Compositional closure for Bayes Risk in probabilistic noninterference

Annabelle McIver[1], Larissa Meinicke[1], and Carroll Morgan[2⋆]

[1] Dept. Computer Science, Macquarie University, NSW 2109 Australia
[2] School of Comp. Sci. and Eng., Univ. New South Wales, NSW 2052 Australia

Abstract. We give a quantitative sequential model for noninterference security with probability (but not demonic choice), and a novel *refinement* order that we prove to be the *greatest* compositional relation consistent with an "elementary" order based on Bayes Risk. This *compositional closure* complements our earlier work defining refinement similarly for qualitative noninterference with demonic choice (but not probability).

The *Three-Judges Protocol* illustrates our model's utility: with compositionality, the embedded sub-protocols can be treated in isolation.

1 Introduction

Noninterference analysis splits a state space into high- and low-security portions and determines whether high-security values can be inferred from observation of low-security values and behaviours [5]. If **probability** is present, however, the question becomes "how much" rather than "whether" — and thus in that case we address the *likelihood* with which high-security values can be inferred from low-security observations. Here, we *compare* programs for that likelihood.

For programs S, I we say that $S \preceq I$ if S and I are functionally equivalent and the **Bayes Risk** (defined below) associated with implementation I is no worse than the Risk associated with specification S. If for all $S \preceq I$ and context $\mathcal{C}$ we had $\mathcal{C}(S) \preceq \mathcal{C}(I)$, then we would say that $(\preceq)$ is compositional.

Our TECHNICAL CONTRIBUTION is that we (i) give a sequential semantics for probabilistic noninterference, (ii) define the above "less likely to leak" order $(\preceq)$ based on Bayes Risk, (iii) show it is *not* compositional, (iv) identify a **compositional** subset of it, a *refinement* order $(\sqsubseteq)$ such that $S \sqsubseteq I$ implies $\mathcal{C}(S) \preceq \mathcal{C}(I)$ for all contexts $\mathcal{C}$ and (v) show that $(\sqsubseteq)$ is in fact the compositional **closure** of $(\preceq)$, so that we have $S \not\sqsubseteq I$ *only* when $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$ for some $\mathcal{C}$.

Our GENERAL CONTRIBUTION is to further the goal of structuring secure protocols hierarchically and then designing/verifying them in separate pieces.

2 A probabilistic, noninterference sequential semantics

We distinguish *visible* variables (low-security), typically v in some finite type $\mathcal{V}$, and *hidden* variables (high-security), typically h in finite $\mathcal{H}$. Variables are in sans serif to avoid confusion with (decorated) values $v{:}\,\mathcal{V}, h{:}\,\mathcal{H}$ they might contain.

⋆ We acknowledge the support of the Australian Research Council Grant DP0879529.

For example, let $\mathsf{h}\colon\{0,1,2\}$ represent one of three boxes: Box 0 has no white balls and two black; Box 1 has one of each; and Box 2 has two white. Let $\mathsf{v}\colon\{w,b,\bot\}$ represent a ball colour: *white*, *black*, *unknown*. Program S, informally written $\mathsf{h}{:=}\,0{\oplus}1{\oplus}2; \mathsf{v}{:\in}\,\{\!|w^{@\frac{\mathsf{h}}{2}}, b^{@1-\frac{\mathsf{h}}{2}}|\!\}; \mathsf{v}{:=}\,\bot$ chooses box h uniformly, then draws a ball v from that box: it has probability $\mathsf{h}/2$ of being white, and $1{-}\mathsf{h}/2$ of being black. Finally it replaces the ball. A typical security concern is then "How much information about h was revealed by the assignments to v?"

Below we give syntax and semantics to make the above program precise, providing the framework for asking –and answering– such security questions.

First however we introduce *distribution* notation, generalising set notations.

2.1 Distributions: explicit, implicit and expected values over them

We write function application as $f.x$, with "." associating to the left. Operators without their operands are written between parentheses, as $(\preceq)$ for example.

By $\mathbb{D}X$ we mean the set of *discrete sub-distributions* on set X that sum to no more than one, and by $\mathsf{D}X$ we mean the *full* distributions that sum to one exactly. The *support* $\lceil\delta\rceil$ of (sub-)distribution $\delta\colon\mathbb{D}X$ is those elements x in X with $\delta.x{\neq}0$. Distributions can be scaled and summed according to the usual pointwise extension of multiplication and addition to real-valued functions.

Here are our notations for *explicit* distributions (cf. set enumerations):

multiple We write $\{\!|x^{@p}, y^{@q}, \cdots, z^{@r}|\!\}$ for the distribution assigning probabilities $p, q, \cdots, r$ to elements $x, y, \cdots, z$ respectively, with $p{+}q{+}\cdots{+}r \leq 1$.

uniform When explicit probabilities are omitted they are uniform: thus $\{\!|x|\!\}$ is the point distribution $\{\!|x^{@1}|\!\}$, and $\{\!|x,y,z|\!\}$ is $\{\!|x^{@\frac{1}{3}}, y^{@\frac{1}{3}}, z^{@\frac{1}{3}}|\!\}$.

We write $(\odot\, d\colon\delta \bullet E)$ for the *expected value* $\sum_{d:\,\lceil\delta\rceil}(\delta.d * E)$ of expression E interpreted as a random variable in d over distribution δ. [3] If however E is Boolean, then it is taken to be 1 if E holds and 0 otherwise, so that the expected value is then just the combined probability in δ of all elements d satisfying E.

We write *implicit* distributions (cf. set comprehensions) as $\{\!|d\colon\delta \mid R \bullet E|\!\}$, for distribution δ, real expression R and expression E, meaning

$$(\odot\, d\colon\delta \bullet R * \{\!|E|\!\}) \;/\; (\odot\, d\colon\delta \bullet R) \tag{1}$$

where, first, an expected value is formed in the numerator by scaling and adding point-distribution $\{\!|E|\!\}$ as a real-valued function: this gives another distribution. The scalar denominator then normalises to give a distribution yet again. A missing E is implicitly d itself; and missing R is implicitly 1, effectively removing the denominator if δ is full (i.e. sums to one).

For example $\{\!|d\colon\delta \bullet E|\!\}$ *maps* expression E in d over distribution δ to make a new distribution on E's type. And a Boolean R is converted to 0,1; in that case $\{\!|d\colon\delta \mid R|\!\}$ is δ's *conditioning* over formula R as predicate on d, if δ is full.

[3] It is a dot-product between the distribution and the random variable as vectors.

Finally, for *Bayesian belief revision* let δ be an a-priori distribution over some D and let expression R for each d in D be the probability of a certain subsequent result if that d is chosen. Then $\{\!\{d{:}\,\delta \mid R\}\!\}$ is the a-posteriori distribution over D when that result actually occurs. Thus in the three-box program S let the value first assigned to v be $\hat{v}$. The a-priori distribution δ over h is uniform, and the probability of $\hat{v}{=}w$ is therefore $1/3{*}(0/2{+}1/2{+}2/2) = 1/2$. But the a-posteriori distribution of h *given that* $\hat{v}{=}w$ is $\{\!\{h{:}\,\delta \mid h/2\}\!\}$, which from (1) we calculate

$$= \quad (\odot\, h{:}\,\{\!\{0,1,2\}\!\} \bullet \frac{h}{2} * \{\!\{h\}\!\}) \;/\; (\odot\, h{:}\,\{\!\{0,1,2\}\!\} \bullet \frac{h}{2}) \quad = \quad \{\!\{1^{@\frac{1}{6}}, 2^{@\frac{1}{3}}\}\!\} \;/\; \frac{1}{2}\ ,$$

that is $\{\!\{1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}\}\!\}$. Thus if a white ball is drawn ($\hat{v}{=}w$) then the chance it came from Box 2 is $2/3$, the probability of $\mathsf{h}{=}2$ in the a-posteriori distribution.

2.2 Program denotations over a visible/hidden "split" state-space

With *visible* and *hidden* partitioning of the finite state space $\mathcal{V}{\times}\mathcal{H}$, probabilistic states in our new model are *split-states* of type $\mathcal{V}{\times}\mathsf{D}\mathcal{H}$, whose typical element (v,δ) indicates that we know $\mathsf{v}{=}v$ exactly, but all we know about h –which is not directly observable– is that it takes value h with probability $\delta.h$.

Programs are functions $(\mathcal{V}{\times}\mathsf{D}\mathcal{H}) \to \mathsf{D}(\mathcal{V}{\times}\mathsf{D}\mathcal{H})$ from split-states to distributions over them, called *hyper-distributions* since they are distributions with other distributions inside them: the outer distribution is directly visible but the inner distribution(s) over $\mathcal{H}$ are not. Thus for a program P, the application $[\![P]\!].(v,\delta)$ is the distribution of final split-states produced from initial (v,δ). Each (v',δ') in that outcome, with probability p say in the outer- (left-hand) D in $\mathsf{D}(\mathcal{V}{\times}\mathsf{D}\mathcal{H})$, means that with probability p an attacker will observe that v is v' and simultaneously be able to deduce that h has distribution δ'.

