A Calculus of Revelations

being an application of
the Shadow theory,
illustrated with source-level proofs
of security protocols

Annabelle McIver — MACQUARIE UNIVERSITY~SYDNEY

Carroll Morgan

Security and refinement using the Shadow

Security and refinement using the Shadow

The principles (a sketch)

This is an *informal* talk on a topic whose basis is rigorously formalised: thus some (indeed, many) claims will be made without proof; and some (indeed, many) manipulations will be carried out without justification.

But they can be (and have been) proved and justified elsewhere.

Security and refinement using the Shadow

R 18+

some informality,
omitted proofs

The principles (a sketch)

This is an *informal* talk on a topic whose basis is rigorously formalised: thus some (indeed, many) claims will be made without proof; and some (indeed, many) manipulations will be carried out without justification.

But they can be (and have been) proved and justified elsewhere.

Security and refinement using the Shadow:

some informality,
omitted proofs

The principles (a sketch): the approach...

... is related to non-interference
... but allows refinement
... encourages source-level reasoning
... is very suitable for mechanisation

Rather than describe the theory in detail,
I will describe how we want to *use* it, and why.

Security and refinement: based on non-interference

Variables are partitioned into *visible* (low-security, labelled **vis**) and *hidden* (high-security, labelled **hid**).

Visible variables are directly observable; hidden variables are not.

Hidden variables' values can be deduced by observing the visible variables' (intermediate) values, knowing the source code, and following the execution's path through it.

Refinement — from a security point of view — is viewed as an *adversary*.

Refinement — from a security point of view — is viewed as an *adversary*.

Refinement as an *adversary*?

Refinement — from a security point of view — is viewed as an *adversary*.

Refinement as an *adversary*?

In brief: allowing someone to refine your carefully crafted code is just as bad as allowing your run-time adversary to have perfect recall and to observe the program flow.

Refinement — from a security point of view — is viewed as an *adversary*.

Refinement as an *adversary*?

In brief: allowing someone to refine your carefully crafted code is just as bad as allowing your run-time adversary to have perfect recall and to observe the program flow.

These are however the same assumptions you might make to model a distributed algorithm with a single sequential program, a technique which allows considerable simplification.

Refinement — from a security point of view — is viewed as an *adversary*.

Refinement as an *adversary*?

In brief: allowing someone to refine your carefully crafted code is just as bad as allowing your run-time adversary to have perfect recall and to observe the program flow.

These are however the same assumptions you might make to model a distributed algorithm with a single sequential program, a technique which allows considerable simplification.

In spite of this, refinement is such an important development tool that it is worth taking the security hit. Indeed, sometimes the adversary has that power anyway (as in today's examples).

Refinement — from a security point of view — is viewed as an *adversary*.

Refinement as an adversary... *Why?*

Shadow-style program logic, based on assertions of knowledge and ignorance

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means “the adversary Knows Ψ,” i.e. that Ψ can be deduced by observation.

$P\Psi$ means “the adversary admits the Possibility of Ψ,” i.e. that Ψ cannot be ruled out by observation.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

They are dual:
$P\Psi$ is the same
as $\neg K(\neg\Psi)$.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h{:=}\ 1$ does establish $K(h{=}1)$

They are dual:
$P\Psi$ is the same
as $\neg K(\neg\Psi)$.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

They are dual:
$P\Psi$ is the same
as $\neg K(\neg\Psi)$.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h{:=}\ 1$ does establish $K(h{=}1)$

$h{:\in}\ \{0,1\}$ does not establish $K(h{=}1)$

$h{:\in}\ \{0,1\}$ does establish $P(h{=}1)$

They are dual:
$P\Psi$ is the same
as $\neg K(\neg\Psi)$.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

$h :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$ does not establish $P(h=2)$

They are dual:
$P\Psi$ is the same
as $\neg K(\neg\Psi)$.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

h:= 1 does establish $K(h=1)$

h:∈ {0,1} does not establish $K(h=1)$

h:∈ {0,1} does establish $P(h=1)$

h:∈ {0,1} does not establish $P(h=2)$

h:∈ {0,1}; v:∈ {0,1} does establish $P(h=1)$

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

$h :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$ does not establish $P(h=2)$

$h :\in \{0,1\}$; $v :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$; $v := h$ does not establish $P(h=1)$

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

$h :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$ does not establish $P(h=2)$

$h :\in \{0,1\};\ v :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\};\ v := h$ does not establish $P(h=1)$ ←

You cannot be sure that, once this program fragment is complete, you will consider $h=1$ to be possible based on what you have observed.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

$h :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$ does not establish $P(h=2)$

$h :\in \{0,1\}$; $v :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$; $v := h$ does not establish $P(h=1)$ ←

You cannot be sure that, once this program fragment is complete, you will consider $h=1$ to be possible based on what you have observed.

