

Probabilistic Semantic Noninterference

February 28, 2018

1 General Message Passing

1.1 Parties and Schedulers

Assume an expression language with values in Val . A *state* St is a map $\text{Var} \rightarrow (\text{Val} + \perp)$, where Var is a set of variable names. A *buffer* is a value of type $\text{list}(\text{PID} \times \text{Val})$, where PID is a set of party names.

A *handler* is given by the following syntax:

$$P := x \leftarrow \text{read } \ell \mid \text{write } \ell \ e \mid \text{send } i \ e \mid x \leftarrow \text{recv } i \mid x \leftarrow D \ e \mid \text{if } e \text{ then } P \text{ else } P \mid P; P,$$

where e is an expression, ℓ is in Var , i is a PID, and D is a set of distributions.

Handlers are given monadic semantics as functions $\text{St} \rightarrow \text{InBuf} \rightarrow \mathcal{D}(\text{St} \times \text{OutBuf})$, where InBuf and OutBuf are both buffers. Messages in the input buffer are pairs (i, m) , where m is the message content and i is the PID the message came from. Messages in the output buffer, dually, are pairs (i, m) , where i is the PID the message is meant for. This semantics is immediate, except for recv , which returns the list of all messages from i in the input buffer (in reverse chronological order).

A *scheduler* is defined by the following syntax:

$$c := \text{Run } P @ i \mid c_1; c_2,$$

where $k \in \text{Handler}$ is a handler name, and i is a PID.

1.2 Semantics

A *trace* τ is a value of type $(\text{PID} \rightarrow \text{St}) \times \text{NewEv} \times \text{OldEv}$, where NewEv and OldEv are logs of the form $\text{list}(\text{PID} \times \text{PID} \times \text{Val})$. The first component is the global state, the second component is ordered buffer of unprocessed messages, and the third component is the ordered buffer of processed messages. Given a buffer B , define $B|_i$ to be the pairs in B such that the second component is equal to i (preserving order).

Then, define our scheduler semantics $\llbracket c \rrbracket : \text{Trace} \rightarrow \mathcal{D}(\text{Trace})$ by

$$\llbracket \text{Run } P @ i \rrbracket (G, B_u, B_p) := \text{bind}_{\mathcal{D}} (P (G \ i) \ B_{u|_i}) (\lambda(s', o). \text{return}_{\mathcal{D}}(G[i := s'], o \parallel (B_u \setminus B_{u|_i}), B_{u|_i} \parallel B_p))$$

and

$$\llbracket c_1; c_2 \rrbracket \tau := \text{bind}_{\mathcal{D}} (\llbracket c_1 \rrbracket \tau) \llbracket c_2 \rrbracket.$$

where $\text{bind}_{\mathcal{D}}$ and $\text{return}_{\mathcal{D}}$ are the monadic bind and return operations for distributions and \parallel is list concatenation.

1.3 Corruption

Extend the syntax of schedulers as so:

$$c := \dots \mid \text{Corrupt } P @ i.$$

The semantics of the added command is exactly the same as that of `Run`.

We define (static) corruption by the following rewrite rules, given by the judgement $c \rightsquigarrow_{\mathcal{A}} c'$:

$$\frac{}{c \rightsquigarrow_{\mathcal{A}} c} \quad \frac{}{\text{Run } P @ i \rightsquigarrow_{\mathcal{A}} \text{Corrupt } Q @ i} \quad \frac{c_1 \rightsquigarrow_{\mathcal{A}} c'_1 \quad c_2 \rightsquigarrow_{\mathcal{A}} c'_2}{c_1; c_2 \rightsquigarrow_{\mathcal{A}} c'_1; c'_2} \quad \frac{c \rightsquigarrow_{\mathcal{A}} c'}{c \rightsquigarrow_{\mathcal{A}} c'; \text{Corrupt } Q @ i}$$

Then, define $\text{crupt}(c)$ to be the set of parties $i \in \text{PID}$ such that `Corrupt $Q @ i$` appears in c for some Q .

(Note that adversarial programs do not share local state. However, they may freely pass messages to and from each other.)