2.3 Program syntax and semantics

The programming language syntax and semantics is given in Fig. 1. Assignments to v or h can use an expression $E.\mathsf{v}.\mathsf{h}$ or a distribution $D.\mathsf{v}.\mathsf{h}$; and assignments to v might reveal information about h. For example, we have that

(i) A direct assignment of h to v reveals everything about h:
$$[\![\mathsf{v}{:=}\,\mathsf{h}]\!].(v,\delta) = \{\!\{h{:}\,\delta \bullet (h, \{\!\{h\}\!\})\}\!\}$$
(ii) Choosing v from a distribution independent of h reveals nothing about h:
$$[\![\mathsf{v}{:=}\,0\ {}_{1/3}\!\oplus\ 1]\!].(v,\delta) = \{\!\{(0,\delta)^{@\frac{1}{3}}, (1,\delta)^{@\frac{2}{3}}\}\!\}$$
(iii) Partially h-dependent assignments to v might reveal something about h:
$$[\![\mathsf{v}{:=}\,\mathsf{h}\ \mathbf{mod}\ 2]\!].(v,\{\!\{0,1,2\}\!\}) = \{\!\{(0,\{\!\{0,2\}\!\})^{@\frac{2}{3}}, (1,\{\!\{1\}\!\})^{@\frac{1}{3}}\}\!\}$$

On the other hand, assignments to h affect δ directly. Thus choosing h might

(i) increase our uncertainty of h: $[\![\mathsf{h}{:=}\,0{\oplus}1{\oplus}2]\!].(v,\{\!\{0,1\}\!\}) = \{\!\{(v,\{\!\{0,1,2\}\!\})\}\!\}$
(ii) or reduce it: $[\![\mathsf{h}{:=}\,0{\oplus}1]\!].(v,\{\!\{0,1,2\}\!\}) = \{\!\{(v,\{\!\{0,1\}\!\})\}\!\}$
(iii) or leave it unchanged: $[\![\mathsf{h}{:=}\,2{-}\mathsf{h}]\!].(v,\{\!\{0,1,2\}\!\}) = \{\!\{(v,\{\!\{2,1,0\}\!\})\}\!\}$

Program type	Program text P	Semantics $[\![P]\!].(v,\delta)$
Identity	**skip**	$\{\!\{\ (v,\delta)\ \}\!\}$
Assign to visible	$\mathsf{v}{:=}\,E.\mathsf{v}.\mathsf{h}$	$\{\!\{\ h{:}\,\delta \bullet (E.v.h, \{\!\{h'{:}\,\delta \mid E.v.h'{=}E.v.h\}\!\})\ \}\!\}$
Assign to hidden	$\mathsf{h}{:=}\,E.\mathsf{v}.\mathsf{h}$	$\{\!\{\ (v, \{\!\{h{:}\,\delta \bullet E.v.h\}\!\})\ \}\!\}$
Choose prob. visible	$\mathsf{v}{:\in}\,D.\mathsf{v}.\mathsf{h}$	$\{\!\{\ v'{:}\,(\odot\, h{:}\,\delta \bullet D.v.h) \bullet (v', \{\!\{h'{:}\,\delta \mid D.v.h'.v'\}\!\})\ \}\!\}$
Choose prob. hidden	$\mathsf{h}{:\in}\,D.\mathsf{v}.\mathsf{h}$	$\{\!\{\ (v, (\odot\, h{:}\,\delta \bullet D.v.h))\ \}\!\}$
Composition	$P_1; P_2$	$(\odot\,(v',\delta'){:}\,[\![P_1]\!].(v,\delta) \bullet\ [\![P_2]\!].(v',\delta'))$
Probabilistic choice	$P_1\ {}_p\!\oplus\ P_2$	$p * [\![P_1]\!].(v,\delta) + (1{-}p) * [\![P_2]\!].(v,\delta)$ $\quad p$ is constant
Conditional choice p is $(\odot\, h{:}\,\delta \bullet G.v.h)$	**if** $G.\mathsf{v}.\mathsf{h}$ **then** P_t **else** P_f **fi**	$p * [\![P_t]\!].(v, \{\!\{h{:}\,\delta \mid G.v.h\}\!\})$ $+\ (1{-}p) * [\![P_f]\!].(v, \{\!\{h{:}\,\delta \mid \neg G.v.h\}\!\})$

For simplicity let $\mathcal{V}$ and $\mathcal{H}$ have the same type $\mathcal{X}$. Expression $E.\mathsf{v}.\mathsf{h}$ is then of type $\mathcal{X}$, distribution $D.\mathsf{v}.\mathsf{h}$ is of type $\mathsf{D}\mathcal{X}$ and expression $G.\mathsf{v}.\mathsf{h}$ is Boolean. Also we assume here that the p in ${}_p\!\oplus$ is constant — but it can in general depend on v and h.

For distributions in *program texts* we allow the more familiar infix notation ${}_p\!\oplus$, so that we can write $\mathsf{h}{:=}\,0\,{}_{\frac{1}{3}}\!\oplus 1$ for $\mathsf{h}{:\in}\,\{\!\{0^{@\frac{1}{3}}, 1^{@\frac{2}{3}}\}\!\}$ and $\mathsf{h}{:=}\,0{\oplus}1$ for the uniform $\mathsf{h}{:\in}\,\{\!\{0,1\}\!\}$. The degenerate cases $\mathsf{h}{:=}\,0$ and $h{:\in}\,\{\!\{0\}\!\}$ are then equivalent, as they should be.

Fig. 1. Split-state semantics of commands

Assignment statements are "atomic" — an attacker may not directly witness the evaluation of the right-hand side. For instance, the atomic probabilistic choice $\mathsf{v}{:=}\,\mathsf{h}{\oplus}\neg\mathsf{h}$ does *not* reveal which of the equally likely operands of $(\oplus)$ was used. We show elsewhere [11] how atomicity can be controlled more generally.

2.4 A strong attacker: perfect recall, and implicit flow

Our definition $[\![P_1; P_2]\!]$ of *sequential composition* [4] gives an attacker **perfect recall** after P_2 of the visible variable v as it was after P_1. Compare Program S (above) with Program I_1 defined $\mathsf{h}{:=}\,0{\oplus}1{\oplus}2; \mathsf{v}{:=}\,\bot$ in which no ball is drawn: the final hyper-distributions are

$$\{\!\{\ (\bot, \{\!\{1^{@\frac{1}{3}}, 2^{@\frac{2}{3}}\}\!\}),\ (\bot, \{\!\{0^{@\frac{2}{3}}, 1^{@\frac{1}{3}}\}\!\})\ \}\!\} \qquad (\Delta'_S)$$

and

$$\{\!\{\ (\bot, \{\!\{0,1,2\}\!\})\ \}\!\}\ . \qquad (\Delta'_{I_1})$$

In neither case does the final value $\bot$ of v reveal anything about h. But Δ'_S, unlike Δ'_{I_1}, has separated pairs which recall implicitly the intermediate value $\hat{v}$ of v. Generally, if two final pairs (v',δ'_1) and (v',δ'_2) occur with $\delta'_1{\neq}\delta'_2$ then it means an attacker can deduce whether h's distribution is δ'_1 or δ'_2 even though v has the same final value v' in both cases. Although the direct evidence $\hat{v}$ has been overwritten, the separated pairs preserve the attacker's deductions from it.

The meaning of *probabilistic choice* $P_1\ {}_p\!\oplus\ P_2$ makes it behave like $[\![P_1]\!]$ with probability p and $[\![P_2]\!]$ with the remaining probability. (In Fig. 1 we gave the

[4] It is effectively the Kleisli composition over the outer distribution.

definition only for constant p; but in general p can be an expression over v, h.) The definition allows an attacker to observe which branch was taken: thus unlike (ii) above we have $[\![h{:=}0 \oplus h{:=}1]\!].(v,\delta) = \{\!\{(v, \{\!\{0\}\!\}), (v, \{\!\{1\}\!\})\}\!\}$, which is an example of **implicit flow**. *Conditional choice* has the same effect, revealing $G.\mathsf{v}.\mathsf{h}$.

Both perfect recall and implicit flow are built-in because, in our earlier "Shadow" semantics of noninterference and demonic- rather than probabilistic choice [13, 14], we found via program-algebraic *gedanken* experiments that compositionality (equivalently, monotonicity of refinement) required the adversary to be treated as if it had those two abilities.

3 The Bayes-Risk based elementary testing order

The elementary testing order has both traditional, functional characteristics and specialised, information-based security characteristics.

Say that two programs are *functionally equivalent* iff from the same input they produce the same *overall* output distribution [8, 12], defined for hyper-distribution Δ' to be $\mathsf{ft}.\Delta' := \{\!\{(v', \delta') : \Delta'; h' : \delta' \bullet (v', h')\}\!\}$. We consider state-space $\mathcal{V}{\times}\mathcal{H}$ jointly, i.e. not only $\mathcal{V}$, because differing distributions over h alone can be revealed by the context $(-\,;\, \mathsf{v}{:=}\mathsf{h})$ that appends an assignment $\mathsf{v}{:=}\mathsf{h}$.

We measure the *security* of a program with "Bayes Risk" [1–3, 18], an attacker's chance of guessing the final value of h in one try. The most effective such attack is to determine which split-state (v', δ') in a final hyper-distribution actually occurred, and then to guess that h has some value h' that maximises δ', i.e. so that $\delta'.h' = \sqcup\delta'$. For a whole hyper-distribution we average the attacks over its elements, weighted by the probability it gives to each, and so we we define the **Bayes Risk** of Δ' to be $\mathsf{br}.\Delta' := (\odot\,(v', \delta') : \Delta' \bullet \sqcup\delta')$.[5]

For Program S the Risk is the chance of guessing h by remembering v's intermediate value, say $\hat{v}$, and then guessing that h at that point had the value most likely to have produced that $\hat{v}$: when $\hat{v}{=}w$ (probability 1/2), guess $\mathsf{h}{=}2$; when $\hat{v}{=}b$, guess $\mathsf{h}{=}0$. Via $\mathsf{br}.\Delta'_S$ that Risk is $1/2{*}2/3 + 1/2{*}2/3 = 2/3$. For I_1 however, where there is no "leaking" $\hat{v}$, the Risk is the lower $\mathsf{br}.\Delta'_{I_1} = 1/3$.

The elementary testing order on (final) hyper-distributions is then defined so that $\Delta'_S \preceq \Delta'_I$ just when $\mathsf{ft}.\Delta'_S{=}\mathsf{ft}.\Delta'_I$ and $\mathsf{br}.\Delta'_S{\geq}\mathsf{br}.\Delta'_I$, and it extends pointwise to the **elementary testing order** on whole programs. That is, we say that $S \preceq I$ just when for corresponding inputs (i) S, I are functionally equivalent and (ii) the Risk for I is no more than the Risk for S. Thus $S \preceq I_1$ because they are functionally equivalent and the Risks of S, I_1 are $2/3, 1/3$ resp.