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means “the adversary Knows Ψ,” i.e. that Ψ can be deduced by observation.

$P\Psi$ means “the adversary admits the Possibility of Ψ,” i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

$h :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$ does not establish $P(h=2)$

$h :\in \{0,1\};\ v :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\};\ v := h$ does not establish $P(h=1)$ ←

You cannot be sure that,
on ... ogram
fragment ... you
will c ... o be
possible ... what
you have observed.

$\sqsubseteq$

Hoare-triples and weakest preconditions for knowledge and ignorance

$K\Psi$ means "the adversary Knows Ψ," i.e. that Ψ can be deduced by observation.

$P\Psi$ means "the adversary admits the Possibility of Ψ," i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h=1)$

$h :\in \{0,1\}$ does not establish $K(h=1)$

$h :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$ does not establish $P(h=2)$

$h :\in \{0,1\}$; $v :\in \{0,1\}$ does establish $P(h=1)$

$h :\in \{0,1\}$; $v := h$ does not establish $P(h=1)$

You cannot be sure that, on… ogram fragment… you will c… o be possibl… what you have observed.

This is the Refinement Paradox

$K\Psi$ means “the adversary **K**nows Ψ,” i.e. that Ψ can be deduced by observation.

$P\Psi$ means “the adversary admits the **P**ossibility of Ψ,” i.e. that Ψ cannot be ruled out by observation.

$h := 1$ does establish $K(h{=}1)$

$h :\in \{0,1\}$ does not establish $K(h{=}1)$

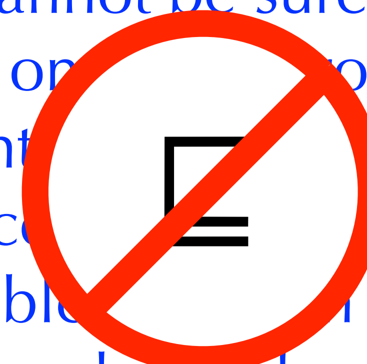
$h :\in \{0,1\}$ does establish $P(h{=}1)$

$h :\in \{0,1\}$ does not establish $P(h{=}2)$

$h :\in \{0,1\};\ v :\in \{0,1\}$ does establish $P(h{=}1)$

$h :\in \{0,1\};\ v := h$ does not establish $P(h{=}1)$

You cannot be sure that,
on … ogram
fragment … you
will c … o be
possibl … what
you have observed.



Essential principles for scaling up: some refinements must be banned

specification

$$\textbf{skip} \quad \not\sqsubseteq \quad \textbf{if}\ h{=}0\ \textbf{then skip else skip fi}$$

implementation

Essential principles for scaling up: some refinements must be banned

$$\textbf{skip} \quad \not\sqsubseteq \quad \textbf{if } h{=}0 \textbf{ then skip else skip fi}$$

$$\sqsubseteq \quad \textbf{if } h{=}0 \textbf{ then}$$

$$|[\textbf{vis } v \cdot \; v := 0]|$$

$$\textbf{else}$$

$$|[\textbf{vis } v \cdot \; v := 1]|$$

$$\textbf{fi}$$

|[•••]| declares local variables

Essential principles for scaling up: some refinements must be banned

$$P() \quad \not\sqsubseteq \quad \textbf{if}\ h{=}0\ \textbf{then}\ P()\ \textbf{else}\ P()\ \textbf{fi}$$

Essential principles for scaling up: some refinements must be banned

$$P() \quad \not\sqsubseteq \quad \mathbf{if}\ h{=}0\ \mathbf{then}\ P()\ \mathbf{else}\ P()\ \mathbf{fi}$$

$$P() \sqsubseteq P_0()$$

Essential principles for scaling up: some refinements must be banned

$$P() \quad \not\sqsubseteq \quad \textbf{if } h{=}0 \textbf{ then } P() \textbf{ else } P() \textbf{ fi}$$

$$P() \sqsubseteq P_0()$$

$$P() \sqsubseteq P_1()$$

Essential principles for scaling up: some refinements must be banned

$$P() \quad \not\sqsubseteq \quad \mathbf{if}\ h{=}0\ \mathbf{then}\ P()\ \mathbf{else}\ P()\ \mathbf{fi}$$

$$P() \sqsubseteq P_0()$$

$$P() \sqsubseteq P_1()$$

These procedure bodies could be in a separate module, their source code hundreds of pages away.

