Verification of Quantitative Hyperproperties Using Trace Enumeration Relations

Shubham Sahai[1][0000−0002−3434−6937],
Pramod Subramanyan[1][0000−0003−2288−3396], and
Rohit Sinha[2][0000−0001−9107−0239]

[1] Indian Institute of Technology, Kanpur
[2] Visa Research

Abstract. Many important cryptographic primitives offer probabilistic guarantees of security that can be specified as quantitative hyperproperties; these are specifications that stipulate the existence of a certain number of traces in the system satisfying certain constraints. Verification of such hyperproperties is extremely challenging because they involve simultaneous reasoning about an unbounded number of different traces. In this paper, we introduce a technique for verifying quantitative hyperproperties based on the notion of trace enumeration relations. These relations allow us to reduce the problem of trace-counting into one of model-counting of formulas in first-order logic. We also introduce a set of inference rules for machine-checked reasoning about the number of satisfying solutions to first-order formulas (aka model counting). Putting these two components together enables semi-automated verification of quantitative hyperproperties on infinite-state systems. We use our methodology to prove confidentiality of access patterns in Path ORAMs of unbounded size, soundness of a simple interactive zero-knowledge proof protocol as well as other applications of quantitative hyperproperties studied in past work.

1 Introduction

Recent years have seen significant progress in automated and semi-automated techniques for the verification of security requirements of computer systems [4, 10, 16, 19, 30, 47, 50, 55]. Much of this progress has built on the theory of *hyperproperties* [21], and these have been used extensively in analysis of whether systems satisfy secure information flow properties [1, 2, 6, 8, 15, 28, 35, 37, 39, 49, 57] such as observational determinism [41, 55] and non-interference [32]. Unfortunately, the security specification of several important security primitives cannot be captured by secure information flow properties like observational determinism. In particular, observational determinism and non-interference are not applicable when reasoning about algorithms that offer probabilistic – as opposed to deterministic – guarantees of confidentiality and integrity. Prominent examples of security primitives offering probabilistic guarantees include Path ORAM [48] and various zero-knowledge proof protocols.

A promising direction for the verification of such protocols are the class of quantitative hyperproperties [29], one example of which is deniability [12, 14]. Deniability states that for every infinitely-long sequence of observations that an adversary makes, there are (exponentially) many different secrets that could have resulted in exactly these observations. Therefore, the adversary learns very little about the secrets in an execution from a particular sequence of observations.

How does one prove a quantitative hyperproperty like deniability? Suppose our goal is to show that for every trace of adversary observations, there exist 2^n traces with the same observations but different secrets. Here n is a parameter of the system, e.g., the length of a password in bits. One option, first suggested by Yasuoka and Terauchi [54] and recently revisited by Finkbeiner, Hahn, and Torfah [29], is to consider the following k-trace property, where $k = 2^n + 1$.

$$\forall \pi_0.\ \exists \pi_1, \pi_2, \ldots, \pi_{2^n}.$$
$$\Big(\bigwedge_{j=1}^{2^n} \mathit{obs}(\pi_0) = \mathit{obs}(\pi_j) \Big) \wedge \Big(\bigwedge_{j=1}^{2^n} \bigwedge_{k=1}^{2^n} (j \neq k) \Rightarrow \mathit{secret}(\pi_j) \neq \mathit{secret}(\pi_k) \Big)$$

The property states that for every trace of the system, there must exist 2^n other traces with identical observations and pairwise different secrets. In the above, $\pi_0, \pi_1, \ldots$ represent trace variables, $\mathit{obs}(\pi_j)$ refers to the trace of adversary observations projected from the trace π_j, while $\mathit{secret}(\pi_j)$ refers to the trace of secret values in the trace π_j. There are at least three problems with the verification of the above property. First, the size of this property grows exponentially with n; verification needs to reason about 2^n traces simultaneously and is not scalable. The second problem is quantifier alternation. Even if we could somehow reason about 2^n traces, we have to show that *for every* trace π_0, there *exist* 2^n other traces satisfying the above condition. The third problem is that the above technique does not work for *symbolic* bounds. While it is possible – at least in principle – to use the above construction by picking a specific value of n, say 16, to show that 2^{16} traces exist that satisfy deniability, we would like to show that the property holds for all n, where n is a state variable or parameter of the transition system. Capturing the dependence of the trace-count bound on parameters, such as n, is important because it shows that the attacker has to work exponentially harder as n increases. Such general proofs are not possible by reduction to a k-trace property because the construction requires k be bounded.

Recent work by Finkbeiner, Hahn, and Torfah [29] has made significant progress in addressing the first two problems by showing a reduction from k-trace property checking into the problem of maximum model counting [31]. However, their technique still produces a propositional formula whose size grows exponentially in the size of the quantitative hyperproperty. Further, model counting itself is a computationally hard problem that is known to be $\#P$-complete, and maximum model counting is even harder. As a result, their technique does not scale well and times out on the verification of an 8-bit leakage bound for an 8-bit password. Finally, their method does not support symbolic bounds, and therefore cannot be used to verify parametric systems; we verify several examples of such systems in this paper (e.g., Path ORAM [48] of symbolic size).

In this work, we propose a new technique for quantitative hyperproperty verification that addresses each of the above problems. Our approach is based on the following insights. First, instead of trying to count the number of traces that have the same observations and different inputs, we instead show injectivity/surjectivity from satisfying assignments of a first-order formula to traces of a transition system. This allows us to bound the number of traces satisfying the quantitative hyperproperty by the number of satisfying solutions to this formula. We introduce the notion of a trace enumeration relation to formalize this relation between the first-order formula and traces of the transition system. An important advantage of the above reduction is that proving the validity of a trace enumeration relation is only a hyperproperty – not a quantitative hyperproperty.

Next, we develop a novel technique to bound the number of satisfiable solutions to a first-order logic formula, which is of independent interest. While this is a hard problem, we exploit the fact that our formulas have a significant amount of structure. We introduce a set of inference rules inspired by ideas from enumerative combinatorics [13, 52, 56]. These rules allow us to bound the number of satisfying assignments to a formula by making only satisfiability queries.

In summary, our techniques can prove quantitative hyperproperties with symbolic bounds on parametric infinite-state systems. We demonstrate their utility by verifying representative quantitative hyperproperties of diverse applications.

Contributions

1. We introduce a specification language for quantitative hyperproperties (QHPs) over symbolic transition systems and define formal satisfaction semantics for this language. Our specification language is more expressive than past work on QHP specification because it allows the bound to be a first-order formula over the state variables of the transition system.
2. We provide several examples of QHPs relevant to security verification. We identify a new class of QHPs, referred to as soundness hyperproperties, applicable to protocols that provide statistical guarantees of integrity.
3. We propose a novel semi-automated verification methodology for proving that a system satisfies a QHP. Our methodology applies to properties that involve a single instance of quantifier alternation and works by reducing the problem of QHP verification to that of checking non-quantitative hyperproperties over two and three traces of the system and counting satisfiable solutions to a formula in first-order logic.
4. We introduce a set of inference rules for bounding the number of satisfiable solutions to a first-order logic formula, using only satisfiability queries.
5. We demonstrate the applicability of our specification language and verification methodology by providing proofs of security for Path ORAM, soundness of a simple zero-knowledge protocol, as well as examples taken from prior work on quantitative security specifications. We show that our verification methodology scales to larger systems than could be handled in prior work. To the best of our knowledge, our work is the first machine-checked proof of confidentiality of the access patterns in Path ORAM.

2 Motivating Example

In this section, we first introduce the model of transition systems used in this paper. We then discuss quantitative hyperproperty (QHP) specification and verification for our running example – a simple zero-knowledge puzzle.

2.1 Preliminaries

Let $FOL(\mathcal{T})$ denote first-order logic modulo a theory $\mathcal{T}$. The theory $\mathcal{T}$ is assumed to be multi-sorted, includes the theory of linear integer arithmetic (LIA), and contains the $=$ relation. Let $\Sigma_{\mathcal{T}}$ be the theory $\mathcal{T}$'s signature: the set consisting of the constant, function, and predicate symbols in the theory. We say that a formula is a $\Sigma_{\mathcal{T}}$-formula if it consists of the symbols in $\Sigma_{\mathcal{T}}$ along with variables, logical connectives, and quantifiers. We only consider theories which are such that the set of satisfying assignments for any $\Sigma_{\mathcal{T}}$-formula is a countable set.[3]

For every variable x, we will assume there exists a unique variable x', which we refer to as the primed version of x. We will use X, Y, and Z to denote sets of variables. Given a set of variables X, we will use X' to refer to the set consisting of the primed version of each variable in X, that is $X' = \{x' \mid x \in X\}$. Similarly X_1, X_2, etc. are sets consisting of new variables defined as follows: $X_1 = \{x_1 \mid x \in X\}$ and $X_2 = \{x_2 \mid x \in X\}$. We will use $F(X)$ to denote the application of a function or predicate symbol F on the variables in the set X. A satisfying assignment σ to the formula $F(X)$ is written as $\sigma \models F(X)$. Given a formula $F(X)$ and a satisfying assignment σ to this formula, we will denote the valuation of the variable $x \in X$ in the assignment σ as $\sigma(x)$. We will abuse notation in two ways and also write $\sigma(X)$ to refer to a map from the variables $x \in X$ to their assignments in σ. We will also write $\sigma(G(X))$ to denote the valuation of the term $G(X)$ under the assignment σ.

The number of satisfiable assignments for the variables in the set X to a formula $F(X, Y)$ as a function of the variables Y will be denoted by $\#X.\, F(X, Y)$. $\#X.\, F(X, Y)$ is the function $\lambda \mathrm{Y}\ .\ |\{\sigma(X) \mid \sigma \models F(X, \mathrm{Y})\}|$ evaluated at Y; $|S|$ is the cardinality of the set S. For example, consider the predicate $f(i, n) \doteq (0 \leq i < 2n)$. In this case, $\#i.\, f(i, n) = \max(0, 2n)$, meaning that for a given value of $n > 0$, there are $2n$ satisfying assignments to i.