1.3.1 Other adversarial models

Above, $\rightsquigarrow_{\mathcal{A}}$ models full malicious corruption. We may recover semihonest corruption (i.e., ordinary party-level noninterference without byzantine faults) by replacing $\rightsquigarrow_{\mathcal{A}}$ with the weakened rewrite rule:

$$\frac{}{c \rightsquigarrow_{\mathcal{S}} c} \quad \frac{}{\text{Run } P @ i \rightsquigarrow_{\mathcal{S}} \text{Corrupt } P @ i} \quad \frac{c_1 \rightsquigarrow_{\mathcal{S}} c'_1 \quad c_2 \rightsquigarrow_{\mathcal{A}} c'_2}{c_1; c_2 \rightsquigarrow_{\mathcal{S}} c'_1; c'_2}$$

In the above rule, we do not change the semantics of any party, but only mark certain parties for corruption. By marking only a single party for corruption, the above definition collapses to ordinary noninterference.

Additionally, we may restrict our corruption model with one that cannot corrupt any party at will, but only a certain subset of the parties. An example of this is *honest-verifier zero knowledge*, where the prover is permitted to be malicious but the verifier is not.

1.4 Noninterference

A *leakage* (or *declassification*) property φ is a function $\text{PID} \rightarrow (\text{PID} \rightarrow \text{St}) \rightarrow \text{Val}$. Given an initial global state G , $\varphi i G$ denotes the information i should be able to learn from G after the execution of the protocol. Given a set T of PIDs, define $\varphi T G := \{(i, \varphi i G) \mid i \in T\}$.

Given two distributions D on traces, write $D \equiv_i D'$ if the marginals $\mathcal{D}(\lambda G B_u B_p. (G i, B_{p[i]}))$ are identical for both D and D' . That is, $D \equiv_i D'$ if from party i 's position, D contains exactly the same information as D' on both states and processed

messages (including order of messages). Similarly lift up to sets of parties by defining $D \equiv_T D' := \bigwedge_{i \in T} D \equiv_i D'$.

Fix a corruption model \rightsquigarrow . Given global states G and G' , define $G =_T G'$ to be $\forall i \in T, (G \ i) = (G' \ i)$. Then, say that c is φ -noninterferent if for all c' such that $c \rightsquigarrow c'$ and global states G, G' ,

$$G =_{\text{crupt}(c')} G' \wedge \varphi \text{ crupt}(c') G = \varphi \text{ crupt}(c') G' \implies \llbracket c' \rrbracket (G, \emptyset, \emptyset) \equiv_{\text{crupt}(c')} \llbracket c' \rrbracket (G', \emptyset, \emptyset).$$

That is, c' is φ -noninterferent if whenever two initial global states look identical to the adversary and agree on values of φ , then their induced final traces will appear identical to the adversary.

Note that in the above definition, equivalence of traces is sensitive to order of message delivery – but is only sensitive to message ordering from the perspective of individual parties. That is, two send commands in a handler may be safely reordered if they are sent to different recipients.

1.5 Authenticity

Let θ be a property of traces $\text{PID} \rightarrow ((\text{PID} \rightarrow \text{St}) \times \text{NewEv} \times \text{OldEv}) \rightarrow \{0, 1\}$, and let ϕ be a property of memories $\text{PID} \rightarrow (\text{PID} \rightarrow \text{St}) \rightarrow \{0, 1\}$. Lift ϕ and θ to operate on sets of PIDs by defining $\phi \ T \ G := \bigwedge_{i \in T} \phi \ i \ G$, and similarly for θ .

Fix a corruption model \rightsquigarrow . Then c is (ϵ, θ, ϕ) -authentic if for all c' such that $c \rightsquigarrow c'$ and G ,

$$\Pr_{\tau \leftarrow (\llbracket c' \rrbracket (G, \emptyset, \emptyset))} [\theta \text{ crupt}(c') \ \tau] > \epsilon \implies \phi \text{ crupt}(c') \ G.$$

That is, c is (ϵ, θ, ϕ) -authentic if whenever the adversary triggers the event θ with probability larger than ϵ , then ϕ must be true of the adversary's initial state.

1.6 Examples

In *multiparty computation*, each party is given an input x_i , and a protocol is devised so that each party receives the value $f(\vec{x})$, but no further information is shared. This is modeled by the leakage function $\phi \ i \ G := f((G \ 1).in, \dots, (G \ n).in)$.

Functions may also be asymmetric, in which the leakage function is $\phi \ i \ G := f_i((G \ 1).in, \dots, (G \ n).in)$. A canonical example is *oblivious transfer*, where the sender has two messages m_0 and m_1 , and the receiver has a bit b . The sender should learn nothing (i.e., $f_S(m_0, m_1, b) = ()$), while the receiver should learn the b th message (i.e., $f_R(m_0, m_1, b) := \text{if } b \text{ then } m_0 \text{ else } m_1$.)