Essential principles for scaling up: some refinements must be banned

$$P() \quad \not\sqsubseteq \quad \textbf{if } h{=}0 \textbf{ then } P_0() \textbf{ else } P_1() \textbf{ fi}$$

$$P() \sqsubseteq P_0()$$

$$P() \sqsubseteq P_1()$$

These procedure bodies could be in a separate module, their source code hundreds of pages away.

Essential principles for scaling up: some refinements must be banned

$$P() \quad \not\sqsubseteq \quad \textbf{if } h{=}0 \textbf{ then } P_0() \textbf{ else } P_1() \textbf{ fi}$$

$$P() \sqsubseteq P_0()$$

$$P() \sqsubseteq P_1()$$

These procedure bodies could be in a separate module, their source code hundreds of pages away.

Essential principles for scaling up: *most refinements must be retained*

...otherwise we could just ban them all

- All refinements in involving only visible variables.
- All equalities in which no hiddens are assigned to visibles.
- Substitution of equals for equals, in any context.
- All structural refinements based on operators' general properties.

$v{:\in}\ \{0,1\} \ \sqsubseteq\ v{:=}\ 0$	Allowed: visible only.
$h{:\in}\ \{0,1\} \ \not\sqsubseteq\ h{:=}\ 0$	Not allowed: not equality.
$v{:\in}\ \{0,1\} \ =\ v{:\in}\ \{h,1\text{-}h\}$	Allowed: equals for equals.
$(v{:=}\ h) \sqcap (v{:=}\ 1\text{-}h) \ \sqsubseteq\ v{:=}\ h$	Allowed: structural.
$v{:\in}\ \{h,1\text{-}h\} \ \not\sqsubseteq\ (v{:=}\ h) \sqcap (v{:=}\ 1\text{-}h)$	Not allowed: hidden assigned to visible.

Outcome of logical analysis of refinement

... most refinement are retained

... characterise (most of) these in a clear way

... exclude some refinements (as few as possible)

Flying high:

algebra does it without logic

“Here’s a little refinement
I prepared earlier.”

Flying high:

algebra does it without logic

Algebraic source-level reasoning
An old example

An "assertion statement" checks an (important) predicate, and halts program execution if it does not hold:

$$\textbf{assert}\ \ pred$$

Algebraic source-level reasoning
An old example

$\mathbf{assert}\ \mathit{pred}$

is just $\mathbf{if}\ \neg \mathit{pred}\ \mathbf{then}$  $\mathbf{fi}$

Algebraic source-level reasoning
An old example

$\textbf{assert } \textit{pred}$

is just $\quad$ $\textbf{if } \neg \textit{pred} \textbf{ then}$  $\textbf{fi}$

$$\textbf{assert } \textit{pre};\ \textit{prog} \;\sqsubseteq\; \textit{prog};\ \textbf{assert } \textit{post}$$

If *pre* holds beforehand then executing *prog* will establish *post,* if termination occurs: in logic {*pre*} *prog* {*post*}.

Algebraic source-level reasoning
A new example

Algebraic source-level reasoning
A new example: *visible* and *hidden* variables

reveal *expression*

Algebraic source-level reasoning
A new example: *visible* and *hidden* variables

$$\textbf{reveal}\ \ expression$$

Shorthand for **assert** *pre*

$$\{pre\}\ prog \quad \sqsubseteq \quad \textbf{reveal}\ expr;\ \ prog$$

If *pre* holds beforehand, then executing *prog* might reveal the initial value of *expr*.

Algebraic source-level reasoning
A new example: *visible* and *hidden* variables

reveal *expression*

$$\{pre\}\ prog \sqsubseteq \mathbf{reveal}\ expr;\ prog$$
$$\{pre\}\ prog \sqsubseteq prog;\ \mathbf{reveal}\ expr$$

If *pre* holds beforehand, then executing *prog* might reveal the initial value of *expr*.
final

Algebraic source-level reasoning over *visible* and *hidden* variables

$$\textbf{reveal}\ \mathit{expression}$$

$$\{v{\neq}0\}\ v{:=}\ v*h \quad \sqsubseteq \quad \textbf{reveal}\ h;\ v{:=}\ v*h$$

If *pre* holds beforehand, then executing *prog* might reveal the initial value of *expr*.