Definition 1 (Transition System). *A transition system M is defined as the tuple $M = \langle X, \mathit{Init}(X), \mathit{Tx}(X, X')\rangle$. X is a finite set of (uninterpreted) constants that represents the state variables of the transition system. Init and Tx are $\Sigma_{\mathcal{T}}$-formulas representing the initial states and the transition relation, respectively. Init is defined over the signature $\Sigma_{\mathcal{T}} \cup X$. Tx is over the signature $\Sigma_{\mathcal{T}} \cup X \cup X'$; X represents the pre-state of the transition and X' represents its post-state.*

A state of the system is an assignment to the variables in X. We use $\sigma^0, \sigma^1, \sigma^2$ etc. to represent states. A trace of the system M is an infinite sequence of states

[3] Our experiments mostly use the AUFLIA theory which allows arrays, uninterpreted functions, and linear integer arithmetic.

$\tau = \sigma^0\sigma^1\sigma^2 \ldots\ \sigma^i \ldots$ such that $\mathit{Init}(\sigma^0)$ is valid and for all $i \geq 0$, $\mathit{Tx}(\sigma^i, \sigma^{i+1})$ is valid; in order to keep notation uncluttered, we will often drop the ≥ 0 qualifier when referring to trace indices. We assume that every state of the transition system has a successor: for all σ there exists some σ' such that $\mathit{Tx}(\sigma, \sigma')$ is valid, ensuring every run of the system is infinite. We will represent traces by τ, τ_1, τ_2, etc. Given a trace τ, we refer to its i^{th} element by τ^i. If $\tau = \sigma^0\sigma^1 \ldots$, then $\tau^0 = \sigma^0$ and $\tau^1 = \sigma^1$. The notation $\tau^{[i,\infty]}$ refers to the suffix of trace τ starting at index i. The set of all traces of the system M is denoted by Φ_M. Given a state σ and a variable $x \in X$, $\sigma(x)$ is the valuation of x in the state σ.

2.2 Motivating Example: Zero-Knowledge Hats

Zero-knowledge (Z-K) proofs are constructions involving two parties: *a prover* and *a verifier*, where the prover's goal is to convince the verifier about the veracity of a given statement without revealing any additional information. We motivate the need for quantitative hyperproperty verification using a Z-K puzzle.

Puzzle Overview: Consider the following scenario. Peggy has a pair of otherwise identical hats of different colors (say, yellow and green). She wants to convince Victor, who is yellow-green color blind, that the hats are of different colors, without revealing the colors of the hats. This problem can be solved using the following interactive protocol. Peggy gives both hats to Victor, and Victor randomly chooses a hat behind a curtain and shows it to Peggy. Next, he goes back behind the curtain and uniformly randomly chooses if he wants to switch the hat or not. He now appears in front of Peggy and asks: "Did I switch?"

If the hats are really of different colors, Peggy will be able to answer correctly with probability 1. If Peggy is cheating – the hats are in fact of the same color – her best strategy is to guess, and with probability 0.5 she will answer incorrectly. If the interaction is repeated k-times, Peggy will be caught with probability $1 - 2^{-k}$. The interaction between Peggy and Victor only reveals the fact that Peggy can detect a switch and not the color of the hat, making this zero-knowledge.

Verification Objectives: A zero-knowledge proof must satisfy three properties: *completeness* (an honest prover should be able to convince an honest verifier of a true statement), *soundness* (a cheating prover can convince an honest verifier with negligible probability) and *zero-knowledge* (no information apart from the veracity of the statement should be revealed). Completeness is a standard trace property, while zero-knowledge is the 2-safety property of indistinguishability. Consequently, the main challenge in automated verification of the zero-knowledge protocol described above is that of soundness. In this section, we discuss its specification and verification using quantitative hyperproperties.

Soundness as a Quantitative Hyperproperty: Consider the transition system $M = \langle X, \mathit{Init}(X), \mathit{Tx}(X, X')\rangle$, shown in Figure 1, representing this protocol.

$$
\begin{array}{lcl}
X & \doteq & \{\mathsf{C}, \mathsf{P}, \mathsf{S}, i, \mathsf{R}\} \\
\mathit{Init}(X) & \doteq & (\forall i.\ 0 \leq \mathsf{C}[i] \leq 1) \wedge (\forall i.\ 0 \leq \mathsf{P}[i] \leq 1) \wedge \mathsf{S} \wedge (i = 1) \wedge (\mathsf{R} > 0) \\
\mathit{Tx}(X, X') & \doteq & (\mathsf{C}' = \mathsf{C}) \wedge (\mathsf{P}' = \mathsf{P}) \wedge (\mathsf{R}' = \mathsf{R}) \wedge \big(\mathsf{S}' = \big(\mathsf{S} \wedge (\mathsf{C}[i] = \mathsf{P}[i])\big)\big) \wedge \\
 & & i' = \min\,(i+1, \mathsf{R})
\end{array}
$$

Fig. 1: Transition system model of the example protocol.

The variable R is a *parameter* of the system and refers to the number of rounds of the protocol. C and P are boolean arrays representing the challenges from the verifier to the prover, and the responses from the prover to the verifier, respectively. i is the current round, and S is a boolean flag that corresponds to whether the zero-knowledge proof has succeeded. C and P are initialized nondeterministically to model the fact that the verifier chooses their challenges randomly, and a cheating prover's best strategy is guessing. While a cheating prover can use any strategy, if the challenges are indistinguishable to her, then the best strategy is to sample responses from a uniform distribution.

Soundness is captured by the following quantitative hyperproperty (QHP):

$$\forall \pi_0 . \# \pi_1 {:}\, \mathbf{F}\,(\delta_{\pi_j, \pi_k}).\ \mathbf{G}\,(\psi_{\pi_0, \pi_1})\ \geq\ 2^{\mathsf{R}} - 1 \tag{1}$$

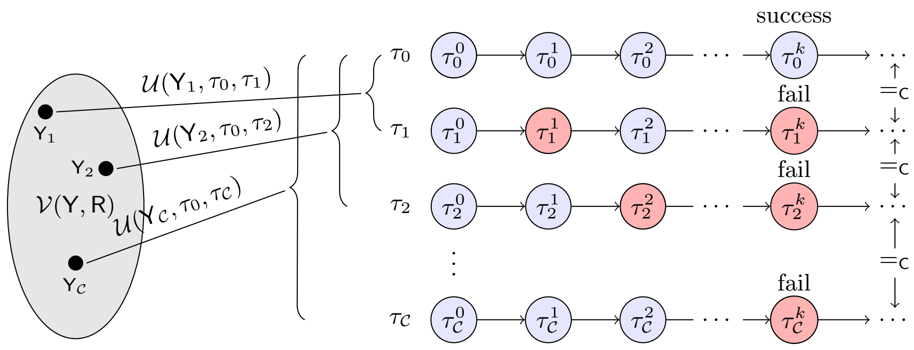
We will provide formal satisfaction semantics for QHPs in Section 3. For now, we informally describe its meaning. The term $\# \pi_1 {:}\, \mathbf{F}\,(\delta_{\pi_j, \pi_k}).\ \mathbf{G}\,(\psi_{\pi_0, \pi_1})\ \geq\ 2^{\mathsf{R}} - 1$ introduces a counting quantifier which stipulates the existence of at least $2^{\mathsf{R}} - 1$ traces satisfying certain conditions: (i) these traces must all be pairwise-different, where difference is defined by satisfaction of the formula $\mathbf{F}\,(\delta_{\pi_j, \pi_k})$ and (ii) all of these traces must be related to trace π_0 by the relation $\mathbf{G}\,(\psi_{\pi_0, \pi_1})$.

The state predicates δ and ψ are defined as follows.

$$
\begin{array}{rcl}
\delta(\sigma_1, \sigma_2) & \doteq & \sigma_1(\mathsf{P}[i]) \neq \sigma_2(\mathsf{P}[i]) \\
\psi(\sigma_1, \sigma_2) & \doteq & \big(\sigma_1((i = \mathsf{R}) \Rightarrow \mathsf{S}) \Rightarrow \sigma_2((i = \mathsf{R}) \Rightarrow \neg \mathsf{S})\big) \quad \wedge \\
 & & \big(\sigma_1(\mathsf{C}) = \sigma_2(\mathsf{C}) \wedge \sigma_1(\mathsf{R}) = \sigma_2(\mathsf{R})\big)
\end{array}
$$

The requirement imposed by δ is that Peggy's responses be different at some step i for every pair of traces captured by the counting quantifier. ψ says that if trace π_0 is a trace where Peggy's cheating succeeds (i.e., $\mathsf{S} = \mathit{true}$ when $i = \mathsf{R}$), then in all traces captured by π_1, the challenges and number of rounds are the same as π_0 but Peggy's cheating is detected by Victor (i.e., $\mathsf{S} = \mathit{false}$ when $i = \mathsf{R}$). These requirements are illustrated in Figure 2(b).

The QHP requires that for every trace in which a cheating prover succeeds in tricking the verifier for a given trace of challenges, there are $2^{\mathsf{R}} - 1$ other traces with the same challenges in which the prover's cheating is detected. Even though soundness is a probabilistic property over the distribution of the system's traces, it can be reduced to counting (and thus specified as a QHP) because each execution trace is sampled uniformly from a finite set. Therefore, if the QHP is satisfied, Peggy's probability of successful cheating is upper-bounded by $2^{-\mathsf{R}}$.