4 Non-compositionality of the elementary testing order

For stepwise development we require more than just $S \preceq I$: we must ensure that $\mathcal{C}(S) \preceq \mathcal{C}(I)$ holds for all contexts $\mathcal{C}(\cdot)$ in which S, I might be placed — and we do not know in advance what those contexts might be.

[5] "Greatest chance of leak" is more convenient than the dual (and more usual) "least chance of no leak." Our definition corresponds to *vulnerability* [18, for example].

Consider Program I_2 in which also Box 1 has two black balls, defined by the code $\mathsf{h}{:=}0{\oplus}1{\oplus}2; \mathsf{v}{:\in}\, \{\!|w^{@(\mathsf{h}\div 2)}, b^{@1-(\mathsf{h}\div 2)}|\!\}; \mathsf{v}{:=}\bot$ with final hyper-distribution

$$\{\!|\ (\bot, \{\!|2|\!\})^{@\frac{1}{3}}\ ,\ (\bot, \{\!|0,1|\!\})^{@\frac{2}{3}}\ |\!\}\ . \tag{Δ'_{I_2}}$$

The Risk for I_2 is $1/3{*}1 + 2/3{*}1/2$, again $2/3$ so that $S{\preceq}I_2$. Now if context $\mathcal{C}$ is defined $(-\,;\, \mathsf{h}{:=}\mathsf{h}\div 2)$, the Risk for $\mathcal{C}(S)$ is $1/2{*}2/3 + 1/2{*}1 = 5/6$: it is more than for S alone because there are fewer final h-values to choose from. But for $\mathcal{C}(I_2)$ it is greater still, at $1/3{*}1 + 2/3{*}1 = 1$.

Since $S{\preceq}I_2$ but $\mathcal{C}(S){\not\preceq}\mathcal{C}(I_2)$, the order $(\preceq)$ is not compositional (monotonic).

5 The refinement order, and compositional closure

To address the non-compositionality exposed just above, we seek the *compositional closure* of $(\preceq)$, the unique *refinement* relation $(\sqsubseteq)$ such that (*soundness*) if $S{\sqsubseteq}I$ then for all $\mathcal{C}$ we have $\mathcal{C}(S){\preceq}\mathcal{C}(I)$; and (*completeness*) if $S{\not\sqsubseteq}I$ then for some $\mathcal{C}$ we have $\mathcal{C}(S){\not\preceq}\mathcal{C}(I)$. Soundness gives refinement the property (refer §4) we need for stepwise development; and completeness makes refinement as liberal as possible consistent with that.

We found above that $S{\not\sqsubseteq}I_2$; we show later (§6.1) that we do have $S{\sqsubseteq}I_1$.

6 Constructive definition of the refinement order ($\sqsubseteq$)

We give a detailed example to help introduce our definition. Let $x\ \mathbf{rnd}\ n$ be a distribution over the multiple(s) of n closest to x, weighted inversely by their distance: thus $1\mathbf{rnd}4$ is $(0_{3/4}{\oplus}4)$ and $3\mathbf{rnd}4$ is $(0_{1/4}{\oplus}4)$; but $2\mathbf{rnd}4$ is $(0_{1/2}{\oplus}4)$. When x divides n exactly, the outcome is definite: thus $2\ \mathbf{rnd}\ 2 = \{\!|2|\!\}$. Now consider the two programs

$$\begin{array}{rll} & P_2{:=} & \mathsf{h}{:=}1{\oplus}2{\oplus}3;\ \mathsf{v}{:\in}\,\mathsf{h}\ \mathbf{rnd}\ 2;\ \mathsf{v}{:=}\mathsf{h}\ \mathbf{mod}\ 2 \\ \text{and} & P_4{:=} & \mathsf{h}{:=}1{\oplus}2{\oplus}3;\ \mathsf{v}{:\in}\,\mathsf{h}\ \mathbf{rnd}\ 4;\ \mathsf{v}{:=}\mathsf{h}\ \mathbf{mod}\ 2\ . \end{array} \tag{2}$$

Both reveal $\mathsf{h}\,\mathbf{mod}\,2$ in v's final value v', but each P_n also reveals in the overwritten visible $\hat{v}$, say, something about $\mathsf{h}\ \mathbf{rnd}\ n$; and intuition suggests that $P_n{\sqsubseteq}P_m$ for $n{\leq}m$ only. Yet the Risk is $5/6$ for both $P_{2,4}$, which we can see from the final hyper-distributions; they are

$$\{\!|\ \ (0, \{\!|2|\!\})^{@\frac{1}{3}},\ (1, \{\!|1|\!\})^{@\frac{1}{6}}, (1, \{\!|1,3|\!\})^{@\frac{1}{3}}, (1, \{\!|3|\!\})^{@\frac{1}{6}}\ \ |\!\} \tag{Δ'_{P_2}}$$

$$\{\!|\ \ (0, \{\!|2|\!\})^{@\frac{1}{3}}, \underbrace{(1, \{\!|1^{@\frac{3}{4}}, 3^{@\frac{1}{4}}|\!\})^{@\frac{1}{3}}}, (1, \{\!|1^{@\frac{1}{4}}, 3^{@\frac{3}{4}}|\!\})^{@\frac{1}{3}}\ \ |\!\} \tag{Δ'_{P_4}}$$

With overall probability $1/3{*}3/4 + 1/3{*}1/4 = 1/3$ the final v' will be 1 and $\hat{v}$ will be 0; since v' is 1 then h must be 1 or 3; but if $\hat{v}$ was 0 that h is three times as likely to have been 1.

so that e.g. indeed $1/3{*}1 + 1/3{*}3/4 + 1/3{*}3/4 = 5/6$ for P_4. The overall distribution of (v', h') is $\{\!|(0,2),(1,1),(1,3)|\!\}$ in both cases, so that $P_{2,4}$ are functionally equivalent; but they have different residual uncertainties of h.

Now we consider just the h-distributions associated with $v'{=}1$ and, by multiplying through their associated probabilities from the hyper-distributions, present

them as a collection of *fractions*, sub-distributions over $\mathcal{H}$. We call such collections *partitions* and here they are given for P_2 and P_4 respectively by

$$\text{when } v'{=}1 \quad \begin{cases} \Pi'_{P_2}: & \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}, \{\!\{ 3^{@\frac{1}{6}} \}\!\} \rangle \\ \Pi'_{P_4}: & \langle \{\!\{ 1^{@\frac{1}{4}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{4}} \}\!\} \rangle \ . \end{cases} \tag{3}$$

Fractions can be summed, so that e.g. $\{\!\{ 1^{@\frac{1}{6}} \}\!\} + \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}$ gives $\{\!\{ 1^{@\frac{1}{3}}, 3^{@\frac{1}{6}} \}\!\}$.

6.1 Constructive definition of refinement

Let the function $\mathsf{fracs}.\Delta.v$ for hyper-distribution Δ and value v give the partition of fractions extracted from Δ for $\mathsf{v}{=}v$. Say that one partition is *finer* than another if its fractions can be summed in groups to realise the *coarser* one. It is *sim-finer* if only fractions that are multiples of each other are summed.

Definition 1. *Secure refinement* We say that hyper-distribution Δ_S is secure-refined by Δ_I, written $\Delta_S \sqsubseteq \Delta_I$, just when for every v there is some intermediate partition Π of fractions so that both (i) Π is sim-finer than $\mathsf{fracs}.\Delta_S.v$ and (ii) Π is finer than $\mathsf{fracs}.\Delta_I.v$.[6] Refinement of hyper-distributions extends pointwise to the programs that produce them. □

Note that the (sim-)finer relation preserves the sum of *all* a partition's fractions, so that ($\sqsubseteq$) from Def. 1 implies functional equality.

Referring to Δ'_S, we get $\mathsf{fracs}.\Delta'_S.\bot = \langle \{\!\{ 1^{@\frac{1}{6}}, 2^{@\frac{1}{3}} \}\!\}, \{\!\{ 0^{@\frac{1}{3}}, 1^{@\frac{1}{6}} \}\!\} \rangle$ by multiplying through, and $\mathsf{fracs}.\Delta'_{I_1}.\bot = \langle \{\!\{ 0^{@\frac{1}{3}}, 1^{@\frac{1}{3}}, 2^{@\frac{1}{3}} \}\!\} \rangle$. The two fractions of the former sum to the single fraction of the latter, and so $S \sqsubseteq I_1$ according to our definition Def. 1 of secure refinement.

For the more detailed $\Delta'_{P_2} \sqsubseteq \Delta'_{P_4}$ and $v'{=}1$, we need the intermediate partition $\Pi := \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 3^{@\frac{1}{6}} \}\!\} \rangle$, whose middle two fractions turn out to be equal: summing them gives the middle $\{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}$ of Π'_{P_2}. Summing the first two gives $\{\!\{ 1^{@\frac{1}{4}}, 3^{@\frac{1}{12}} \}\!\}$, the first fraction of Π'_{P_4}; and the last two give the second fraction of Π'_{P_4}. Partition $\langle \{\!\{ 2^{@\frac{1}{3}} \}\!\} \rangle$ deals trivially with $v'{=}0$, and so indeed $P_2 \sqsubseteq P_4$. In §7.2 and in §B we show however that $P_4 \not\sqsubseteq P_2$.