All variables here are natural numbers.

Algebraic source-level reasoning over *visible* and *hidden* variables

$$\textbf{reveal}\ \textit{expression}$$

$$\{v{\neq}0\}\ v{:=}\ v*h \quad \sqsubseteq \quad \textbf{reveal}\ h;\ v{:=}\ v*h$$

If *pre* holds beforehand, then executing *prog* might reveal the initial value of *expr*.

A calculus of revelations

(1) Replace *E* with *F*.

$$\{pre\}\ \mathbf{reveal}\ E \quad \sqsubseteq \quad \mathbf{reveal}\ F$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

A calculus of revelations

(1) Replace *E* with *F*.

$$\{pre\}\ \mathbf{reveal}\ E \quad \sqsubseteq \quad \mathbf{reveal}\ F$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$\{pre\}\ \mathbf{reveal}\ E \quad \sqsubseteq \quad \mathbf{reveal}\ F$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$\{pre\}\ \textbf{reveal}\ E \quad \sqsubseteq \quad \textbf{reveal}\ F$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

Note that ***reveal*** *(empty)* is just **skip**.

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$\{pre\}\ \mathbf{reveal}\ E \quad \sqsubseteq \quad \mathbf{reveal}\ F$$
$$\{h{=}0\}\ \mathbf{skip} \quad \sqsubseteq \quad \mathbf{reveal}\ h$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

Note that ***reveal*** *(empty)* is just **skip**.

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$\{pre\}\ \mathbf{reveal}\ E \quad \sqsubseteq \quad \mathbf{reveal}\ F$$
$$\{h{=}0\} \quad \sqsubseteq \quad \mathbf{reveal}\ h$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

Note that ***reveal*** *(empty)* is just **skip**.

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$
\begin{array}{lcl}
\{pre\}\ \textbf{reveal}\ E & \sqsubseteq & \textbf{reveal}\ F \\
\{h{=}0\} & \sqsubseteq & \textbf{reveal}\ h \\
\textbf{reveal}\ h & \sqsubseteq & \textbf{reveal}\ h{\ominus}1
\end{array}
$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

Note that ***reveal*** *(empty)* is just **skip**.

$h{-}1\ \textbf{max}\ 0$

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$
\begin{array}{lcl}
\{pre\}\ \textbf{reveal}\ E & \sqsubseteq & \textbf{reveal}\ F \\
\{h=0\} & \sqsubseteq & \textbf{reveal}\ h \\
\textbf{reveal}\ h & \sqsubseteq & \textbf{reveal}\ h\ominus 1 \\
\textbf{reveal}\ h\ominus 1 & \not\sqsubseteq & \textbf{reveal}\ h
\end{array}
$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,
for some context $\mathbb{F}$ with only visible variables.

Note that ***reveal*** *(empty)* is just **skip**.

A calculus of revelations

(1) Replace *E* with *F*: examples.

$$\begin{array}{lcl} \{pre\}\ \textbf{reveal}\ E & \sqsubseteq & \textbf{reveal}\ F \\ \{h{=}0\} & \sqsubseteq & \textbf{reveal}\ h \\ \textbf{reveal}\ h & \sqsubseteq & \textbf{reveal}\ h{\ominus}1 \\ \textbf{reveal}\ h{\ominus}1 & \not\sqsubseteq & \textbf{reveal}\ h \end{array}$$

$$\{h{>}0\}\ \textbf{reveal}\ h{\ominus}1 \quad \sqsubseteq \quad \textbf{reveal}\ h$$

... provided truth of *pre* implies that $F = \mathbb{F}(E)$,

for some context $\mathbb{F}$ with only visible variables.

Note that ***reveal*** *(empty)* is just **skip**.