$\mathcal{U}(\mathsf{Y}_1, \tau_0, \tau_1)$
$\mathcal{U}(\mathsf{Y}_2, \tau_0, \tau_2)$
$\mathcal{V}(\mathsf{Y}, \mathsf{R})$
$\mathcal{U}(\mathsf{Y}_\mathcal{C}, \tau_0, \tau_\mathcal{C})$
Y_1
Y_2
$\mathsf{Y}_\mathcal{C}$
success
τ_0 τ_0^0 τ_0^1 τ_0^2 τ_0^k
fail
$=_\mathsf{C}$
τ_1 τ_1^0 τ_1^1 τ_1^2 τ_1^k
fail
$=_\mathsf{C}$
τ_2 τ_2^0 τ_2^1 τ_2^2 τ_2^k
fail
$=_\mathsf{C}$
$\tau_\mathcal{C}$ $\tau_\mathcal{C}^0$ $\tau_\mathcal{C}^1$ $\tau_\mathcal{C}^2$ $\tau_\mathcal{C}^k$

(a) Trace enumeration predicates.

(b) Traces in the soundness QHP.

Fig. 2: Using trace enumeration predicates to verify the soundness QHP.

2.3 Solution Outline

To prove a QHP of the form $\forall \pi_0.\ \#\pi_1 : \Delta_{\pi_j,\pi_k}.\ \varphi \ \triangleleft\ N(Z)$, we construct a *trace enumeration predicate* $\mathcal{V}(\mathsf{Y}, Z)$ and show an injective/bijective mapping from assignments to Y in $\mathcal{V}(\mathsf{Y}, Z)$ and traces of the system. This allows us to prove $\forall \pi_0.\ \#\pi_1 : \Delta_{\pi_j,\pi_k}.\ \varphi \ \triangleleft\ \#Y.\, \mathcal{V}(Y, Z)$. This part of the proof relies on the notion of a trace enumeration relation (§ 4). In the next step, we show that $\#Y.\, \mathcal{V}(Y, Z) \triangleleft N(Z)$ using the inference rules presented in § 5. We now describe these steps in the context of the motivating example.

Verification of Soundness for the Z-K Hats Puzzle: Property 1 is illustrated in Figure 2(b). τ_0 is a trace where the Z-K proof succeeds, while the proof fails for the set of traces $\varPhi_\mathcal{C} = \{\tau_1, \tau_2, \ldots, \tau_\mathcal{C}\}$. The red states show the particular step of the proof in which an incorrect response is given by the prover, and each of these steps as well as their associated prover responses are pairwise different. The QHP is satisfied if $|\varPhi_\mathcal{C}| \geq 2^{\mathtt{R}} - 1$ for every $\tau_0 \in \varPhi_M$, where $\mathtt{R} = \tau_0^0(\mathsf{R})$.

The first step in our methodology is to construct a parameterized relation, called a trace enumeration relation, $\mathcal{U}(\mathsf{Y}, \tau_0, \tau_1)$. This relates τ_0 to each trace in the set $\varPhi_\mathcal{C}$ and is parameterized by Y. For every value of the parameter Y, $\mathcal{U}$ relates a trace in which the proof succeeds (τ_0) to a trace in which the proof fails (τ_1). For every trace τ_0 in which the proof succeeds, the set $\{\tau_1 \mid \exists \mathsf{Y}.\, \mathcal{U}(\mathsf{Y}, \tau_0, \tau_1)\}$ corresponds to the set of traces with the same challenges and the same number of rounds, but with failed proofs of knowledge. Note this is a subset of $\varPhi_\mathcal{C}$.

Next, we construct a predicate $\mathcal{V}(\mathsf{Y}, \mathsf{R})$ which defines valid assignments to Y for a particular value of R. For a particular R, consider the set: $\{\sigma(\mathsf{Y}) \mid \sigma \models \mathcal{V}(\mathsf{Y}, \mathsf{R})\}$. Suppose we are able to show that the relation $\mathcal{U}$ is injective in Y and τ_0 for assignments to Y drawn from this set, then we can lower-bound the size of $\varPhi_\mathcal{C}$ by the size of this set. In other words, we have reduced the problem of trace counting to the problem of counting assignments to $\mathcal{V}(\mathsf{Y}, \mathsf{R})$.

Precisely stated, using $\mathcal{V}$ and $\mathcal{U}$, we show the following.

1. For every trace τ_0, and every assignment Y_i satisfying $\mathcal{V}(\mathsf{Y}_i, \tau_0^0(\mathsf{R}))$, there exists a corresponding trace τ_i that satisfies both $\mathcal{U}(\mathsf{Y}_i, \tau_0, \tau_i)$ and $\psi(\tau_0, \tau_i)$. (Note $\tau_0^0(\mathsf{R})$ refers to the valuation of R in the initial state of τ_0.)
2. Given two different satisfying assignments to $\mathcal{V}$ for a particular value of R, say Y_j and Y_k, the corresponding traces τ_j and τ_k are guaranteed to have different prover responses; in other words, the traces satisfy $\delta(\tau_j, \tau_k)$.

The above two properties, illustrated in Figure 2(a), imply there is an injective mapping from satisfying assignments of $\mathcal{V}(\mathsf{Y}, \mathsf{R})$ to traces in $\mathit{\Phi}_{\mathcal{C}}$. Therefore, the number of traces in $\mathit{\Phi}_{\mathcal{C}}$ can be lower bounded by the number of satisfying assignments to Y in $\mathcal{V}(\mathsf{Y}, \mathsf{R})$, i.e. $\#\mathsf{Y}.\, \mathcal{V}(\mathsf{Y}, \mathsf{R})$. We have reduced the difficult problem of counting traces into a slightly easier problem of counting satisfying assignments to a $FOL(\mathcal{T})$ formula.

The final step is to bound $\#\mathsf{Y}.\, \mathcal{V}(\mathsf{Y}, \mathsf{R})$. For example, one well-known idea from enumerative combinatorics is that if a set A is the union of disjoint sets B and C, then $|A| = |B| + |C|$. Translated to model counting, the above can be written as $\#X.\, F(X, Y) = \#X.\, G(X, Y) + \#X.\, H(X, Y)$ if $F(X, Y) \Leftrightarrow G(X, Y) \lor H(X, Y)$ is valid and $G(X, Y) \land H(X, Y)$ is unsat.[4] We present a set of inference rules in Section 5 that build on this and related ideas. These inference rules allow us derive a machine-checked proof of the bound $\#\mathsf{Y}.\, \mathcal{V}(\mathsf{Y}, \mathsf{R}) \geq 2^{\mathsf{R}} - 1$, thus completing the proof of Property 1 for the Z-K hats puzzle.

3 Overview of Quantitative Hyperproperties

This section introduces a logic for the specification of quantitative hyperproperties over symbolic transition systems. We present satisfaction semantics for this logic and then discuss its applications in security verification.

$$\begin{array}{lcl}
\psi & ::= & \forall \pi.\ \psi \mid \#\pi{:}\, \mathit{\Delta}_{\pi_j, \pi_k}.\ \psi \ \lhd\ N(Z) \mid \varphi \\
\varphi & ::= & \mathcal{P}_{\pi_1, \pi_2, \ldots, \pi_k} \mid \neg\varphi \mid \varphi \lor \varphi \mid \varphi\, \mathbf{U}\, \varphi \mid \mathbf{X}\, \varphi \\
\lhd & ::= & \leq \mid = \mid \geq
\end{array}$$

Fig. 3: Grammar of Quantitative HyperLTL.

3.1 Quantitative Hyperproperties

Figure 3 shows the syntax of Quantitative HyperLTL, our extension of HyperLTL [30] that allows specification of quantitative hyperproperties over symbolic transition systems. There are two noteworthy differences from the presentation

[4] We note there is an implied universal quantifier here. To be precise, we must write $\forall Y.\ \#X.\, F(X, Y) = \#X.\, G(X, Y) + \#X.\, H(X, Y)$.

of HyperLTL in [30]. The first is the predicate $\mathcal{P}_{\pi_1,\pi_2,\ldots,\pi_k}$. This refers to a k-ary state predicate $\mathcal{P}$ that is applied to the first element of each trace in the subscript. These are analogous to atomic propositions in presentations that use Kripke structures and are defined as k-ary state predicates to capture relational properties over traces of the transition system. For example, consider the predicate $\mathcal{P}(\sigma_0, \sigma_1) \doteq (\mathit{input}(\sigma_0) = \mathit{input}(\sigma_1))$. Given this definition, a system M with exactly two traces $\varPhi_M = \{\tau_1, \tau_2\}$ satisfies the HyperLTL formula $\forall \pi_1, \pi_2.\ \mathcal{P}_{\pi_1,\pi_2}$ iff $\mathit{input}(\tau_1^0) = \mathit{input}(\tau_2^0)$. This hyperproperty requires that the input in the initial state of the system be deterministically initialized.

The second difference is the new *counting quantifier*: $\#\pi\colon \varDelta_{\pi_j,\pi_k}.\ \psi \lhd N(Z)$.[5] $\varDelta_{\pi_j,\pi_k}$ is an unquantified HyperLTL formula over two "fresh" trace variables π_j and π_k that encodes when two traces are considered different. ψ is another (possibly-quantified) HyperLTL formula. The operator $\lhd$ can be $\leq$, $=$, or $\geq$. $N(Z)$ is an integer-sorted term in $\mathit{FOL}(\mathcal{T})$ over the variables in the set Z, $Z \subset X$ where X is the set of state variables of the transition system under consideration. Z typically refers to the subset of the state variables that define the parameters of the transition system; e.g. $Z = \{\mathsf{R}\}$ for the Z-K proof transition system in Figure 1, the number of blocks in a model of Path ORAM, the size of an array, etc. Typically, the variables in the set Z do not change after initialization. Informally stated, the counting quantifier is satisfied if a maximally large set $\varPhi_{\mathcal{C}} \subseteq \varPhi$, satisfying the two conditions below, has cardinality $\lhd$ *count* where *count* is the valuation of $N(Z)$ in the initial state of every trace in $\varPhi_{\mathcal{C}}$. Those conditions are: (i) each of the traces in $\varPhi_{\mathcal{C}}$ are pairwise different as defined by satisfaction of $\varDelta_{\pi_j,\pi_k}$, and (ii) every trace in this set satisfies the HyperLTL formula ψ.