Refinement ($\sqsubseteq$) is a partial order, and our programs preserve it [11]:

Lemma 1. *Monotonicity of refinement.* If $S \sqsubseteq I$ then $\mathcal{C}(S) \sqsubseteq \mathcal{C}(I)$ for all contexts $\mathcal{C}$ built from programs as defined in Fig. 1. □

7 Refinement ($\sqsubseteq$) is the compositional closure of ($\preceq$)

7.1 Soundness (sketch)

Our approach shows first that $\Delta_S \sqsubseteq \Delta_I$ implies $\Delta_S \preceq \Delta_I$, using arithmetic properties of maximum, and the addition and multiplication by non-negative scalars that occur in the operations (i,ii) of Def. 1. Since $S \sqsubseteq I$ implies $\mathcal{C}(S) \sqsubseteq \mathcal{C}(I)$ from Lem. 1, the above gives $\mathcal{C}(S) \preceq \mathcal{C}(I)$ as required. The full proof is in §A.1.

[6] In our earlier *qualitative* work [14] refinement reduces to taking unions of equivalence classes of hidden values, so-called "Shadows." Köpf et al. observe similar effects [7].

7.2 Completeness (sketch)

Here from $S \not\sqsupseteq I$ we must discover a context $\mathcal{C}$ such that $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$. This is done by reformulating the set of *all* refinements of specification S as a convex subset X_S of (higher-dimensional) Euclidean space, and the implementation I as a single point x_I in that space: the refinement-failure then becomes $x_I \notin X_S$.

The *Separating Hyperplane Lemma* [19] then gives us a hyperplane having all of X_S on one side and x_I on the other, and the coefficients of its normal M determine the parameters of a context $(-\,;C)$ that assigns conditionally to h and distinguishes x_I from all x_S's in X_S. For example, if we take C to be

$$\begin{array}{l}\textbf{if } \mathsf{v}{=}1 \textbf{ then } \mathsf{h}{:\in}\ (\,\{\!|1^{@\frac{1}{2}}, 2^{@\frac{1}{4}}, 3^{@\frac{1}{4}}|\!\}\ \textbf{if } \mathsf{h}{=}1 \textbf{ else } \{\!|2^{@\frac{1}{2}}, 3^{@\frac{1}{2}}|\!\}\,) \\ \qquad\textbf{else } \ \mathsf{h}{:=}1 \textbf{ fi}\ ,\end{array} \tag{4}$$

whose probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}$ and $0, \frac{1}{2}, \frac{1}{2}$ come from just such an M (Fig. 4 in §B), then we can concentrate on $v'{=}1$ and use (3,4) to give the partitions

$$v'{=}1 \left\{ \begin{array}{ll} \Pi'_{P_2;C}\colon & \langle \{\!|1^{@\frac{1}{12}}, 2^{@\frac{1}{24}}, 3^{@\frac{1}{24}}|\!\}, \{\!|1^{@\frac{1}{12}}, 2^{@\frac{1}{8}}, 3^{@\frac{1}{8}}|\!\}, \{\!|2^{@\frac{1}{12}}, 3^{@\frac{1}{12}}|\!\}\rangle \\ \Pi'_{P_4;C}\colon & \langle \{\!|1^{@\frac{1}{8}}, 2^{@\frac{5}{48}}, 3^{@\frac{5}{48}}|\!\}, \{\!|1^{@\frac{1}{24}}, 2^{@\frac{7}{48}}, 1^{@\frac{7}{48}}|\!\}\rangle\ . \end{array} \right.$$

Now the Risk of a partition on its own is obtained by simple summing of maxima, since its constituent fractions have been scaled already: thus at (3) above for Π'_{P_2} we get $1/6{+}1/6{+}1/6 = 1/2$, and for Π'_{P_4} we get $1/4{+}1/4 = 1/2$, the same. But for partition $\Pi'_{P_2;C}$ we get $1/12{+}1/8{+}1/12 = 7/24$; and for $\Pi'_{P_4;C}$ we get $1/8{+}7/48 = 13/48$, just smaller. Since the other partition (for $v'{=}0$) is $\langle \{\!|1^{@\frac{1}{3}}|\!\}\rangle$ in both cases, we establish $(P_4;C) \not\preceq (P_2;C)$. The full proof is in §A.2.

8 Case study: The *Three Judges* protocol

The **motivation** for our case study is to suggest and illustrate techniques for reasoning compositionally from specification to implementation of noninterference [6]. Our previous examples include (unboundedly many) Dining Cryptographers [4, 6, `Morgan:06`], Oblivious Transfer [16, 14] and Multi-Party Shared Computation [20, 13]. All of them however used our *qualitative* model for compositional noninterference [14, 13, 6, `Morgan:06`]; here we have instead a *quantitative* model.

To demonstrate the utility of our approach, we invented the following example. Three judges A, B, C are to give a majority decision, innocent or guilty, by exchanging messages but concealing their individual votes.[7] We describe this protocol with a program fragment, a **specification** which captures exactly the functional and security properties we want. Its variables are Boolean, equivalently $\{0,1\}$ and, including some notational conventions explained below, it evaluates $(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)$ atomically, and assigns it to visible j:

$$\begin{array}{ll} \textbf{vis } \mathsf{j};\ \textbf{vis}_A\ \mathsf{a};\ \textbf{vis}_B\ \mathsf{b};\ \textbf{vis}_C\ \mathsf{c}; & \leftarrow \text{These are global variables.} \\ \quad \mathsf{j}{:=}(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)\ . & \leftarrow \text{Atomic assignment.} \end{array} \tag{5}$$

[7] The generalised dining cryptographers would instead reveal the *actual number* of guilty votes. Here we reveal only the majority, a more difficult job [6, `Morgan:09a`].

We borrow features introduced in our earlier case studies to extend the reach of our technique without essentially changing its character, a sort of "syntactic sugaring." They include a convention –used in (5) above– that allows us to treat multiple-observer systems: annotation **vis** declares a variable visible to all observers, whereas e.g. $\mathbf{vis}_A$ declares visiblity to the observer "Judge A" only. Furthermore, we allow multiple variables so that, in terms of Fig. 1, for A's view we would have (j,a) as v jointly and (b,c) as h; similarly for Judge B we treat (j,b) as v and (a,c) as h.

Although the **vis**-convention suggests that protocol development, e.g. as in §8.2 to come, requires proofs for each observer (since the patterns of variables' visibility might differ), in fact we can usually follow a single chain of reasoning whose steps are shown to be valid for two or even all three observers at once.

We also allow *local* variables, with which (as usual) implementations introduce extra structure that does not appear in the specification and so escapes any requirements of *functional* correctness. But local variables are still *security* risks as much as global variables, especially when they are used to describe message passing: with perfect recall, their visibility is not affected by their being local.

8.1 The encryption lemma: qualitative vs. quantitative reasoning

Rather than appeal constantly to the basic semantics (Fig. 1) instead we have accumulated, with experience, a repertoire of identities –a program algebra– which we use to reason at the source level. Those identities themselves are proved directly in the semantics but, after that, they become permanent members of the designer's toolkit. One of the most common is the *Encryption Lemma*.

Let statement $(\mathsf{v}\nabla\mathsf{h}){:=}\,E$ set Booleans v,h so that their exclusive-or $\mathsf{v}\nabla\mathsf{h}$ equals Boolean E: there are exactly two possible ways of doing so. In our earlier work [14], we proved that when the choice is made demonically, on a single run nothing is revealed about E; in our refinement style we express that as

$$\mathbf{skip} \quad = \quad |\!|\!|[\ \mathbf{vis}\ \mathsf{v};\mathbf{hid}\ \mathsf{h};\ (\mathsf{v}\nabla\mathsf{h}){:=}\,E\ ]|\!|\!| \ , \tag{6}$$

where equality is "refinement both ways," and the "begin-end" brackets $|\!|\!|[\cdots]|\!|\!|$ enclose local declarations. In our current model we can prove that *exactly the same identity holds* if the choice is made uniformly [11]. That means that extant qualitative source-level proofs that rely only on "upgradeable identities" like (6) can be used *as is* for quantitative results provided the demonic choices involved are converted to uniform choice. And that is the case with our current example.

Beyond the *Encryption Lemma*, we use *Two-party Conjunction* [20, 13] and *Oblivious Transfer* [16, 14] in our implementation, but in our development we refer *only to their specifications*. Just as for the Encryption Lemma, the algebraic proofs of their implementations [14, 13] apply quantitatively provided we interpret the (formerly) demonic choice as uniform.

8.2 The Three-Judges *development* (sketch)

Here we present only a sketch of the steps used to take specification (5) to an implementation (Fig. 2); details are given elsewhere [11].

We begin with a two-party conjunction [20, 13] which sets variables $\mathsf{b}_0, \mathsf{c}_0$ so that their exclusive-or equals the conjunction of two other variables b, c. We show how to implement this elsewhere [13]; but here we need only refer to its specification to check –via (6)– that the introduction of the two-party conjunction into (5) preserves both its functional and secure correctness. We have

$$\mathsf{j}{:=}(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) \quad = \quad \textbf{skip};\ \mathsf{j}{:=}(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2) \qquad \text{“}\textbf{skip}\text{ is identity”}$$

$$= \quad |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0; \mathbf{vis}_C\ \mathsf{c}_0;\ (\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\mathsf{b} \wedge \mathsf{c};\ ]\!|\!|; \quad \text{“Introduce two-party conjunction:}$$
$$\mathsf{j}{:=}\ (\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)\ \ldots \qquad \text{correctness trivial for } A\text{; use (6) for } B, C\text{.”}$$