A calculus of revelations

(2) Combine *E* with *F*.

$$\textbf{reveal } E;\ \textbf{reveal } F$$
$$= \quad \textbf{reveal } (E, F)$$

A calculus of revelations

(1+2=3) Combine *E* with *F*; replace *F* with *F′*; separate *E* and *F′*.

$$\begin{aligned}\textbf{reveal}\ E;\ \textbf{reveal}\ F \\ &= \textbf{reveal}\ (E, F)\end{aligned}$$

$$\textbf{reveal}\ x{\oplus}y;\ \textbf{reveal}\ y{\oplus}z$$

A calculus of revelations

(1+2=3) Combine *E* with *F*; replace *F* with *F′*; separate *E* and *F′*.

$$\textbf{reveal } E;\ \textbf{reveal } F$$
$$= \quad \textbf{reveal } (E, F)$$

$$\textbf{reveal } x \oplus y;\ \textbf{reveal } y \oplus z$$

addition mod 2 or, equivalently, exclusive-or

A calculus of revelations

(1+2=3) Combine *E* with *F*; replace *F* with *F′*; separate *E* and *F′*.

$$\textbf{reveal } E;\ \textbf{reveal } F$$
$$= \quad \textbf{reveal } (E, F)$$

$$\textbf{reveal } x{\oplus}y;\ \textbf{reveal } y{\oplus}z$$
$$= \quad \textbf{reveal } (x{\oplus}y, y{\oplus}z)$$

A calculus of revelations

(1+2=3) Combine *E* with *F*; replace *F* with *F′*; separate *E* and *F′*.

$$
\begin{aligned}
&\mathbf{reveal}\ E;\ \mathbf{reveal}\ F \\
&\qquad = \quad \mathbf{reveal}\ (E, F)
\end{aligned}
$$

$$
\begin{aligned}
& \mathbf{reveal}\ x{\oplus}y;\ \mathbf{reveal}\ y{\oplus}z \\
= \quad & \mathbf{reveal}\ (x{\oplus}y, y{\oplus}z) \\
= \quad & \mathbf{reveal}\ (x{\oplus}y, x{\oplus}z)
\end{aligned}
$$

A calculus of revelations

(1+2=3) Combine *E* with *F*; replace *F* with *F′*; separate *E* and *F′*.

$$\begin{aligned} &\textbf{reveal } E;\ \textbf{reveal } F \\ &\qquad = \quad \textbf{reveal } (E, F) \end{aligned}$$

$$\begin{aligned} & & \textbf{reveal } x{\oplus}y;\ \textbf{reveal } y{\oplus}z \\ &= & \textbf{reveal } (x{\oplus}y, y{\oplus}z) \\ &= & \textbf{reveal } (x{\oplus}y, x{\oplus}z) \\ &= & \textbf{reveal } x{\oplus}y;\ \textbf{reveal } x{\oplus}z \end{aligned}$$

A calculus of revelations

Having an explicit **reveal** command simplifies the algebra considerably, and focuses attention —where desired— on the pure security properties.

A calculus of revelations

Having an explicit **reveal** command simplifies the algebra considerably, and focuses attention —where desired— on the pure security properties.

$$\textbf{reveal}\ E \quad = \quad |[\ \textbf{vis}\ v \cdot v{:=}\,E\ ]|$$

A calculus of revelations

Having an explicit **reveal** command simplifies the algebra considerably, and focuses attention —where desired— on the pure security properties.

$$\textbf{reveal}\ E \quad = \quad |[\ \textbf{vis}\ v \cdot v{:=}\,E\ ]|$$

Previous approach: harder to manipulate.

A calculus of revelations

$$wp.(\textbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)$$

A calculus of revelations

$wp.(\mathbf{reveal}\ (h\ \mathbf{mod}\ \mathbf{2})).P(h{=}3)$

$$wp.(\mathbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)$$

A calculus of revelations

$$
\begin{array}{ll}
 & wp.(\textbf{reveal}\ (h\ \textbf{mod 2})).P(h{=}3) \\
= & P((h\ \textbf{mod 2}){=}(h_0\ \textbf{mod 2}) \quad \wedge \quad h{=}3) \\
= & P((h_0\ \textbf{mod 2}){=}1 \quad \wedge \quad h{=}3) \\
= & (h\ \textbf{mod 2}){=}1 \quad \wedge \quad P(h{=}3) \\
= & \textbf{odd}\ h \wedge P(h{=}3)\ .
\end{array}
$$

$$
wp.(\textbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)
$$

A calculus of revelations

$$\begin{array}{ll} & wp.(\textbf{reveal}\ (h\ \textbf{mod 2})).P(h{=}3) \\ = & P((h\ \textbf{mod 2}){=}(h_0\ \textbf{mod 2}) \quad \wedge \quad h{=}3) \\ = & P((h_0\ \textbf{mod 2}){=}1 \quad \wedge \quad h{=}3) \\ = & (h\ \textbf{mod 2}){=}1 \quad \wedge \quad P(h{=}3) \\ = & \textbf{odd}\ h \wedge P(h{=}3)\ . \end{array}$$