The remaining operators are standard, so we do not discuss them further and instead provide formal satisfaction semantics.

Satisfaction Semantics of Quantitative HyperLTL The validity judgement of a property φ by a set of traces $\varPhi$ is defined with respect to a trace assignment $\varPi : \mathit{Vars} \rightarrow \varPhi$. Here, Vars is the set of trace variables. We use $\pi, \pi_1, \pi_2, \ldots$ to refer to trace variables.[6] The partial function $\varPi$ is a mapping from trace variables to traces. We use the notation $\varPi[\pi \mapsto \tau]$ to refer to a trace assignment that is identical to $\varPi$ except for the trace variable π which now maps to the trace τ. We write $\varPi \models_{\varPhi} \psi$ if the set of traces $\varPhi$ satisfies the property ψ under the trace assignment $\varPi$. We will drop the subscript $\varPhi$ from $\models_{\varPhi}$ if it is clear from the context or irrelevant. The notation $\varPi^{[i,\infty]}$ is an abbreviation for the new trace assignment obtained by taking the suffix starting from index i of every trace in $\varPi$: $\varPi^{[i,\infty]}(\pi) = \varPi(\pi)^{[i,\infty]}$ for every trace $\pi \in \mathit{dom}(\varPi)$ where $\mathit{dom}(\varPi)$ is the domain of $\varPi$. We write $\varPi \not\models_{\varPhi} \psi$ when $\varPi \models_{\varPhi} \psi$ is not satisfied. Satisfaction rules for HyperLTL formulas are shown in Figure 4.

[5] A counting quantifier over Kripke structures was introduced by Finkbeiner et al. [29]. Our definition is slightly different and a detailed comparison is deferred to Section 7.

[6] Note the distinction between trace variables denoted by π_1, π_2, etc. and traces which are denoted by τ_1, τ_2, etc.

$\Pi \models_{\Phi} \forall\pi.\ \psi$	iff for all $\tau \in \Phi : \Pi[\pi \mapsto \tau] \models_{\Phi} \psi$
$\Pi \models_{\Phi} \#\pi{:}\, \Delta_{\pi_j,\pi_k}.\ \psi\ \triangleleft\ N(Z)$	iff $\lvert\Phi_{\mathcal{C}}\rvert = 0 \Rightarrow 0 \triangleleft N(Z)$ is valid, and $\lvert\Phi_{\mathcal{C}}\rvert > 0 \Rightarrow \forall\tau \in \Phi_{\mathcal{C}}.\ \lvert\Phi_{\mathcal{C}}\rvert \triangleleft \tau^0(N(Z))$, where, $\Phi_{\mathcal{C}} \subseteq \Phi$ is a maximally large set such that: $\forall\tau_j, \tau_k \in \Phi_{\mathcal{C}}.$ $\tau_j \neq \tau_k \Leftrightarrow \{\pi_j \mapsto \tau_j, \pi_k \mapsto \tau_k\} \models\ \Delta_{\pi_j,\pi_k}$ and, $\forall\tau \in \Phi_{\mathcal{C}}.\ \Pi[\pi \mapsto \tau] \models_{\Phi} \psi$
$\Pi \models_{\Phi} \mathcal{P}_{\pi_1,\ldots,\pi_k}$	iff $\mathcal{P}(\Pi(\pi_1)^0, \ldots, \Pi(\pi_k)^0)$ is valid
$\Pi \models_{\Phi} \neg\psi$	iff $\Pi \not\models_{\Phi} \psi$
$\Pi \models_{\Phi} \psi \vee \varphi$	iff $\Pi \models_{\Phi} \psi$ or $\Pi \models_{\Phi} \varphi$
$\Pi \models_{\Phi} \mathbf{X}\,\varphi$	iff $\Pi^{[1,\infty]} \models_{\Phi} \varphi$
$\Pi \models_{\Phi} \varphi\,\mathbf{U}\,\psi$	iff there exists $j \geq 0 : \Pi^{[j,\infty]} \models_{\Phi} \psi$ and for all $0 \leq i < j : \Pi^{[i,\infty]} \models_{\Phi} \varphi$

Fig. 4: Satisfaction semantics for Quantitative HyperLTL formulas over symbolic transition systems.

Definition 2 (Quantitative HyperLTL Satisfaction). *We say that the transition system M satisfies the property ψ, denoted by $M \models \psi$ if the empty trace assignment $\emptyset$ satisfies formula ψ for the set of traces Φ_M, that is $\emptyset \models_{\Phi_M} \psi$.*

Additional Operators: The above showed the minimal set of operators required in Quantitative HyperLTL. The rest of this paper will use the other standard operators such as $\wedge$ (conjunction), $\Rightarrow$ (implication), $\mathbf{F}$ (future/eventually) and $\mathbf{G}$ (globally/always) which can be defined in terms of the operators in Figure 3.

Well-defined Formulas: In order for the semantics of Quantified HyperLTL to be meaningful, we need certain semantic restrictions on the structure of QHPs.

Definition 3 (Well-defined QHPs). *An instance of a counting quantifier $\#\pi{:}\ \Delta_{\pi_j,\pi_k}.\ \varphi\ \triangleleft\ N(Z)$ is said to be well-defined if:*

1. *$\neg\Delta_{\pi_j,\pi_k}$ is an equivalence relation over the set of all traces Φ, and*
2. *In every set of the traces $\Phi_{\mathcal{C}}$ captured by the counting quantifier in the semantics shown in Figure 4, the term $N(Z)$ has the same valuation for all initial states: $\forall\tau_i, \tau_j \in \Phi_{\mathcal{C}}.\ \tau_i^0(N(Z)) = \tau_j^0(N(Z))$.*

A Quantified HyperLTL formula is said to be well-defined if every instance of a counting quantifier in the formula is well-defined.

Example 1 (Well-defined QHPs). The QHPs presented in the rest of this paper are all well-defined, so here we give an example of a QHP that is *not* well-defined. Consider this variant of Property 1: $\forall\pi_0.\#\pi_1{:}\ \mathit{true}.\ \mathbf{G}\,(\psi_{\pi_0,\pi_1})\ \geq\ 2^{\mathsf{R}} - 1$. This is not a well-defined QHP because Δ_{π_j,π_k} in the counting quantifier is simply *true*, and its negation is not an equivalence relation over the set of traces.

Note that condition (1) in the definition above affects Δ_{π_j,π_k} while condition (2) places a restriction on φ. The former condition prevents double-counting of traces, while the latter ensures that the trace count is unambiguous.

The properties in our experiments require only syntactic checks to verify well-definedness. Specifically, Δ_{π_j,π_k} is always of the form $\mathbf{F}\,(\mathcal{P}_{\pi_j,\pi_k})$ where $\mathcal{P}$ is of the form $\mathcal{P}(\sigma_1, \sigma_2) \doteq f(\sigma_1) \neq f(\sigma_2)$. The negation of this is obviously an equivalence relation over the set of all traces. Secondly, our QHPs are of the form $\forall \pi_0.\ \#\pi_1\colon \Delta_{\pi_j,\pi_k}.\ \varphi \ \triangleleft\ N(Z)$ where φ enforces equality of the variables in Z between the traces π_0 and π_1. These two features guarantee well-definedness. In the rest of this paper, we only consider well-defined QHPs.

3.2 Applications of QHPs in Security Specification

Deniability: Our first example of a quantitative hyperproperty is deniability. Suppose $obs(\sigma)$ is a term that corresponds to the adversary observable part of the state σ, while $secret(\sigma)$ corresponds to the secret component of the state σ. Deniability is satisfied when every trace of adversary observations can be generated by at least $N(Z)$ different secrets. For this, we define $\delta(\sigma_1, \sigma_2) \doteq secret(\sigma_1) \neq secret(\sigma_2)$ and $\approx^O (\sigma_1, \sigma_2) \doteq obs(\sigma_1) = obs(\sigma_2)$.

$$\forall \pi_0.\#\pi_1\colon \mathbf{F}\,(\delta_{\pi_j,\pi_k}).\ \mathbf{G}\,(\approx^O_{\pi_0,\pi_1}) \ \geq\ N(Z)$$

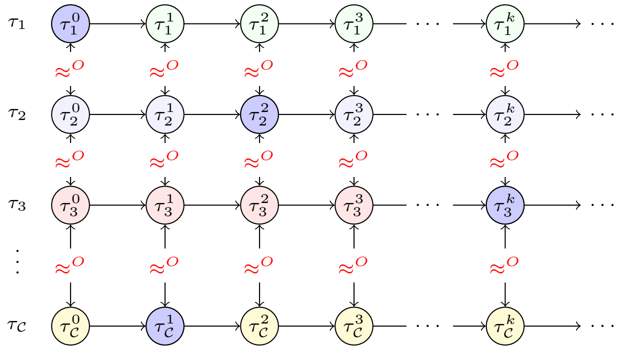

τ_1
τ_2
τ_3
$\tau_\mathcal{C}$
τ_1^0 τ_1^1 τ_1^2 τ_1^3 $\cdots$ τ_1^k $\cdots$
$\approx^O$
τ_2^0 τ_2^1 τ_2^2 τ_2^3 $\cdots$ τ_2^k $\cdots$
τ_3^0 τ_3^1 τ_3^2 τ_3^3 $\cdots$ τ_3^k $\cdots$
$\tau_\mathcal{C}^0$ $\tau_\mathcal{C}^1$ $\tau_\mathcal{C}^2$ $\tau_\mathcal{C}^3$ $\cdots$ $\tau_\mathcal{C}^k$ $\cdots$

Fig. 5: Illustrating deniability.