We may define *zero-knowledge* proofs to be authentic relative to the predicates “the verifier output 1” and “the prover has a correct witness w to the NP-statement x ”, and noninterferent relative to the leakage function “ $R(x, w) = 1$ ” for the verifier, and no leakage for the prover.

2 Session-restricted Parties

Here, we will consider parties and adversaries which are assumed to follow the protocol's intended session structure.

Let St be as before. The type $\text{Party } M \ M'$ is defined to be functions $\text{St} \rightarrow M \rightarrow \mathcal{D}(\text{St} \times M')$. Parties are given by the following syntax:

$$P := x \leftarrow \text{read } \ell \mid \text{write } \ell \ e \mid \text{send } e \mid x \leftarrow \text{recv} \mid x \leftarrow D \ e \mid \text{if } e \text{ then } P \text{ else } P \mid P; P,$$

where recv evaluates to the current input message, and send alters the intended output message. (I.e., a second invocation of send will overwrite the first one.) Parties come with a typing relation $\vdash P : \text{Party } M \ M'$, which says that all receives and sends respect M and M' .

Schedulers have the same syntax as before:

$$c := \text{Run } P @ i \mid \text{Corrupt } P @ i \mid c; c$$

Schedulers are well-typed when they respect the sessions of the parties:

$$\frac{\vdash P : \text{Party } M \ M'}{\vdash \text{Run } P @ i : \text{Sched } M \ M'} \quad \frac{\vdash P : \text{Party } M \ M'}{\vdash \text{Corrupt } P @ i : \text{Sched } M \ M'} \quad \frac{\vdash c : \text{Sched } M \ M'' \quad \vdash c' : \text{Sched } M'' \ M'}{\vdash c; c' : \text{Sched } M \ M'}$$

A scheduler is *runnable* if it has type $\text{Sched unit } M$, for some M .

2.1 Semantics

As before, a global state is a map $\text{PID} \rightarrow \text{St}$. A *trace* is a value of type $(\text{PID} \rightarrow \text{St}) \times \text{list}(\text{PID} \times \text{Val})$.

Commands are given as mappings from traces to distributions over traces:

$$\begin{aligned} \llbracket \text{Run } P @ i \rrbracket(G, m :: \tau) &:= \text{bind}(P \ (G \ i) \ m)(\lambda(S, m'). \text{return } (G[i := S], m' :: m :: \tau)) \\ (\text{Same for Corrupt}) \\ \llbracket c; c' \rrbracket(G, \tau) &:= \text{bind}(\llbracket c \rrbracket(G, \tau))\llbracket c' \rrbracket \end{aligned}$$

2.2 Corruption

Given a set of PIDs T , define

$$\frac{\vdash P : \text{Party } M \ M' \quad \vdash Q : \text{Party } M \ M' \quad i \in T \quad \frac{c_1 \rightsquigarrow c'_1 \quad c_2 \rightsquigarrow c'_2}{c_1; c_2 \rightsquigarrow c'_1; c'_2}}{c \rightsquigarrow c} \quad \text{Run } P @ i \rightsquigarrow \text{Corrupt } Q @ i$$

As before, define $\text{crupt}(c)$ to be the set of parties in c that are corrupted.

2.3 Trace properties

Trace properties for this language are essentially the same as before: given a trace τ and a set of PIDs T , define $\tau|_T$ to be T 's visible part of τ . Then, c is φ -noninterferent if it is runnable and for all c' such that $c \rightsquigarrow c'$ and global states G and G' ,

$$G =_{\text{crupt}(c')} G' \wedge \varphi \text{ crupt}(c') G = \varphi \text{ crupt}(c') G' \implies \llbracket c' \rrbracket (G, [()]) \equiv_{\text{crupt}(c')} \llbracket c' \rrbracket (G', [()]).$$

Similarly, c is (ϵ, θ, ϕ) -*authentic* if for all c' such that $c \rightsquigarrow c'$ and G ,

$$\Pr_{\tau \leftarrow (\llbracket c' \rrbracket (G, [()])))} [\theta \text{ crupt}(c') \tau] > \epsilon \implies \phi \text{ crupt}(c') G,$$

where $\theta : \text{PID} \rightarrow (\text{PID} \rightarrow \text{St}) \times \text{list}(\text{PID} \times \text{Val}) \rightarrow \{0, 1\}$, and $\phi : \text{PID} \rightarrow (\text{PID} \rightarrow \text{St}) \rightarrow \{0, 1\}$.

2.4 Contexts and Runtime

Let $c[\cdot]$ denote single-holed contexts.