Does $h{:}\in \{1{,}3\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? Yes

$$wp.(\textbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)$$

A calculus of revelations

$$
\begin{aligned}
& wp.(\textbf{reveal}\ (h\ \textbf{mod 2})).P(h{=}3) \\
= \quad & P((h\ \textbf{mod 2}){=}(h_0\ \textbf{mod 2}) \quad \wedge \quad h{=}3) \\
= \quad & P((h_0\ \textbf{mod 2}){=}1 \quad \wedge \quad h{=}3) \\
= \quad & (h\ \textbf{mod 2}){=}1 \quad \wedge \quad P(h{=}3) \\
= \quad & \textbf{odd}\ h \wedge P(h{=}3)\ .
\end{aligned}
$$

Does $h{:\in}\ \{1,3\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? Yes

Does $h{:\in}\ \{1,5\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? No

$$wp.(\textbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)$$

A calculus of revelations

$$
\begin{aligned}
& wp.(\textbf{reveal}\ (h\ \textbf{mod}\ \textbf{2})).P(h{=}3) \\
= \quad & P((h\ \textbf{mod}\ \textbf{2}){=}(h_0\ \textbf{mod}\ \textbf{2}) \quad \wedge \quad h{=}3) \\
= \quad & P((h_0\ \textbf{mod}\ \textbf{2}){=}1 \quad \wedge \quad h{=}3) \\
= \quad & (h\ \textbf{mod}\ \textbf{2}){=}1 \quad \wedge \quad P(h{=}3) \\
= \quad & \textbf{odd}\ h \wedge P(h{=}3)\ .
\end{aligned}
$$

Does $h{:\in}\ \{1,3\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? Yes

Does $h{:\in}\ \{1,5\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? No

Does $h{:\in}\ \{2,3\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? No

$$wp.(\textbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)$$

A calculus of revelations

$$
\begin{array}{ll}
 & wp.(\textbf{reveal}\ (h\ \textbf{mod}\ \textbf{2})).P(h{=}3) \\
= & P((h\ \textbf{mod}\ \textbf{2}){=}(h_0\ \textbf{mod}\ \textbf{2}) \quad \wedge \quad h{=}3) \\
= & P((h_0\ \textbf{mod}\ \textbf{2}){=}1 \quad \wedge \quad h{=}3) \\
= & (h\ \textbf{mod}\ \textbf{2}){=}1 \quad \wedge \quad P(h{=}3) \\
= & \textbf{odd}\ h \wedge P(h{=}3)\ .
\end{array}
$$

Does $h{:}{\in}\ \{1,3\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? Yes

Does $h{:}{\in}\ \{1,5\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? No

Does $h{:}{\in}\ \{2,3\}$; **reveal** (h **mod 2**) establish $P(h{=}3)$? No

Does $(h{:=}\ 1 \sqcap h{:=}\ 3)$; **reveal** ($h$ **mod 2**) establish $P(h{=}3)$?

No

$$wp.(\textbf{reveal}\ E).P\phi \quad = \quad P(E{=}E_0 \wedge \phi)$$

Flying low:

the logic *justifies* the algebra, creating <u>beforehand</u> a library of very small but reusable identities

"Here's *how* I prepared that little refinement earlier."

Flying low:

the logic *justifies* the algebra, creating <u>beforehand</u> a library of very small but reusable identities

The Encryption Lemma: a piece of algebraic Lego

$$\textbf{hid } h \cdot$$

$$|[\ \textbf{hid } h';\ h' {:\in} \{0,1\};\ \textbf{reveal } h \oplus h'\ ]|$$

Does this program fragment reveal anything about *h*?

The Encryption Lemma: a piece of algebraic Lego

$$\textbf{hid}\ h\ \cdot$$

$$|[\ \textbf{hid}\ h';\ \ h'{:}{\in}\ \{0,1\};\ \ \textbf{reveal}\ h{\oplus}h'\ ]|$$

Does this program fragment reveal anything about *h*?

The context of declared variables

The Encryption Lemma: a piece of algebraic Lego

$$\textbf{hid } h \cdot$$

$$|[\ \textbf{hid } h';\ h' :\in \{0,1\};\ \textbf{reveal } h \oplus h'\ ]|$$

$$\stackrel{?}{\sqsupseteq} \quad \textbf{skip}$$

Does this program fragment reveal anything about *h*?