We must justify this step for all three observers: for A the introduced block equals **skip** since all assignments are to local variables hidden from A; for B the block is equivalent to **skip** because it is an instance of the Encryption Lemma with v as b_0 and h as c_0; for C it is as for B, but reversed. With similar reasoning we introduce a two-party disjunction and reorganise:

$$\ldots = \quad |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0; \mathbf{vis}_C\ \mathsf{c}_0;\ (\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\mathsf{b} \wedge \mathsf{c};\ ]\!|\!|; \quad \text{“Introduce two-party disjunction}$$
$$|\!|\![\ \mathbf{vis}_B\ \mathsf{b}_1; \mathbf{vis}_C\ \mathsf{c}_1;\ (\mathsf{b}_1 \nabla \mathsf{c}_1){:=}\mathsf{b} \vee \mathsf{c};\ ]\!|\!|; \quad \text{with same justification as above.”}$$
$$\mathsf{j}{:=}(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)$$

$$= \quad |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; \qquad \text{“Reorganise declarations and scoping”}$$
$$(\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\mathsf{b} \wedge \mathsf{c};\ (\mathsf{b}_1 \nabla \mathsf{c}_1){:=}\mathsf{b} \vee \mathsf{c};$$
$$\mathsf{j}{:=}(\mathsf{a}{+}\mathsf{b}{+}\mathsf{c} \geq 2)\ ]\!|\!|$$

$$= \quad |\!|\![\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1; \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; \qquad \text{“Boolean algebra;}$$
$$(\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\mathsf{b} \wedge \mathsf{c};\ (\mathsf{b}_1 \nabla \mathsf{c}_1){:=}\mathsf{b} \vee \mathsf{c}; \qquad \text{expression } \mathsf{b_a} \nabla \mathsf{c_a} \text{ abbreviates}$$
$$\mathsf{j}{:=}\mathsf{b_a} \nabla \mathsf{c_a}\ ]\!|\!| \ldots \qquad \mathsf{b}_1 \nabla \mathsf{c}_1\ \textbf{if a else}\ \mathsf{b}_0 \nabla \mathsf{c}_0\text{.”}$$

Now since its being assigned to j will reveal $\mathsf{b}_a \nabla \mathsf{c}_a$ to everyone, and thus to A in particular, it does no harm first to capture that value in variables visible to A alone and then to have Agent A reveal those to everyone else:

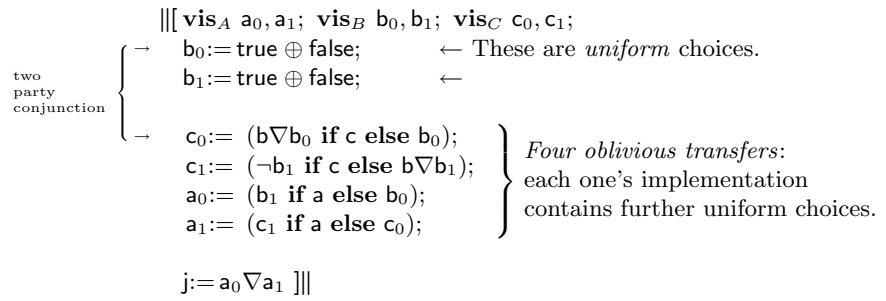
$$\ldots = \quad |\!|\![\ \mathbf{vis}_A\ \mathsf{a}_0, \mathsf{a}_1;\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; \qquad \text{“Introduce private variables of } A\text{”}$$
$$(\mathsf{b}_0 \nabla \mathsf{c}_0){:=}\mathsf{b} \wedge \mathsf{c};\ (\mathsf{b}_1 \nabla \mathsf{c}_1){:=}\mathsf{b} \vee \mathsf{c};$$
$$(\mathsf{a}_0 \nabla \mathsf{a}_1){:=}\mathsf{b_a} \nabla \mathsf{c_a}; \quad (\dagger)$$
$$\mathsf{j}{:=}\ \mathsf{a}_0 \nabla \mathsf{a}_1\ ]\!|\!|\ .$$

The point of using two variables $\mathsf{a}_{\{0,1\}}$ rather than one is to separate the communication $(B, C) \rightarrow A$ at (†) into two oblivious transfers $B \rightarrow A$ and $C \rightarrow A$ [16]. Our last step does that [11], giving finally the code of Fig. 2.

A full implementation at the level of assignments is indicated in Fig. 5 of §C.

9 Conclusions — and an open question

There are many alternatives for a definition of “elementary testing.” Bayes Risk measures how difficult it is to determine h using only one query $\mathsf{h}{=}h$, while



We replace the two Two-Party 'junctions by their implementations as oblivious transfers [13]: each becomes two statements instead of one. The uniformly random flipping of bits $\mathsf{b}_{\{0,1\}}$ is then collected at the start. Correctness is preserved by compositionality.

Fig. 2. Three-Judges Protocol assuming Oblivious Transfers as primitives

Marginal Guesswork [15, 7] allows multiple queries of that form. And *Shannon Entropy* [17] can be related to multiple queries of the form $\mathsf{h}{\in}H$. Other measures, such as *Guessing Entropy* [9, 7], are further variations.

No single one of these elementary testing orderings is objectively the best. Although Shannon Entropy is a well-accepted measure of uncertainty, Pliam observes [15] that e.g. Marginal Guesswork might be better for measuring vulnerability to brute-force attacks; he also shows that there can be no general ordering between these two measures. Smith compares Bayes Risk and Shannon Entropy, speculating that also those measures are mutually incomparable [18].

Whichever definition of elementary testing is chosen, a crucial *practical* concern is to determine whether compositional reasoning can be based on it. Smith suggested that Bayes Risk might not be compositional [18]; we *prove* it isn't compositional — and then we identify a subset ($\sqsubseteq$) of it that is. Further, we have given a practical –and non-trivial– example of how useful that can be (§8).

Our refinement ensures that uncertainty cannot decrease for Marginal Guesswork, Shannon- or Guessing Entropy [11]: it is sound for those three as well as for Bayes Risk (§7). This means that –in spite of the speculations above– in fact the orders *are* comparable: *with contexts*, Bayes Risk is at least as discriminating as any of the others. **Still open is whether ($\sqsubseteq$) is complete for those others.** If it were, then –again, with contexts– *they would all be equally discriminating.*

With concurrency over visible- and hidden actions giving attackers strong capabilities as we do (§2.4), Braun et al. [1] identify *safe* contexts $\mathcal{C}_{\text{safe}}$ such that I "$\preceq$" $\mathcal{C}_{\text{safe}}(I)$; with our emphasis on implementations I *and* their specifications S, by analogy we would be looking for $S{\preceq}I$ implies $\mathcal{C}_{\text{safe}}(S){\preceq}\mathcal{C}_{\text{safe}}(I)$. But rather than restrict *contexts* $\mathcal{C}$ to $\mathcal{C}_{\text{safe}}$'s, instead we restrict the *testing-relation* ($\preceq$) to a subset ($\sqsubseteq$), keeping contexts as they are — a complementary approach.

We are not aware of others' having identified a refinement relation that is either sound or complete with respect to all contexts in their language.

The semantics presented here is similar to a Hidden Markov Model (HMM) augmented with an ordering which determines when a program yields more or less information about the hidden state than another. Whether the established techniques of HMM's apply here (and *vice versa*) has not yet been explored.

Finally the merits of compositional refinement and its relationship to other treatments of noninterference are discussed in detail elsewhere [14, 11, 6].

THE APPENDICES §A,§B,§C to this paper may be found online [10, 11].

References

1. C. Braun, K. Chatzikokolakis, and C. Palamidessi. Compositional methods for information-hiding. In *Proc. FOSSACS'08*, *LNCS* 4962:443–57. Springer, 2008.
2. C. Braun, K. Chatzikokolakis, and C. Palamidessi. Quantitative notions of leakage for one-try attacks. In *Proc. MFPS*, *ENTCS* 249. Elsevier, 2009.
3. K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Probability of error in information-hiding protocols. In *Proc. CSF*, pages 341–354. IEEE, 2007.
4. D. Chaum. The Dining Cryptographers problem: Unconditional sender and recipient untraceability. *Jnl Cryptol.*, 1(1):65–75, 1988.
5. J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp on Security and Privacy*, pages 75–86. IEEE, 1984.
6. Probabilistic Systems Group. `www.cse.unsw.edu.au/~carrollm/probs`.
7. B. Köpf and D. Basin An information-theoretic model for adaptive side-channel attacks. *Proc. 14th ACM Conf. Comp. Comm. Security*. 2007.
8. D. Kozen. A probabilistic PDL. *Jnl Comp Sys Sci*, 30(2):162–78, 1985.
9. J.L. Massey. Guessing and entropy. In *Proc. IEEE International Symposium on Information Theory*, page 204. 1994.
10. A.K. McIver, L.A. Meinicke, and C.C. Morgan. Draft of this paper including its appendices. [6, `McIver:10`].
11. A.K. McIver, L.A. Meinicke, and C.C. Morgan. Draft full version of this paper. `www.comp.mq.edu.au/~lmeinick/icalp.pdf`.
12. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Tech. Mono. Computer Science. Springer, 2005.
13. A.K. McIver and C.C. Morgan. Sums and lovers: Case studies in security, compositionality and refinement. In A. Cavalcanti and D. Dams, editors, *Proc. FM '09*, *LNCS* 5850. Springer, 2009. Treats *Two-party secure computation*.
14. C.C. Morgan. *The Shadow Knows:* Refinement of ignorance in sequential programs. *Science of Computer Programming*, 74(8), 2009. Treats *Oblivious Transfer*.
15. J.O. Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. In *Proc. INDOCRYPT 2000*, *LNCS* 1977:67–79. Springer, 2000.
16. R. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initialiser. Technical report, M.I.T., 1999. `//theory.lcs.mit.edu/~rivest/Rivest-commitment.pdf`.
17. C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.
18. G. Smith. Adversaries and information leaks. In G. Barthe and C. Fournet, eds., *Proc. 3rd Symp. Trust. Global Comp.*, *LNCS* 4912:383–400. Springer, 2007.
19. K. Trustrum. *Linear Programming*. Library of Mathematics. Routledge and Kegan Paul, London, 1971.
20. A.C. Yao. Protocols for secure computations (extended abstract). In *Proc. FOCS 1982*, pages 160–164. IEEE, 1982.

Appendices: Detailed proofs from §7; and *The Three Judges* code in full.