Figure 5 illustrates deniability. It shows a set of traces $\Phi_\mathcal{C} := \{\tau_1, \tau_2, \ldots, \tau_\mathcal{C}\}$; the circles represent the states in each trace and the secret values are shown by color of the circle. For these traces, every pair of corresponding states have the same observations: represented by $\approx^O$, and every distinct pair of traces differ in the secrets. Deniability is satisfied if $|\Phi_\mathcal{C}| \geq N(Z)$. Satisfaction implies that every trace of adversary observations has at least $N(Z)$ counterparts with identical observations but different values of $secret(\sigma)$. If we can show in a system satisfying deniability that each trace of secrets is equiprobable and $N(Z)$ grows exponentially in some parameters of the system, then we can conclude that the system satisfies computational indistinguishability. Deniability can capture probabilistic notions of confidentiality, such as confidentiality of Path ORAM.

Soundness: While deniability encodes a form of confidentiality, soundness is its dual in the context of integrity. One example of soundness was given in § 2.2 for the Z-K hats puzzle. Soundness is generally applicable to protocols that offer probabilistic integrity guarantees. For instance, many interactive challenge-response protocols consist of repeated rounds such that if the prover succeeds in all rounds, the verifier can be convinced with a high probability that the prover is not cheating. This can be viewed as a QHP stating that for every trace in which a dishonest prover tricks a verifier into accepting an invalid proof, there are at least $N(Z)$ other traces with different prover responses in which the cheating is detected. As usual, we require that traces be uniformly sampled from a finite set in order to state soundness as a QHP.

Soundness is stated as $\forall\pi_0.\#\pi_1\colon \mathbf{F}\,(\delta_{\pi_j,\pi_k}).\ \mathbf{G}\,(\psi_{\pi_0,\pi_1}) \ \geq\ N(Z)$. The relation δ is defined as two states having different prover responses. ψ requires the challenge-response protocol to fail in π_1 if it succeeded in π_0 and also that the system parameters (the variables in Z) be identical between π_0 and π_1.

Summarizing QHP Specification: These examples demonstrate that QHPs have important applications in security verification. They capture probabilistic notions of both confidentiality and integrity. In particular, the following form of QHPs consisting of a single quantifier alternation seems especially relevant for security verification: $\forall\pi_0.\ \#\pi_1\colon \varDelta_{\pi_j,\pi_k}.\ \varphi\ \triangleleft\ N(Z)$. Each of the examples of quantitative hyperproperties discussed in the previous subsection – deniability, soundness, as well as others like quantitative non-interference [46, 54] fit in this template. Therefore, in the rest of this paper, we focus on developing scalable verification techniques for QHPs that follow this template.

4 Trace Enumerations

This section introduces the notion of a trace enumeration, which is a technique that allows us to reduce the problem of counting traces to that of counting satisfiable assignments to a formula in $\mathit{FOL}(\mathcal{T})$.

4.1 Trace Enumeration Relations

We now formalize injective trace enumerations, which allow us to lower-bound the number of traces captured by a counting quantifier in a QHP.

Definition 4 (Injective Trace Enumeration). *Let us consider a transition system $M = \langle X, \mathit{Init}(X), \mathit{Tx}(X, X')\rangle$ and the relation $\mathcal{U}(Y, \tau_1, \tau_2)$ where Y is a set of variables disjoint from X, τ_1 and τ_2 are traces of this transition system.*

Let $\forall \pi_0.\ \#\pi_1 \colon \Delta_{\pi_j,\pi_k}.\ \varphi\ \geq\ N(Z)$ *be a QHP where* $Z \subset X$*. Suppose* $\mathcal{V}(Y, Z)$ *is a predicate over the variables in* Y *and* Z*. We say that the pair* $\mathcal{V}(Y, Z)$ *and* $\mathcal{U}(Y, \tau_1, \tau_2)$ *form an injective trace enumeration of the system* M *for the QHP* $\forall \pi_0.\ \#\pi_1 \colon \Delta_{\pi_j,\pi_k}.\ \varphi\ \geq\ N(Z)$ *iff the following conditions are satisfied:*

1. *For every trace* τ_0 *in* Φ_M *and every satisfying assignment* $(\mathtt{Y}, \mathtt{Z})$ *for the predicate* $\mathcal{V}(Y, Z)$*, there exists a trace* $\tau_1 \in \Phi_M$ *which is related to the trace* τ_0 *as per the relation* $\mathcal{U}$ *via this same assignment to* Y*. Further, the pair* τ_0 *and* τ_1 *satisfy the property* φ *and the valuation of the variables in* Z *in the initial state of* τ_1 *is equal to* $\mathtt{Z}$*.*

$$\begin{aligned}&\forall \tau_0 \in \Phi_M, \mathtt{Y}, \mathtt{Z}.\ \mathcal{V}(\mathtt{Y}, \mathtt{Z}) \Rightarrow \\ &\quad \big(\exists \tau_1 \in \Phi_M.\ \mathcal{U}(\mathtt{Y}, \tau_0, \tau_1) \land \{\pi_0 \mapsto \tau_0, \pi_1 \mapsto \tau_1\} \models \varphi \land \tau_1^0(Z) = \mathtt{Z}\big)\end{aligned} \tag{2}$$

2. *Different assignments to the variables in* Y *for the formula* $\mathcal{V}(Y, Z)$ *enumerate different traces in* $\mathcal{U}(Y, \tau_0, \tau_1)$*, where "different" means satisfaction of* Δ_{π_j,π_k}*.*

$$\begin{aligned}&\forall \tau_0, \tau_1, \tau_2 \in \Phi_M, \mathtt{Y}_1, \mathtt{Y}_2, \mathtt{Z}. \\ &\quad \mathcal{V}(\mathtt{Y}_1, \mathtt{Z}) \land \mathcal{V}(\mathtt{Y}_2, \mathtt{Z}) \land \mathtt{Y}_1 \neq \mathtt{Y}_2 && \Rightarrow \\ &\quad \mathcal{U}(\mathtt{Y}_1, \tau_0, \tau_1) \land \mathcal{U}(\mathtt{Y}_2, \tau_0, \tau_2) \land \tau_1^0(Z) = \mathtt{Z} \land \tau_2^0(Z) = \mathtt{Z} && \Rightarrow \\ &\quad \{\pi_j \mapsto \tau_1, \pi_k \mapsto \tau_2\} \models\ \Delta_{\pi_j,\pi_k}\end{aligned} \tag{3}$$

If $\mathcal{V}$ and $\mathcal{U}$ form an injective trace enumeration M for the property $\forall \pi_0.\ \#\pi_1 \colon \Delta_{\pi_j,\pi_k}.\ \varphi\ \geq\ N(Z)$, then for every trace τ_0, there exist at least as many traces satisfying the counting quantifier as there are satisfying assignments to Y in $\mathcal{V}(Y, Z)$. This is made precise in the following lemma.

Lemma 1. *[Trace Count Lower-Bound] If* $\mathcal{V}(Y, Z)$ *and* $\mathcal{U}(Y, \tau_1, \tau_2)$ *form an injective trace enumeration of the system* M *for the QHP* $\forall \pi_0.\ \#\pi_1 \colon \Delta_{\pi_j,\pi_k}.\ \varphi\ \geq\ N(Z)$ *and if* $\#Y.\, \mathcal{V}(Y, Z)$ *is finite for all assignments to* Z*, then* $M \models \forall \pi_0. \#\pi_1 \colon \Delta_{\pi_j,\pi_k}.\ \varphi\ \geq\ \#Y.\, \mathcal{V}(Y, Z)$*.*

Example 2 (Injective Trace Enumeration). Let $\mathsf{P}_0[1], \ldots, \mathsf{P}_0[\mathsf{R}]$ be a trace of correct responses for some particular sequence of challenges for our running example. Consider the array $\mathsf{Y}[1], \mathsf{Y}[2], \ldots, \mathsf{Y}[\mathsf{R}]$ where each $\mathsf{Y}[j] \in \{0, 1\}$. Y is a boolean array of size R, and $\mathsf{Y}[i] = 1$ means that the prover gives an incorrect response to the challenge in round i. We can define the predicate $\mathcal{V}$ as follows.

$$\mathcal{V}(\mathsf{Y}, \mathsf{R}) \doteq \big(\exists i.\ 1 \leq i \leq \mathsf{R} \land \mathsf{Y}[i] \neq 0\big) \land \big(\forall i.\ (i < 1 \lor i > \mathsf{R}) \Rightarrow \mathsf{Y}[i] = 0\big) \tag{4}$$

The above definition ensures that at least one response is incorrect. Notice that for every assignment to Y except the assignment of all zeros, the trace of responses defined by $\forall j.\ \mathsf{P}_1[j] = \mathsf{P}_0[j] \oplus \mathsf{Y}[j]$ (where $\oplus$ is exclusive or) corresponds to a valid trace of the system and satisfies the counting quantifier in Property 1.