The context of declared variables

The *wp* semantics validates the *construction* of the Lego

Identity	$wp.\mathbf{skip}.\Psi$	$\widehat{=}$	Ψ
Revelation	$wp.(\mathbf{reveal}\ E).\Psi$	$\widehat{=}$	$[\Downarrow E_0{=}E]\Psi$
Assign to visible	$wp.(v{:=}\ E).\Psi$	$\widehat{=}$	$[e\backslash E]\,[\Downarrow e{=}E]\,[v\backslash e]\,\Psi$
Choose visible	$wp.(v{:\in}\ E).\Psi$	$\widehat{=}$	$(\forall e{:}\ E \cdot [\Downarrow e{\in}E]\,[v\backslash e]\,\Psi)$
Assign to hidden	$wp.(h{:=}\ E).\Psi$	$\widehat{=}$	$[h{\leftarrow}E]\,\Psi$
Choose hidden	$wp.(h{:\in}\ E).\Psi$	$\widehat{=}$	$(\forall e{:}\ E \cdot [h\backslash e]\,[h{\Leftarrow}E]\,\Psi)$

Demonic choice	$wp.(\mathrm{S1} \sqcap \mathrm{S2}).\Psi$	$\widehat{=}$	$wp.\mathrm{S1}.\Psi\ \wedge\ wp.\mathrm{S2}.\Psi$
Composition	$wp.(\mathrm{S1};\mathrm{S2}).\Psi$	$\widehat{=}$	$wp.\mathrm{S1}.(wp.\mathrm{S2}.\Psi)$
Conditional	$wp.(\mathbf{if}\ E\ \mathbf{then}\ \mathrm{S1}\ \mathbf{else}\ \mathrm{S2}\ \mathbf{fi}).\Psi$		

$$\widehat{=}\quad E \Rightarrow [\Downarrow E]\,wp.\mathrm{S1}.\Psi\ \wedge\ \neg E \Rightarrow [\Downarrow \neg E]\,wp.\mathrm{S2}.\Psi$$

Declare visible	$wp.(\textsc{vis}\ v).\Psi$	$\widehat{=}$	$(\forall e \cdot [v\backslash e]\,\Psi)$
Declare hidden	$wp.(\textsc{hid}\ h).\Psi$	$\widehat{=}$	$(\forall e \cdot [h{\leftarrow}e]\,\Psi)$

In this case...

$$wp.|[\ \mathbf{hid}\ h';\ \ h' \!:\in \{0,1\};\ \ \mathbf{reveal}\ h \oplus h'\ ]|.(P\Psi)$$

In this case...

$$wp.|[\ \mathbf{hid}\ h';\ h'{:\in}\ \{0,1\};\ \mathbf{reveal}\ h{\oplus}h'\ ]|.(P\Psi)$$

$$\ldots$$

$$= \quad (\forall e \in \{0,1\} \cdot [h\backslash e][h{\Leftarrow}\{0,1\}]$$
$$P(h{\oplus}h' = h_0{\oplus}h'_0 \wedge \Psi))$$
$$= \quad (\forall e \in \{0,1\} \cdot [h\backslash e]$$
$$P(\exists h \in \{0,1\} \cdot h{\oplus}h' = h_0{\oplus}h'_0 \wedge \Psi))$$
$$= \quad (\forall e \in \{0,1\} \cdot [h\backslash e]P\Psi)$$

In this case...

$$wp.|[\ \textbf{hid}\ h';\ h' :\in \{0,1\};\ \textbf{reveal}\ h \oplus h'\ ]|.(P\Psi)$$

$$\ldots$$

$$= \quad (\forall e \in \{0,1\} \cdot [h \backslash e][h \Leftarrow \{0,1\}]$$
$$P(h \oplus h' = h_0 \oplus h'_0 \wedge \Psi))$$
$$= \quad (\forall e \in \{0,1\} \cdot [h \backslash e]$$
$$P(\exists h \in \{0,1\} \cdot h \oplus h' = h_0 \oplus h'_0 \wedge \Psi))$$
$$= \quad (\forall e \in \{0,1\} \cdot [h \backslash e] P\Psi)$$

$$= \quad P\Psi$$
$$= \quad wp.\textbf{skip}.(P\Psi)$$

The Encryption Lemma:
a piece of algebraic Lego

$$\boxed{\textbf{hid}\ h\ \cdot}$$

$$|[\ \textbf{hid}\ h';\ \ h'{:\in}\ \{0,1\};\ \ \textbf{reveal}\ h{\oplus}h'\ ]|$$

$$=\quad \textbf{skip}$$

Does this program fragment reveal anything about *h*?