These appendices are accessible also (on the internet) in the draft *full* version of this report [11].

The first, §A, gives the complete proofs of *soundness* (§7.1) and *completeness* (§7.2); the second §B provides a fully worked-out example of the completeness proof §A.2 in action.

The third, short appendix §C gives in its Fig. 5 a look at the implementation of the Three Judges protocol as it would appear at the level of simple assignment statements, i.e. with the sub-protocols in Fig. 2 in-lined. That gives a better idea of the great utility of stepwise developments in this context, since a brute force correctness-by-construction *by hand* is unusual for a secure protocol of this size.

A Proof of soundness and completeness

We recall from §7 that the Bayes Risk of an output hyper-distribution Δ' is given by $\mathsf{br}.\Delta' := (\odot\,(v',\delta') : \Delta' \bullet \sqcup\delta')$, and that functional behaviour of Δ' is $\mathsf{ft}.\Delta' := \{\!\{(v',\delta') : \Delta'; h' : \delta' \bullet (v',h')\}\!\}$. Based on that, the elementary testing order between hyper-distributions $\Delta'_S \preceq \Delta'_I$ is defined to be $\mathsf{br}.\Delta'_S \geq \mathsf{br}.\Delta'_I$ and $\mathsf{ft}.\Delta'_S = \mathsf{ft}.\Delta'_I$.

That definition is extended pointwise to programs so that $S \preceq I$ just when $[\![S]\!].(v,\delta) \preceq [\![I]\!].(v,\delta)$ for all input split-states (v,δ).

A.1 Soundness

Fix an initial split-state and construct the output hyper-distributions $\Delta'_{\{S,I\}}$ that result from S, I respectively applied to it. Then since we are assuming that $S \sqsubseteq I$, we must have $\Delta'_S \sqsubseteq \Delta'_I$ — and we show first that this implies $\Delta'_S \preceq \Delta'_I$.

Since $S \sqsubseteq I$ trivially guarantees that $\mathsf{ft}.\Delta'_S = \mathsf{ft}.\Delta'_I$ –recall §6.1– we need only show that the Bayes-Risk condition in the elementary testing order is satisfied. Because refinement is defined in terms of partitions, we define elementary testing brp for them on their own as $\mathsf{brp}.\Pi := (\sum \pi : \Pi \bullet \sqcup\pi)$ so that we get the useful identity $\mathsf{br}.\Delta = (\sum v \bullet \mathsf{brp}.(\mathsf{fracs}.\Delta.v))$.

Now we observe (i) that if Π_1 is sim-finer than Π_2 then $\mathsf{brp}.\Pi_1 = \mathsf{brp}.\Pi_2$, since maximum distributes multiplication by non-negative scalars; and (ii) that if Π_1 is (not necessarily sim-)finer than Π_2 then (still) we have $\mathsf{brp}.\Pi_1 \geq \mathsf{brp}.\Pi_2$, since the maximum of a sum is no more than the sum of the maxima. Referring again to Def. 1 in §6.1, we then have that $\Delta'_S \sqsubseteq \Delta'_I$ implies

$$
\begin{array}{lll}
 & \mathsf{br}.\Delta'_S & \\
= & (\sum v \bullet \mathsf{brp}.(\mathsf{fracs}.\Delta'_S.v)) & \text{“definition } \mathsf{brp}\text{”} \\
= & (\sum v \bullet \mathsf{brp}.\Pi) & \text{“}\Pi \text{ is sim-finer than } \mathsf{fracs}.\Delta'_S.v\text{, for some partition } \Pi\text{”}
\end{array}
$$

$$\begin{array}{lll} \geq & (\sum v \bullet \mathsf{brp}.(\mathsf{fracs}.\Delta'_I.v)) & \text{“}\Pi\text{ is finer than }\mathsf{fracs}.\Delta'_I.v\text{”} \\ = & \mathsf{br}.\Delta'_I\ , & \text{“definition }\mathsf{brp}\text{”} \end{array}$$

hence $\Delta'_S \preceq \Delta'_I$ as claimed. Since the input split-state was arbitrary, we have more generally established $S \preceq I$.

Now we complete the argument by appealing to Monotonicity (Lem. 1), which tells us that from $S \sqsubseteq I$ we have $\mathcal{C}(S) \sqsubseteq \mathcal{C}(I)$ for any $\mathcal{C}$. Using the above with $\mathcal{C}(S), \mathcal{C}(I)$ instead of S, I then gives us $\mathcal{C}(S) \preceq \mathcal{C}(I)$ as required.

A.2 Completeness

Since $\mathcal{H}$ is finite, we assume *wlog* that it is the integers $1..H$ for some H; full distributions over $\mathcal{H}$ are thus H-tuples of reals, summing to 1. In §B we follow a particular example through this proof.

If $S \not\sqsubseteq I$ then there must be an initial split-state from which S, I yield hyper-distributions $\Delta'_{\{S,I\}}$ with $\Delta'_S \not\sqsubseteq \Delta'_I$. We can assume however that those $\Delta'_{S,I}$ give equal overall probabilities to *visible* variables since, if they did not, they would be functionally different, giving $S \not\preceq I$ immediately. This being so, we can assume that for some final v' occurring with overall probability p' we have that partition $\Pi_S := \mathsf{fracs}.\Delta'_S.v'$ cannot be transformed into partition $\Pi_I := \mathsf{fracs}.\Delta'_I.v'$ via the two steps given in Def. 1.

Let N^0, N be the number of fractions in Π_S, Π_I respectively; note that the probabilities in both Π_S and Π_I sum to the same value, namely p' from above.

Now Π_I can be considered to be an H-row-by-N-column matrix, since it contains N fractions each one an H-tuple written as a column. Further, if we concatenate all the columns together we have the coordinates of a single point in HN-dimensional space. Call that point x_I.

Similarly we can write Π_S as an H-row-by-N^0-column matrix but, instead of making a point from that, we generate from it a *set* of matrices, each of the same shape $H \times N$ as Π_I, as follows.

For a *decision* function $\sigma\colon (1..N^0) \to (1..N)$ let $\Pi_S \times \overline{\overline{\sigma}}$ be an $H \times N$ matrix formed by post-multiplying Π_S with an $N^0 \times N$ matrix $\overline{\overline{\sigma}}$ whose element at row n^0 and column n is (1 **if** $n{=}\sigma.n^0$ **else** 0); in effect each column n_0 of the N^0 columns of Π_S is added into the column $\sigma.n^0 \in 1..N$ of a new matrix with N columns. Thus by varying σ we make a set of HN-matrices X_S and in fact *the convex closure of X_S contains exactly the possible N-fraction refinements of Π_S* that can be formed by the procedure (i,ii) of Def. 1.[8] Thus by considering X_S equivalently to be a set of points $\{\sigma \bullet \Pi_S \times \overline{\overline{\sigma}}\}$ in HN space, our assumption of non-refinement becomes simply that x_I is not in the convex closure of X_S.

[8] This can be seen as follows. Once the dimensions N^0, N are fixed, combining (i,ii) in our current Def. 1 of refinement is equivalent to asserting the existence of a row-1-summing matrix R of dimensions $N^0 \times N$ such that $\Pi_S \times R = \Pi_I$. We are claiming here that it is the same to use a number of exactly-one-1-per-row strategy-matrices $\overline{\overline{\sigma}}$ and then interpolate the results. That is, we are asserting the equality

$$\{R \bullet \Pi_S \times R\} \quad = \quad \mathsf{ccl}.\{\sigma \bullet \Pi_S \times \overline{\overline{\sigma}}\}\ ,$$

Now since $x_I \not\in X_S$ and X_S is convex, the *Separating Hyperplane Lemma* [19] gives us an HN-dimensional hyperplane with normal some M, also an $H\times N$ matrix, separating x_I from X_S in the sense that for all $x_S \in X_S$ we have $x_S \cdot M < x_I \cdot M$ where dot-product $(\cdot)$ is the element-wise product of the two matrices then all summed to a single scalar.

Note that we can choose M to have all non-negative coefficients because x_I and all the x_S's have the same sum p', and thus we can add any constant to all elements of M without affecting its separating property; similarly we can scale it by any positive number. Thus we assume *wlog* that M is non-negative and that all its rows sum to no more than 1. This M will give us a context $\mathcal{C}$ such that $\mathcal{C}(S) \not\preceq \mathcal{C}(I)$, as we now show.

The context is $(-\,;C)$ where C is **if** $\mathsf{v}{=}v'$ **then** $\mathsf{h} :\in D.\mathsf{h}$ **else** $\mathsf{h} := 0$ **fi** for a distribution-valued function D we will define below, that is when $\mathsf{v}{=}v'$ an overwriting of h based on a distribution $D.\hat{h}$ that thus depends on the value $\hat{h}$ found incoming to C in h itself: the outgoing value of h will then be h' with probability $(D.\hat{h}).h'$. We effectively ignore the $\mathsf{v}{\neq}v'$ case by always assigning 0 to h then.

Now suppose we execute an attack against $I;C$ as follows. Observe I first and, whenever its final v value is v', note which fraction π_n in Π_I it produced in h at the same time: this information is available from I's program-counter behaviour, the earlier values of v and the source code. Then allow execution to continue and finish off with C, and finally guess h's final value to be that same n we observed. What is our probability of success?

For any particular fraction π_n produced, that probability is be given by the summation $(\sum \hat{h}\colon 1..H \bullet \pi_n.\hat{h} * D.\hat{h}.n)$ since (recall) the fractions are pre-scaled by their probability of occurrence; thus over all possible fractions it is $(\sum n\colon 1..N \bullet (\sum \hat{h}\colon 1..H \bullet \pi_n.\hat{h} * D.\hat{h}.n))$. Collecting the summations, we rewrite that as $(\sum n, \hat{h} \bullet \pi_n.\hat{h} * D.\hat{h}.n)$ — which is just the dot-product between π and D as above. Therefore if we could choose D to be M itself, the attack above will

---

where ccl is convex closure. By linearity of matrix multiplication we can take the $(\Pi_S \times)$ out, leaving to be shown that any such R can be expressed as a ccl-interpolation of such $\overline{\overline{\sigma}}$'s.