Specifically, every such response from the prover is incorrect and will result in the protocol failing. We can use the above facts to define the relation $\mathcal{U}$ as follows:

$$\begin{aligned}\mathcal{U}(\mathsf{Y},\tau_1,\tau_2) \doteq \big(\forall j.\ \tau_1^{0}(\mathsf{P}[j]) = \tau_2^{0}(\mathsf{P}[j]) \oplus \mathsf{Y}[j]\big) \quad & \wedge \\ \tau_1^{0}(\mathsf{C}) = \tau_2^{0}(\mathsf{C}) \wedge \tau_1^{0}(\mathsf{R}) = \tau_2^{0}(\mathsf{R}) \wedge (\tau_1^{\mathsf{R}}(S) \Rightarrow \neg\tau_2^{\mathsf{R}}(S)) &\end{aligned} \tag{5}$$

The pair $\mathcal{V}$ and $\mathcal{U}$ form an injective trace enumeration for the system M (defined in Figure 1) for the Property 1. This is because different Y's will result in different prover responses for the same challenges. By Lemma 1, we can conclude that Property 1 is satisfied if $\#\mathsf{Y}.\, \mathcal{V}(\mathsf{Y},\mathsf{R}) \geq 2^{\mathsf{R}} - 1$

Analogous to injective trace enumerations, it is also possible to define surjective trace enumerations that upper-bound the number of traces captured by a counting quantifier. Details of surjective trace enumerations are presented in the extended version of the paper [43].

5 Model Counting

As discussed in the previous section, trace enumeration relations can bound the number of satisfying traces in a QHP. Given a QHP $\forall\pi_0.\ \#\pi_1 : \Delta_{\pi_j,\pi_k}.\ \varphi \ \triangleleft\ N(Z)$, appropriate trace enumeration predicates $\mathcal{V}(Y,Z)$ and $\mathcal{U}$ can be used to derive that $\forall\pi_0.\ \#\pi_1 : \Delta_{\pi_j,\pi_k}.\ \varphi \triangleleft \#Y.\, \mathcal{V}(Y,Z)$. The final step in our verification methodology is to show validity of $\#Y.\, \mathcal{V}(Y,Z) \triangleleft N(Z)$. To that end, this section discusses our novel technique for model counting.

5.1 Model Counting via SMT Solving

Our approach borrows ideas from enumerative combinatorics [13, 52, 56] and introduces the inference rules shown in Figure 6 to reason about model counts for formulas in $\mathit{FOL}(\mathcal{T})$. Each of the conclusions in the inference rules is a statement involving model counts of $\mathit{FOL}(\mathcal{T})$ formulas, while each of the premises is a formula in $\mathit{FOL}(\mathcal{T})$ that *does not involve model counts* and can, therefore, be checked using SAT/SMT solvers. Most of the rules are straightforward, and we do not describe them due to space constraints. The three interesting rules – *Injectivity*, $\mathit{Ind}_{\leq}$ and $\mathit{Ind}_{\geq}$ – are discussed below.

Injectivity: This rule is based on the following idea from enumerative combinatorics. Suppose we have two sets A and B. We can show that $|A| \leq |B|$ if there exists an injective function from A to B. Translating this to model counts, the set A in the rule corresponds to satisfying assignments to $f(X)$, B corresponds to satisfying assignments to $g(Y)$ and $\mathscr{F}$ is the injective witness function.

$\mathit{Ind}_{\geq}$ and $\mathit{Ind}_{\leq}$: Suppose the formulas $f(X,n)$ and $g(Y,n)$ are parameterized by the integer variable n. If an injective witness function $\mathscr{G}(X,Y,n)$ is able to "lift" satisfying assignments of $f(X_n,n)$ and $g(Y_n,n)$ into a satisfying assignment of

$f(X_{n+1}, n+1)$, then we can conclude that the number of satisfying assignments to $f(X, n+1)$ are at least as many as the product of the number of satisfying assignments to $f(X, n)$ and $g(Y, n)$. $\mathit{Ind}_{\le}$ is the surjective version of this rule. It applies when a satisfying assignment to $f(X_{n+1}, n+1)$ can be "lowered" into satisfying assignments to $f(X_n, n)$ and $g(Y_n, n)$ where the values of X_n and Y_n are given by the witness functions $\mathscr{H}_x$ and $\mathscr{H}_y$ respectively.

$$\frac{}{(\#i.\, a \le i < b) \;=\; \max(b-a, 0)}\;\mathit{Range} \qquad \frac{}{\#Y.\, f(X) \;\ge\; 0}\;\mathit{Positive}$$

$$\frac{\bigwedge_{i=1}^{c} f(X_i) \wedge \mathit{distinct}(X_1, \ldots, X_c) \text{ is } \mathsf{sat}}{\#X.\, f(X) \;\ge\; c}\;\mathit{ConstLB}$$

$$\frac{\bigwedge_{i=1}^{c} f(X_i) \wedge \mathit{distinct}(X_1, \ldots, X_c) \text{ is } \mathsf{unsat}}{\#X.\, f(X) \;<\; c}\;\mathit{ConstUB}$$

$$\frac{f(X, Y) \Rightarrow g(X, Y)}{\#X.\, f(X, Y) \;\le\; \#X.\, g(X, Y)}\;\mathit{UB}$$

$$\frac{h(X, Y) \Leftrightarrow f(X) \wedge g(Y)}{\#X \cup Y.\, h(X, Y) \;\le\; \#X.\, f(X) \times \#Y.\, g(Y)}\;\mathit{AndUB}$$

$$\frac{\begin{array}{c} f(X) \Rightarrow g(\mathscr{F}(X)) \\ \big(f(X_1) \wedge f(X_2) \wedge X_1 \ne X_2\big) \Rightarrow \mathscr{F}(X_1) \ne \mathscr{F}(X_2) \end{array}}{\#X.\, f(X) \;\le\; \#Y.\, g(Y)}\;\mathit{Injectivity}$$

$$\frac{h(X, Y) \Leftrightarrow f(X) \wedge g(Y) \quad X \cap Y = \emptyset}{\#X \cup Y.\, h(X, Y) \;=\; \#X.\, f(X) \times \#Y.\, g(Y)}\;\mathit{Disjoint}$$

$$\frac{f(X, Y) \Leftrightarrow g(X, Y) \vee h(X, Y)}{\#X.\, f(X, Y) \;=\; \#X.\, g(X, Y) + \#X.\, h(X, Y) - \#X.\, \big(g(X, Y) \wedge h(X, Y)\big)}\;\mathit{Or}$$

$$\frac{\begin{array}{c} \big(f(X, n) \wedge g(Y, n)\big) \Rightarrow f(\mathscr{G}(X, Y, n), n+1) \\ (X_1 \ne X_2 \vee Y_1 \ne Y_2) \Rightarrow \mathscr{G}(X_1, Y_1, n) \ne \mathscr{G}(X_2, Y_2, n) \end{array}}{\#X.\, f(X, n+1) \;\ge\; \#X.\, f(X, n) \times \#Y.\, g(Y, n)}\;\mathit{Ind}_{\ge}$$

$$\frac{\begin{array}{c} f(X, n+1) \Rightarrow \big(f(\mathscr{H}_x(X, n+1), n) \wedge g(\mathscr{H}_y(X, n+1), n)\big) \\ X_1 \ne X_2 \Rightarrow \big(\mathscr{H}_x(X_1, n) \ne \mathscr{H}_x(X_2, n) \vee \mathscr{H}_y(Y_1, n) \ne \mathscr{H}_y(Y_2, n)\big) \end{array}}{\#X.\, f(X, n+1) \;\le\; \#X.\, f(X, n) \times \#Y.\, g(Y, n)}\;\mathit{Ind}_{\le}$$

Fig. 6: Model counting proof rules. Unless otherwise specified, premises are satisfied when the formula is valid. Conclusions have an implicit universal quantifier.

5.2 Model Counting in the Motivating Example

The definition of the predicate $\mathcal{V}$ in the motivating example is shown below.

$$\mathcal{V}(\mathsf{Y}, \mathsf{R}) \doteq \big(\exists i.\ 1 \le i \le \mathsf{R} \wedge \mathsf{Y}[i] \ne 0\big) \wedge \big(\forall i.\ ((i < 1 \vee i > \mathsf{R}) \Rightarrow \mathsf{Y}[i] = 0)\big)$$

Our task is to show $\#\mathsf{Y}.\,\mathcal{V}(\mathsf{Y},\mathsf{R}) = 2^{\mathsf{R}} - 1$. Recall that Y is an array of binary values (i.e. the integers 0 and 1) and consider the following predicates: $\mathcal{V}_f(\mathsf{Y},\mathsf{R}) \doteq \big(\forall i.\ (i < 1 \vee i > \mathsf{R}) \Rightarrow \mathsf{Y}[i] = 0\big)$, $\mathcal{V}_1(\mathsf{Y},\mathsf{R}) \doteq \big(\forall i.\ \mathsf{Y}[i] = 0\big)$ and $\mathcal{W}(i) \doteq 0 \leq i < 2$. Using these definitions, the proof is as follows.

1. (*ConstUB*, *Positive*) $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}) \wedge \mathcal{V}_1(\mathsf{Y},\mathsf{R}) = 1$.
2. (*Or*) $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}) = \#\mathsf{Y}.\,\mathcal{V}(\mathsf{Y},\mathsf{R}) + \#\mathsf{Y}.\,\mathcal{V}_1(\mathsf{Y},\mathsf{R})$.
3. (*ConstLB*, *ConstUB*) $\#\mathsf{Y}.\,\mathcal{V}_1(\mathsf{Y},\mathsf{R}) = 1$.
4. (*ConstLB*, *ConstUB*) $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},1) = 2$.
5. ($Ind_{\leq}$): $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}) \leq \#i.\,\mathcal{W}(i) \times \#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}-1)$.
6. ($Ind_{\geq}$): $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}) \geq \#i.\,\mathcal{W}(i) \times \#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}-1)$.
7. (*Range*): $\#i.\,\mathcal{W}(i) = 2$.
8. $(4 - 7)$ imply that $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}) = 2 \times \#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}-1)$, $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},1) = 2$, this means $\#\mathsf{Y}.\,\mathcal{V}_f(\mathsf{Y},\mathsf{R}) = 2^{\mathsf{R}}$.
9. (2, 3, 8) imply that $\#\mathsf{Y}.\,\mathcal{V}(\mathsf{Y},\mathsf{R}) = 2^{\mathsf{R}} - 1$.

