leq T\}\end{aligned}$$

Case study: Propose-Test-Release (Dwork & Lei, 2009)

Setup.

- A query is a function $q : \mathit{DBDomain} \to \mathit{QOutDomain}$.
- $\mathrm{Dist} : \mathit{DBDomain} \times \mathit{DBDomain} \to \mathbb{N}$ satisfying the metric axioms.

Distance to instability

$\mathrm{DTI}(q, d)$ is the largest r such that $q(d') = q(d)$ for every d' with $\mathrm{Dist}(d, d') \leq r$.

By definition, $\mathrm{DTI}(q, d)$ is 1-sensitive:

$$|\mathrm{DTI}(q, d_1) - \mathrm{DTI}(q, d_2)| \leq 1 \text{ whenever } \mathrm{Dist}(d_1, d_2) \leq 1.$$

Case study: Propose-Test-Release (Dwork & Lei, 2009)

The PTR algorithm. Given a fixed query q and a threshold parameter T, PTR releases $q(d)$ only when d appears far enough from instability.

```
x := DTI[q, d];
y := Lap(Epsilon, x);
if (AbsVal(y) > T) {
  out := q[d];
} else {
  out := DummyQOut;
}
```

Adjacency. $\Phi(d_1, d_2)$: $\mathrm{Dist}(d_1, d_2) \leq 1$.

Result. PTR is $(\epsilon, \texttt{LapTail}[\epsilon, T])$-DP with respect to Φ.

Case study: Propose-Test-Release (Dwork & Lei, 2009)

Proof sketch. The proof case-splits on whether the two databases agree on the query. Recall that the input to the Laplace call is $\mathrm{DTI}(q, d)$, which is 1-sensitive.

If $q(d_1) = q(d_2)$:
The standard Laplace spec applies. Both branches release the same value, and the call costs $\epsilon \cdot \underbrace{|\mathrm{DTI}(q, d_1) - \mathrm{DTI}(q, d_2)|}_{\leq 1} \leq \epsilon$.

Otherwise, if $q(d_1) \neq q(d_2)$:
Since d_1 and d_2 are adjacent, $\mathrm{DTI}(q, d_1) = \mathrm{DTI}(q, d_2) = 0$. The accuracy spec applies, i.e. $|y - \mathrm{DTI}(q, d_i)| = |y| \leq T$ and the dummy answer is released on both sides. The cost is $\texttt{LapTail}[\epsilon, T]$ on the δ side.

Remark: both specs are needed in the same proof, at the same Laplace call.

Limitations

Limited support for real arithmetic.

- TLA$^+$ and TLAPS do not yet support reasoning about real arithmetic.
- ϵ has type `Real`, and `LapTail`$[\epsilon, T]$ $(2e^{-\epsilon T})$ is not definable in TLA$^+$.
- We work around this by treating `LapTail` as an uninterpreted operator, and isolating trivial real-arithmetic facts as admitted assumptions.

More mechanisms, more case studies.

- The exponential mechanism is supported by the framework, but no case study using it has been completed yet.

Model checking with alternating specs.

- TLC cannot instantiate a constant operator with multiple definitions.
- PTR currently relies on alternating the Hoare and accuracy Laplace specs.
- A unified spec definition that subsumes both seems feasible.

Conclusion and future work

Artifact. `github.com/uguryavuz/tla-dp`

Contribution. A framework for verifying differential privacy of PlusCal programs, by adapting Barthe et al.'s self-product construction to the TLA$^+$ ecosystem. To the best of our knowledge, the first verification of a probabilistic hyperproperty in TLA$^+$.

Future work.

- **DP in concurrent settings.**
  Leverage TLA$^+$'s strengths in modeling concurrent systems to reason about DP algorithms in concurrent and distributed settings.
- **Product constructions for other probabilistic hyperproperties.**
  Extend the approach beyond DP to properties like memory-trace obliviousness for ORAM-like constructions.

Thank you!