We argue as follows. Fix R, and identify a non-zero minimum element in each of its rows; let c be the minimum of those row-minima; select the $\overline{\overline{\sigma}}$ that has 1's in the row-minima positions exactly; and subtract $c\overline{\overline{\sigma}}$ from R to give some R'.

Now R' has at least one more 0 entry than R did, and yet the rows of R' still have equal sums, now $1{-}c$. Continue this process from R' onwards : it must stop, since the number of 0's increases each time; and when it does stop it must be because there is an all-0 row, in which case *all* rows must be all-0, since the row sums have remained the same all the way through.

The collection of $\overline{\overline{\sigma}}$'s and their associated c's is the interpolation we had to find: for example, in three steps the procedure above generates the interpolation

$$\begin{pmatrix} 1/3 & 2/3 \\ 3/4 & 1/4 \end{pmatrix} \quad = \quad 1/4 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + 1/12 \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} + 2/3 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \ .$$

succeed with probability $x_I{\cdot}M$ exactly — and the Bayes Risk of $I;C$ thus has value *at least* $x_I{\cdot}M$.

Unfortunately we cannot take M for D just as it is, since its rows $M.h$ sum possibly to less than 1 — they are thus *sub*-distributions over n. And so to turn it into a distribution that can be used in a program fragment we define D to be M with an extra "column zero" whose entries are chosen to make its rows add up to 1 exactly. Note that our attack never actually chooses 0, since I produces π_n only for $n{:}\,1..N$ — there is no π_0.

Now we show that the Bayes Risk of $S;C$ is strictly less than the success probability of our *particular* guessing attack against $I;C$; and we determine that Risk $S;C$ as follows. If the value of v produced by S is not v', ignore this run (as for I). Otherwise note the index n^0 of the fraction π_{n^0} produced by S, and note also which value n produced finally in h would occur with maximum probability were C then to be executed, carrying on from that π_{n^0}. Define a decision function $\sigma_S.n^0{:=}n$ by doing this for each n^0. Note σ_S is defined, in advance, by examination of the source text of S and C — nothing need be run.

The Bayes Risk of $S;C$, during an actual run, is realised by observing which fraction n^0 was produced by S and then consulting σ_S to determine the greatest guess $\mathsf{h}{=}\sigma_S.n^0$, the one that is most likely to succeed after the C.

More significantly, the chance our attack will succeed is just $(\mathit{\Pi}_S{\times}\overline{\overline{\sigma_S}}){\cdot}M$, that is $x_S{\cdot}M$ for some $x_S{\in}X_S$ formed by taking σ_S to be strategy σ in the discussion of refinement that led to our construction of M in the first place. This is because the chance of success is $(\sum n^0, \hat{h} \bullet \pi_{n^0}.\hat{h} * D.\hat{h}.(\sigma_S.n^0))$, by analogy with the I-case but replacing the last n by $\sigma.n^0$, and in the calculation

$$
\begin{array}{lll}
= & & \text{“}\sigma_S.n^0 \text{ is never 0, so } M \text{ and } D \text{ agree here by definition”} \\
 & (\sum n^0, \hat{h} \bullet \pi_{n^0}.\hat{h} * M.\hat{h}.(\sigma_S.n^0)) & \\
= & (\sum n^0, \hat{h}, n \mid n{=}\sigma_S.n^0 \bullet \pi_{n^0}.\hat{h} * M.\hat{h}.n) & \text{“introduce } n\text{”} \\
= & (\sum \hat{h}, n \bullet (\sum n^0 \mid n{=}\sigma_S.n^0 \bullet \pi_{n^0}).\hat{h} * M.\hat{h}.n) & \text{“rearrange summations”} \\
= & (\sum \hat{h}, n \bullet \pi_n.\hat{h} * M.\hat{h}.n) & \text{“define } \pi_n{:=}(\sum n^0 \mid n{=}\sigma_S.n^0 \bullet \pi_{n^0})\text{”}
\end{array}
$$

we define π to be $\mathit{\Pi}_S{\times}\overline{\overline{\sigma_S}}$, the refinement of $\mathit{\Pi}_S$ under strategy σ_S, as claimed.

Now since $x_S{\cdot}M{<}x_I{\cdot}M$ for any $x_S{\in}X_S$, our particular attack against $I;C$ will succeed with strictly greater probability than the greatest attack against $S;C$. Thus –in spite of their names– in fact $(S;C) \not\preceq (I;C)$, just as we require. [9]

[9] We deal with the detail not being allowed to choose 0, if necessary, by adding a second context program that acts as **skip** when $h{\neq}0$; but when $h{=}0$ it executes a large probabilistic choice over h to distribute the 0 value over enough new values $-1, -2\ldots$ to make sure none of them will have a large enough probability to be chosen.

B Example of completeness construction in §A.2

Here we illustrate the completeness proof by applying it to the example of §6.1 from the main paper, where we claimed that $P_4 \not\sqsubseteq P_2$. That is, we justify the claim by using §A.2 to find a C such that indeed $P_4;C \not\preceq P_2;C$.

First suppose that $\mathcal{H}$ is $\{1..3\}$. Our v' is 1, since that is where we find the difference between P_2 and P_4 in the residual uncertainties of h, and we calculate

$$\text{when } v'{=}1 \quad \left\{ \begin{array}{lcl} \Pi_{P_4} & := & \langle \{\!\{ 1^{@\frac{1}{4}}, 3^{@\frac{1}{12}} \}\!\}, \{\!\{ 1^{@\frac{1}{12}}, 3^{@\frac{1}{4}} \}\!\} \rangle \\ \Pi_{P_2} & := & \langle \{\!\{ 1^{@\frac{1}{6}} \}\!\}, \{\!\{ 1^{@\frac{1}{6}}, 3^{@\frac{1}{6}} \}\!\}, \{\!\{ 3^{@\frac{1}{6}} \}\!\} \rangle \ , \end{array} \right.$$

so that N^0, the number of fractions in Π_{P_4}, is 2 and N is 3. Both partitions sum to $p'{=}2/3$, the probability associated with our overall focus on the $v'{=}1$ case.

The HN matrix corresponding to Π_{P_2} is then given by $\begin{pmatrix} 1/6 & 1/6 & 0 \\ 0 & 1/6 & 1/6 \end{pmatrix}$ of dimensions $2{\times}3$, where the rows correspond to values $1, 3$ of h and the columns to Π_{P_2}'s three fractions. The point obtained by concatenating the columns is $(1/6, 0, 1/6, 1/6, 0, 1/6)$, and is in 6-dimensional space; but to avoid a proliferation of fractions, we scale everything up from now on by a factor of 12, and so take x_{P_2} to be the point $(2, 0, 2, 2, 0, 2)$.

Now the scaled-up HN^0 matrix corresponding to Π_{P_4}, followed by the first 4 of the $3^2 = 9$ possible decision functions that can be applied to it, is given by

$$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \cdots$$

Applying the decision functions above shows the first four elements of X_{P_4} to be

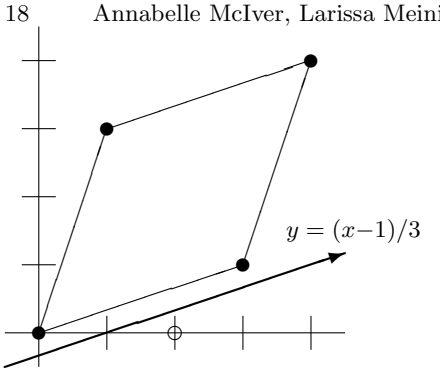
$$\begin{pmatrix} 4 & 0 & 0 \\ 4 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 3 & 1 & 0 \\ 1 & 3 & 0 \end{pmatrix}, \begin{pmatrix} 3 & 0 & 1 \\ 1 & 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 3 & 0 \\ 3 & 1 & 0 \end{pmatrix} \cdots \ ,$$

whence doing all of them, and writing them as points in 6-dimensional space, gives us the 9 elements

$$\begin{array}{rcl} X_{P_4} & := & \{\ (4,4,0,0,0,0)\ ,\ (3,1,1,3,0,0)\ ,\ (3,1,0,0,1,3)\ , \\ & & \ \ (1,3,3,1,0,0)\ ,\ (0,0,4,4,0,0)\ ,\ (0,0,3,1,1,3)\ , \\ & & \ \ (1,3,0,0,3,1)\ ,\ (0,0,1,3,3,1)\ ,\ (0,0,0,0,4,4)\ \ \} \end{array}$$

altogether. Our claim that $P_4 \not\sqsubseteq P_2$ suggests that x_{P_2} should not lie in the convex closure of X_{P_4} in 6-space.

We can see this easily by concentrating on the first two dimensions only: for x_{P_2} that is $(2, 0, \cdots)$; and for X_{P_4}, after removing duplicates, that is $(4, 4, \cdots)$, $(3, 1, \cdots)$, $(1, 3, \cdots)$ and $(0, 0, \cdots)$. Point x_{P_2} is not in their convex closure because all four have first two coordinates both positive or both zero, a property preserved by any convex combination but not shared by $(2, 0, \cdots)$.



We insert a hyperplane (just a line, in 2-space) midway between the separated point and the convex shape, parallel to the boundary of the latter.

Fig. 3. Finding a separating hyperplane $y = (x{-}1)/3$ in 2-space.