In step 5, the witness function is $\mathscr{G}(\mathsf{Y},\mathsf{R},i) \doteq \mathsf{Y}[\mathsf{R}+1 \mapsto i]$, while in step 6, they are $\mathscr{H}_{\langle\mathsf{Y},\mathsf{R}\rangle}(\mathsf{Y},\mathsf{R}+1) \doteq \langle\mathsf{Y}[\mathsf{R}+1 \mapsto 0],\mathsf{R}\rangle$ and $\mathscr{H}_i(\mathsf{Y},\mathsf{R}+1) \doteq (\mathsf{Y}[\mathsf{R}+1])$.[7] Note steps 8 and 9 are automatically discharged by the SMT solver.

6 Experimental Results and Discussion

In this section, we present an experimental evaluation of the use of trace enumerations for the verification of quantitative hyperproperties.

6.1 Methodology

We studied five systems with varying complexity and QHPs. These were modeled in the Uclid5 modeling and verification framework [44, 51], which uses the Z3 SMT solver (v4.8.6) [23] to discharge the proof obligations. The experiments were run on an Intel i7-4770 CPU @ 3.40GHz with 8 cores and 32 GB RAM.

The verification conditions are currently manually generated from the models, but automation of this is straightforward and ongoing. The k-trace properties were proven using self-composition [9, 10] and induction. A number of strengthening invariants had to be specified manually for the inductive proofs. Many of the invariants are relational *and* quantified and, therefore, difficult to infer algorithmically. We note that recent work has made progress toward automated inference of quantified invariants [27, 36].

6.2 Overview of Results

Due to limited space, we only provide a brief description of our benchmarks for evaluation and refer the interested reader to the extended version of our paper [43] for a more detailed discussion. We have also made the models and associated proof scripts available at [25]. A brief overview of the case studies follows.

[7] The notation $arr[i \mapsto v]$ denotes an array that is identical to arr except for index i which contains v.

Table 1: Verification Results of Models.

Benchmark	Hyperproperty	Model LoC	Proof LoC	Num. Annot.	Verif. Time
Electronic Purse [7]	Deniability	46	93	9	3.92s
Password checker [29]	Quantitative non-interference	59	100	10	4.69s
F-Y Array Shuffle	Quantitative information flow	86	195	96	7.38s
ZK Hats (Sec. 2.2)	Soundness	91	191	36	6.34s
Path ORAM [48]	Deniability	587	209	142	9.74s

1. **Electronic Purse.** We model an electronic purse, with a secret initial balance, proposed by Backes et al. [7]. A fixed amount is debited from the purse until the balance is insufficient for the next transaction. We prove a deniability property: there is a sufficient number of traces with identical attacker observations but different initial balances.
2. **Password Checker.** We model the password checker from Finkbeiner et al. [29], but we allow passwords of unbounded length n. We prove quantitative non-interference: information leakage to an attacker is $\leq n$ bits.
3. **Array Shuffle.** We implement a variant of the Fisher-Yates shuffle. We chose this because producing random permutations of an array is an important component of certain cryptographic protocols (e.g., Ring ORAM [40]). We prove a quantitative information flow property stating that all possible permutations are indeed generated by the shuffling algorithm.
4. **ZK Hats.** We prove soundness of the zero-knowledge protocol in Section 2.
5. **Path ORAM.** Discussed in Section 6.3.

The properties we prove on these models and the results of our evaluation are presented in Table 1 which shows the size of each model, the number of lines of proof code (this is the code for self-composition, property specification, etc.), the number of verification annotations (invariants and procedure pre-/post-conditions) and the verification time for each example. Once the auxiliary strengthening invariants are specified, the verification completes within a few seconds. This suggests that the methodology can scale to larger models, and even implementations. The main challenge in the application of the methodology is the construction of the trace enumeration relations, associated witness functions, and the specification of strengthening invariants. Each of these requires application-specific insight. Since most of our enumerations and invariants are quantified, some of the proofs also required tweaking the SMT solver's configuration options (e.g. turning off model-based quantifier instantiation in Z3).

6.3 Deniability of Path ORAM

In this section, we discuss our main case study: the application of trace enumerations for verifying deniability of server access patterns in Path ORAM [48], a practical variant of Oblivious RAM (ORAM) [33]. ORAMs refer to a class of algorithms that allow a client with a small amount of storage to store/load a large amount of data on an untrusted server while concealing the client access pattern from the server. Path ORAM stores encrypted data on the server in an augmented binary tree format. Each node stores Z data blocks, referred to as *buckets* of size Z. Additionally, the client has a small amount of local storage called the *stash*. The client maintains a secret mapping called the *position map* to keep track of the path where a data block is stored on the server. Each entry in the position map maps a client address to a leaf on the server. Path ORAM maintains the invariant that every block is stored somewhere along the path from the root to the leaf node that the block is mapped to by the position map.

Deniability of Server Access Patterns in Path ORAM: We formulate security of access patterns in Path ORAM as a deniability property stating that for every infinitely-long trace of server accesses, there are $(\mathsf{numBlks}-1)!$ traces of client accesses with identical server observations but different client requests.

$$\forall \pi_0.\ \#\pi_1 : \mathbf{F}\,(\delta_{\pi_j,\pi_k}).\ \mathbf{G}\,(\psi_{\pi_0,\pi_1})\ \geq\ (\mathsf{numBlks}-1)! \tag{6}$$

The binary predicate δ imposes the requirement that the client's request are different in each of the traces captured by the counting quantifier, and the condition in ψ states that all the traces captured by the counting quantifier have the same observable access pattern as π_0.

Verification of Deniability in Path ORAM: To verify the QHP stated in Equation 6, for every trace of server accesses we need to generate $(\mathsf{numBlks}-1)!$ traces of client requests that produce the same server access.

Suppose we have Path ORAM (a) that is initialized with some position map. Now consider the Path ORAM (b) with the same number of blocks, but with an initial position map that is a derangement of the position map of (a).[8] The key insight is that ORAM (b) can simulate an identical server access pattern as ORAM (a) by appropriately choosing a different client request that maps to the same leaf that is being accessed by (a) and then updating the position map identically as (a). This is shown in Figure 7, which shows two Path ORAMs that produce identical server access patterns but service different client requests.

The above insight leads to a trace enumeration where two traces are related via $\mathcal{U}$ if their position maps are derangements of each other, the client accesses are permuted as per the derangement while all other parameters of the ORAM are identical. We use this to prove Property 6. Further details are given in [43].

7 Related Work

Hyperproperties: Research into secure information flow started with the seminal work of Denning and Denning [24], Goguen and Meseguer [32] and Rushby [42].

[8] A **derangement** of a set is a permutation of the elements of the set such that no element appears in its original position.

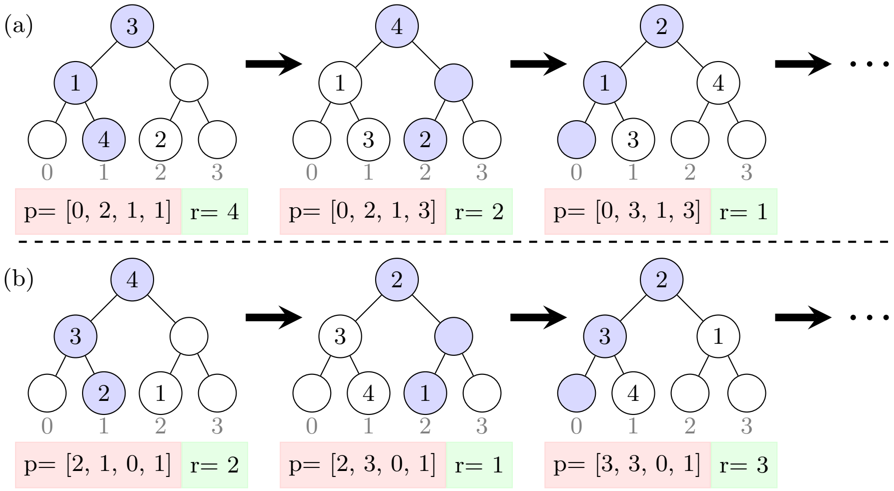


Fig. 7: Path ORAMs satisfying the counting quantifier of Equation 6, where, p represents the position map indexed from 1 and r is the client's request.

The self-composition construction for the verification of secure information flow was introduced by Barthe et al. [10]. Clarkson and Schneider [21] introduced the class of specifications called hyperproperties. Clarkson and colleagues also introduced HyperLTL and HyperCTL* [19], which are temporal logics for specifying hyperproperties, while verification algorithms for these were introduced by Finkbeiner and colleagues in [30]. Cartesian Hoare Logic [47] was introduced by Sousa and Dillig and enables the specification and verification of hyperproperties over programs as opposed to transition systems. A number of subsequent efforts have studied hyperproperties in the context of program verification [5, 26, 45, 53].

Quantitative Information Flow: Quantitative hyperproperties build on the rich literature of quantitative information flow (QIF) [3, 17, 20, 34, 46]. The QIF problem is to quantify (or bound) the number of bits of secret information that is attacker-observable. Certain notions of QIF can be expressed as QHPs. It is important to note QHPs can express security specifications (e.g., soundness) that are not QIF. Yasuoka and Terauchi studied QIF from a theoretical perspective and showed that it could be expressed as hypersafety and hyperliveness [54]. Approaches based on QIF measures such as min-entropy [46], Shannon entropy [18] etc. have also been applied in the context of static analysis [38].

Quantitative Hyperproperties: Quantitative Cartesian Hoare Logic (QCHL) enables verification of certain quantitative properties of programs [16]. QHPs are more expressive than QCHL, the latter counts events within a trace (e.g. memory accesses), while QHPs count the number of traces satisfying certain conditions.