Now thinking in only two dimensions, we can easily find the separating hyperplane with a picture. Fig. 3 shows x_{P_2} as an open circle at $(2, 0)$, while the filled circles give the boundaries of the diamond-shaped convex closure of X_{P_4}. Clearly x_{P_2} is not in that closure, and indeed the (degenerate) hyperplane, the line $y = (x{-}1)/3$ separates it from the diamond. The normal of that hyperplane (up and left, perpendicular to the line) has direction $(-1, 3)$, and when we fill in the other four dimensions as zero –since they're not needed for separation– that gives us a candidate normal of $(-1, 3, 0, 0, 0, 0)$ in 6-space — but it is not quite M yet. Its dot-product with x_{P_2} is -2; and with the nine elements of X_{P_4} we get the nine dot-products $8, 0, 0, 8, 0, 0, 8, 0, 0$, showing indeed the separation we expect, but in the wrong direction: the values 0 and 8 for P_4 are both greater than the value -2 for P_2, and we want them to be less. Accordingly we multiply the normal above by -1 then we add 3 to all its elements to make them non-negative; and finally we divide everything by 10 to make its rows sum to no more than 1. That gives

$$M := \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0 & 0.3 & 0.3 \end{pmatrix} \quad \text{and then} \quad D := \begin{pmatrix} \overset{\downarrow \text{Zero'th column, for one-summing}}{0} & 0.4 & 0.3 & 0.3 \\ 0.4 & 0 & 0.3 & 0.3 \end{pmatrix} ,$$

where to get D from M we added a new "zero-th column" to make each row one-suming exactly. The distinguishing context $(-\,;\, C)$, say, must then overwrite h according to the distribution given by a row of D, the one selected by the value $\hat{h}$ of h incoming to C; thus we construct C to be

$$\begin{array}{l} \textbf{if } \mathsf{v}{=}1 \textbf{ then } \ \mathsf{h}{:\in} \left(\{\!\{ 1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3} \}\!\} \textbf{ if } \mathsf{h}{=}1 \textbf{ else } \{\!\{ 0^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3} \}\!\}\right) \\ \qquad \textbf{else } \ \mathsf{h}{:=}\,0 \textbf{ fi} \ , \end{array}$$

with the outer **if** effectively restricting our attack to occur only when $v'{=}1$. This gives us our context $(-\,;\, C)$. Let us now check that it actually works.

We begin with $P_4; C$. Its output hyper-distribution is (after some calculation) given by

$$\begin{array}{l} \{\!\{ \ (0, \{\!\{0\}\!\})^{@\frac{1}{3}} \ , \\ \quad (1, \{\!\{ 0^{@\,0.1}, 1^{@\,0.3}, 2^{@\,0.3}, 3^{@\,0.3} \}\!\})^{@\frac{1}{3}} \ , \\ \quad (1, \{\!\{ 0^{@\,0.3}, 1^{@\,0.1}, 2^{@\,0.3}, 3^{@\,0.3} \}\!\})^{@\frac{1}{3}} \ \ \}\!\} \ , \end{array}$$

and the greatest guessing attack has value $1/3{*}1+1/3{*}0.3+1/3{*}0.3 \approx 0.53$. On the other hand, for $P_2;C$ the output hyper-distribution is

$$\begin{array}{l}\{\!\{\ (0, \{\!\{0\}\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\{1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})^{@\frac{1}{6}} \qquad\quad ,\\ \quad (1, \{\!\{0^{@\,0.2}, 1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\{0^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})^{@\frac{1}{6}} \qquad\quad \}\!\}\ ,\end{array}$$

and here the greatest attack has value $1/3{*}1{+}1/6{*}0.4{+}1/3{*}0.3{+}1/6{*}0.3 \approx 0.55$. Note that in the third summand we took 0.3 rather than the larger 0.4 associated with 0, since as part of our construction we exclude guesses that h is 0. [10]

Thus we have established that $P_4;C \not\sqsubseteq P_2;C$ (for the adjusted C — see Footnote 10), because the Risk for the former has value 0.53 but for the latter has the greater value 0.55. Hence when our refinement relation insists that $P_4 \sqsubseteq P_2$ –as we argued earlier above– in fact it is not being too severe, but rather it is acting just as a compositional closure should. It protects us not only against the context C we just made, but all other contexts too — in spite of the fact that in isolation P_4 and P_2 are not distinguished by elementary testing.

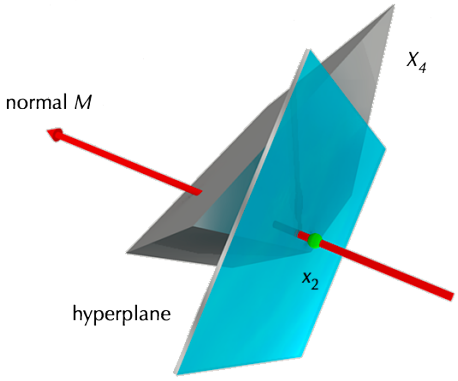


Fig. 4. Finding a separating hyperplane, with (2,1,1) as normal, in 3 of the 6 dimensions.

Finally — we gave a different distinguishing context earlier, in §7.2 in the main paper,

[10] Dealing with this detail would split the 0-case in half, uniformly distributed over $-1, -2$, since the resulting probability 0.4/2 for each would then be small enough that a greatest attack would never choose it. The adjusted context C' would contain

$$\mathsf{h}{:}\in (\{\!\{1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\} \textbf{ if } \mathsf{h}{=}1 \textbf{ else } \{\!\{ -2^{@\,0.2}, -1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})$$

and the resulting output for $P_2;C'$ would be

$$\begin{array}{l}\{\!\{\ (0, \{\!\{0\}\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\{1^{@\,0.4}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})^{@\frac{1}{6}}\ ,\\ \quad (1, \{\!\{ -2^{@\,0.1}, -1^{@\,0.1}, 1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})^{@\frac{1}{3}}\ ,\\ \quad (1, \{\!\{ -2^{@\,0.2}, -1^{@\,0.2}, 2^{@\,0.3}, 3^{@\,0.3}\}\!\})^{@\frac{1}{6}} \qquad\quad \}\!\}\ ,\end{array}$$

in which neither -2 nor -1 would ever be chosen for a greatest attack. This trick is made easier by our using greatest success rather than least failure.

$$
\begin{array}{l}
|||[\ \mathbf{vis}_A\ \mathsf{a}_0, \mathsf{a}_1;\ \mathbf{vis}_B\ \mathsf{b}_0, \mathsf{b}_1;\ \mathbf{vis}_C\ \mathsf{c}_0, \mathsf{c}_1; \\
\quad \mathsf{b}_0 := \mathsf{true} \oplus \mathsf{false}; \\
\quad \mathsf{b}_1 := \mathsf{true} \oplus \mathsf{false}; \\
\\
\quad |||[\ \mathbf{vis}_B\ \mathsf{m}'_0, \mathsf{m}'_1; \mathbf{vis}_C\ \mathsf{c}', \mathsf{m}'; \\
\qquad \mathsf{c}' := \mathsf{true} \oplus \mathsf{false}; \\
\qquad \mathsf{m}'_0 := \mathsf{true} \oplus \mathsf{false}; \\
\qquad \mathsf{m}'_1 := \mathsf{true} \oplus \mathsf{false}; \\
\qquad \mathsf{m}' := \mathsf{m}'_{\mathsf{c}'}; \\
\\
\qquad \mathbf{vis}_{ABC}\ \mathsf{x}, \mathsf{y}_0, \mathsf{y}_1; \\
\qquad \mathsf{x} := \mathsf{c} \nabla \mathsf{c}'; \\
\qquad \mathsf{y}_0 := \mathsf{b}_0 \nabla \mathsf{m}'_{\mathsf{x}}; \\
\qquad \mathsf{y}_1 := \mathsf{b} \nabla \mathsf{b}_0 \nabla \mathsf{m}'_{\neg \mathsf{x}}; \\
\qquad \mathsf{c}_0 := \mathsf{y}_{\mathsf{c}} \nabla \mathsf{m}' \\
\quad]||; \\
\\
\quad \mathsf{c}_1 := \ (\neg \mathsf{b}_1\ \mathbf{if}\ \mathsf{c}\ \mathbf{else}\ \mathsf{b} \nabla \mathsf{b}_1); \\
\quad \mathsf{a}_0 := \ (\mathsf{b}_1\ \mathbf{if}\ \mathsf{a}\ \mathbf{else}\ \mathsf{b}_0); \\
\quad \mathsf{a}_1 := \ (\mathsf{c}_1\ \mathbf{if}\ \mathsf{a}\ \mathbf{else}\ \mathsf{c}_0); \\
\\
\quad \mathsf{j} := \ \mathsf{a}_0 \nabla \mathsf{a}_1\ ]||
\end{array}
$$

Implementation of oblivious transfer [16], whose correctness in this semantics is proved elsewhere [11, 14] and whose use here in place of its specification is justified by compositionality with no further proof required.

Three more oblivious transfers, each one to be expanded as above.

Each of the other three transfers would expand to a similar block of code, making about 40 lines of code in all.

Fig. 5. Three-Judges *implementation* in elementary terms

and we did not say where it came from. In fact it is one this proof can generate, and is simpler than the other in that it needs no $\mathsf{h} := 0$ case: the rows of its normal M just happen to have the same sum. (But it therefore does not illustrate all the features of the completeness proof, which is why we didn't use it for our example above.)

Finding the normal that generates the C of §7.2, however, is harder if done geometrically: it turns out that we would have had to specialise to three coordinate indices 2, 3 and 5 rather than just the first two 1 and 2. (That is another reason we chose a different example above.) The resulting inspection of X_4 and x_2 –to see just where to slip the hyperplane in between– would then have had to be done in three- rather than two dimensions, as Fig. 4 illustrates (in a side view). In general such hyperplanes can of course be found, without drawing pictures, by using constraint solvers to deal with the linear inequalities symbolically.

C Full implementation of *The Three Judges*

Starting from Fig. 2, in Fig. 5 we replace the specification of the first of its four *Oblivious Transfers* by an implementation in elementary terms [13]. The preservation of correctness, under expansion, is guaranteed by the compositionality of the security semantics.