The most closely related work to ours is of Finkbeiner et al. [29] who introduced Quantitative HyperLTL over Kripke structures. They also introduced a

verification algorithm for this logic that is based on maximum model counting. However, their algorithm does not scale to reasonable-sized systems, and experiments from their paper show that the approach times out when checking an 8-bit leak in a password checker (using 8-bit passwords). We differ from their work in three important ways. First, our properties are defined over symbolic transition systems rather than Kripke structures. This allows modeling and verification of QHPs over infinite-state systems. Second, our bounds are symbolic, which enables us to express bounds as functions of transition system parameters. Finally, our definition of Quantitative HyperLTL is also more expressive. It is not possible to convert our QHPs into (non-quantitative) HyperLTL formulas with k-traces for any fixed value of k.
Verification of ORAMs: In concurrent work with ours, Barthe et al. [11] and Darais et al. [22] have introduced specialized mechanisms to prove security of ORAMs. Barthe et al. [11] introduced a probabilistic separation logic (PSL) that (among other things) can be used to reason about the security of ORAMs. Unlike QHPs, PSL does not permit quantitative reasoning about probabilities of events and also does not (yet) support machine-checked reasoning. Darais et al. [22] introduce a type system that enforces obliviousness; they use this type system to implement a tree-based ORAM. Note that QHPs can express specifications other than obliviousness, and obliviousness need not necessarily be a QHP.

8 Conclusion

Quantitative hyperproperties are a powerful class of specifications that stipulate the existence of a certain number of traces satisfying certain constraints. Many important security guarantees, especially those involving probabilistic guarantees of security, can be expressed as quantitative hyperproperties. Unfortunately, verification of quantitative hyperproperties is a challenging problem because these specifications require simultaneous reasoning about a large number of traces of a system. In this paper, we introduced a specification language, satisfaction semantics, and a verification methodology for quantitative hyperproperties. Our verification methodology is based on reducing the problem of counting traces into that of counting the number of assignments that satisfy a first-order logic formula. Our methodology enables security verification of many interesting security protocols that were previously out of reach, including confidentiality of access pattern accesses in Path ORAM.

Acknowledgements

We sincerely thank the anonymous reviewers for their insightful comments, which helped improve this paper. This work was supported in part by the Semiconductor Research Corporation under Task 2854 and the Science and Engineering Research Board of India, a unit of the Department of Science and Technology, Government of India.

Bibliography

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. Verifiable side-channel security of cryptographic implementations: constant-time mee-cbc. In *International Conference on Fast Software Encryption*, pages 163–184. Springer, 2016.

[2] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *25th USENIX Security Symposium, USENIX Security.*, pages 53–70, 2016.

[3] Mário S Alvim, Miguel E Andrés, and Catuscia Palamidessi. Quantitative information flow in interactive systems. *Journal of Computer Security*, 20(1):3–50, 2012.

[4] Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. Decomposition instead of self-composition for proving the absence of timing channels. In *PLDI*, pages 362–375, 2017.

[5] Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. Decomposition instead of self-composition for proving the absence of timing channels. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 362–375, New York, NY, USA, 2017. ACM.

[6] J. Bacelar Almeida, Manuel Barbosa, Jorge S. Pinto, and Bárbara Vieira. Formal verification of side-channel countermeasures using self-composition. *Sci. Comput. Program.*, 78(7):796–812, July 2013.

[7] Michael Backes, Boris Kopf, and Andrey Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 141–153, Washington, DC, USA, 2009. IEEE Computer Society.

[8] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. System-level non-interference for constant-time cryptography. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1267–1279. ACM, 2014.

[9] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational verification using product programs. In Michael Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods*, pages 200–214, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17)*, pages 100–114, 2004.

[11] Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proc. ACM Program. Lang.*, 4(POPL), December 2019.

[12] Vincent Bindschaedler, Reza Shokri, and Carl A Gunter. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment*, 10(5):481–492, 2017.

[13] Anders Björner and Richard P. Stanley. *A Combinatorial Miscellany.* L'Enseignement mathématique, 2010.

[14] Anrin Chakraborti, Chen Chen, and Radu Sion. Datalair: Efficient block storage with plausible deniability against multi-snapshot adversaries. *Proceedings on Privacy Enhancing Technologies*, 2017(3):179–197, 2017.

[15] Kevin Cheang, Cameron Rasmussen, Sanjit Seshia, and Pramod Subramanyan. A formal approach to secure speculation. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 288–28815, June 2019.

[16] Jia Chen, Yu Feng, and Isil Dillig. Precise detection of side-channel vulnerabilities using quantitative cartesian hoare logic. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 875–890, New York, NY, USA, 2017. ACM.

[17] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation*, 15(2):181–199, 2005.

[18] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[19] Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust*, pages 265–284. Springer, 2014.

[20] Michael R Clarkson, Andrew C Myers, and Fred B Schneider. Belief in information flow. In *18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 31–45. IEEE, 2005.

[21] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[22] David Darais, Ian Sweet, Chang Liu, and Michael Hicks. A language for probabilistically oblivious computation. *Proc. ACM Program. Lang.*, 4(POPL), December 2019.

[23] L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[24] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, July 1977.

[25] Experiments: Models and Proof Scripts for the paper "Verification of Quantitative Hyperproperties Using Trace Enumeration Relations". Available at https://github.com/ssahai/CAV-2020-benchmarks, 2020.

[26] Azadeh Farzan and Anthony Vandikas. Automated hypersafety verification. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, pages 200–218, 2019.

[27] Grigory Fedyukovich, Sumanth Prabhu, Kumar Madhukar, and Aarti Gupta. Quantified invariants via syntax-guided synthesis. In *International Conference on Computer Aided Verification*, pages 259–277. Springer, 2019.

[28] Andrew Ferraiuolo, Rui Xu, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, pages 555–568, 2017.

[29] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. Model checking quantitative hyperproperties. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 144–163, 2018.

[30] Bernd Finkbeiner, Markus N Rabe, and César Sánchez. Algorithms for model checking hyperltl and hyperctl *. In *International Conference on Computer Aided Verification*, pages 30–48. Springer, 2015.

[31] Daniel J Fremont, Markus N Rabe, and Sanjit A Seshia. Maximum model counting. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[32] Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20, 1982.

[33] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[34] James W Gray III. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, 1(3-4):255–294, 1992.

[35] Marco Guarnieri, Boris Köpf, José F. Morales, Jan Reineke, and Andrés Sánchez. SPECTECTOR: principled detection of speculative information flows. *CoRR*, abs/1812.08639, 2018.

[36] Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. Quantifiers on demand. In *International Symposium on Automated Technology for Verification and Analysis*, pages 248–266. Springer, 2018.

[37] Chris Hawblitzel, Jon Howell, Jacob R Lorch, Arjun Narayan, Bryan Parno, Danfeng Zhang, and Brian Zill. Ironclad Apps: End-to-End Security via Automated Full-System Verification. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 165–181, 2014.

[38] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In *International Conference on Computer Aided Verification*, pages 564–580. Springer, 2012.

[39] Sujit Kumar Muduli, Pramod Subramanyan, and Sayak Ray. Verification of Authenticated Firmware Loaders. In *Proceedings of Formal Methods in Computer-Aided Design*. IEEE, 2019.

[40] Ling Ren, Christopher Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious RAM. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 415–430, Washington, D.C., August 2015. USENIX Association.

[41] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 114–127, 1995.

[42] John M. Rushby. Proof of separability: A verification technique for a class of a security kernels. In *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, pages 352–367, 1982.

[43] Shubham Sahai, Pramod Subramanyan, and Rohit Sinha. Verification of Quantitative Hyperproperties Using Trace Enumeration Relations. *arXiv e-prints*, abs/2005.04606, May 2020.

[44] Sanjit A. Seshia and Pramod Subramanyan. UCLID5: Integrating modeling, verification, synthesis and learning. In *Proceedings of the 16th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, October 2018.

[45] Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. Property directed self composition. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 161–179, Cham, 2019. Springer International Publishing.

[46] Geoffrey Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, pages 288–302. Springer, 2009.

[47] Marcelo Sousa and Isil Dillig. Cartesian Hoare Logic for verifying k-safety properties. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 57–69, New York, NY, USA, 2016. ACM.

[48] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310, 2013.

[49] Pramod Subramanyan, Rohit Sinha, Ilia A. Lebedev, Srinivas Devadas, and Sanjit A. Seshia. A formal foundation for secure remote execution of enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2435–2450, 2017.

[50] Tachio Terauchi and Alexander Aiken. Secure information flow as a safety problem. In *Static Analysis, 12th International Symposium, SAS, Proceedings*, pages 352–367, 2005.

[51] UCLID5 Verification and Synthesis System. Available at http://github.com/uclid-org/uclid/, 2019.

[52] Herbert S. Wilf. *generatingfunctionology*. AK Peters/CRC Press, 2005.

[53] Weikun Yang, Pramod Subramanyan, Yakir Vizel, Aarti Gupta, and Sharad Malik. Lazy Self-Composition for Security Verification. In *Computer Aided Verification - 30th International Conference, CAV 2018, Oxford, UK, July 14-17, 2018, Proceedings*, 2018.

[54] Hirotoshi Yasuoka and Tachio Terauchi. Quantitative information flow as safety and liveness hyperproperties. *Theoretical Computer Science*, 538:167–182, 2014.

[55] Steve Zdancewic and Andrew C Myers. Observational determinism for concurrent program security. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 29–43. IEEE, 2003.

[56] Doron Zeilberger. Enumerative and algebraic combinatorics. In *The Princeton Companion to Mathematics*, pages 550–561. Princeton University Press, 2010.

[57] Danfeng Zhang, Yao Wang, G. Edward Suh, and Andrew C. Myers. A Hardware Design Language for Timing-Sensitive Information-Flow Security. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*, pages 503–516, 2015.