The Encryption Lemma:
a piece of algebraic Lego

$$\boxed{\textbf{hid}\ h\ \cdot}$$

$$|[\ \textbf{hid}\ h';\ \ h'{:}\!\in \{0,1\};\ \ \textbf{reveal}\ h{\oplus}h'\ ]|$$

$$=\quad \textbf{skip}$$

Does this program fragment reveal anything about *h*?
No — it reveals nothing at all.

The Encryption Lemma: a piece of algebraic Lego

"Here's a little refinement I prepared earlier."

$$\textbf{hid}\ h\ \cdot$$

$$|[\ \textbf{hid}\ h';\ \ h' {:\in}\ \{0,1\};\ \ \textbf{reveal}\ h{\oplus}h'\ ]|$$

$$=\quad \textbf{skip}$$

Does this program fragment reveal anything about *h*?

No — it reveals nothing at all.

The Encryption Lemma: a piece of algebraic Lego

$$|[\ \textbf{hid}\ h';\ \ h'{:}{\in}\ \{0,1\};\ \ \textbf{reveal}\ h{\oplus}h'\ ]| \\ =\ \ \textbf{skip}$$

$\textbf{hid}\ h\ \cdot$

$$|[\ \textbf{hid}\ h';\ \ h'{:}{\in}\ \{0,1\};\ \ \textbf{reveal}\ h{\oplus}h'\ ]|$$

$$=\ \ \textbf{skip}$$

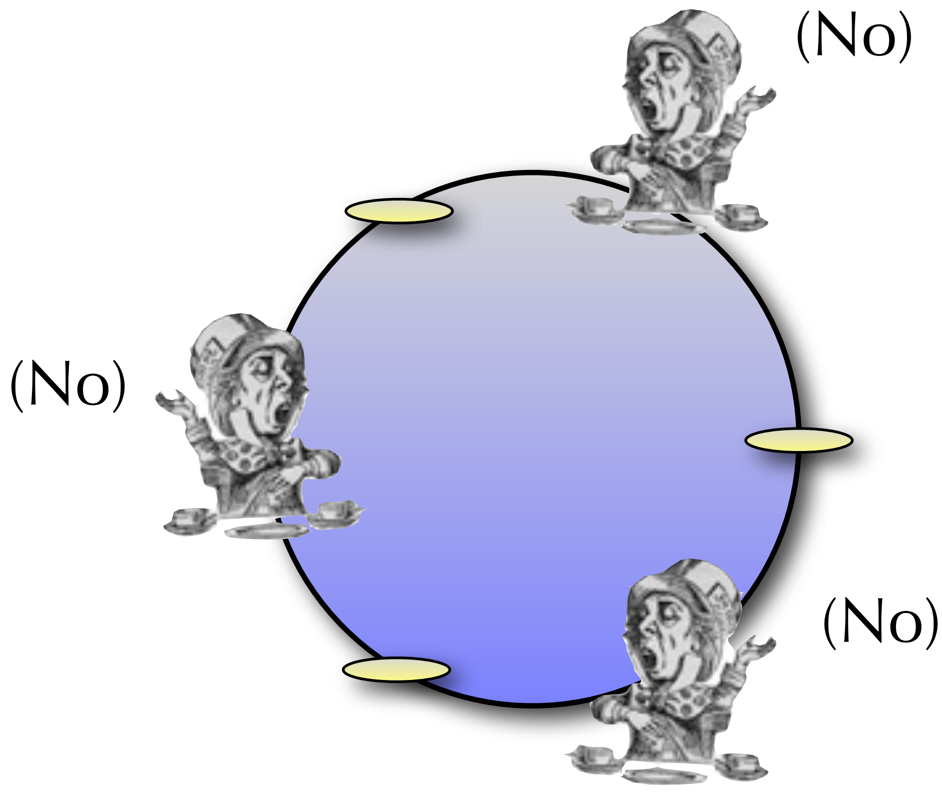
Does this program fragment reveal anything about *h*?
No — it reveals nothing at all.

The Dining Cryptographers *algebraically*

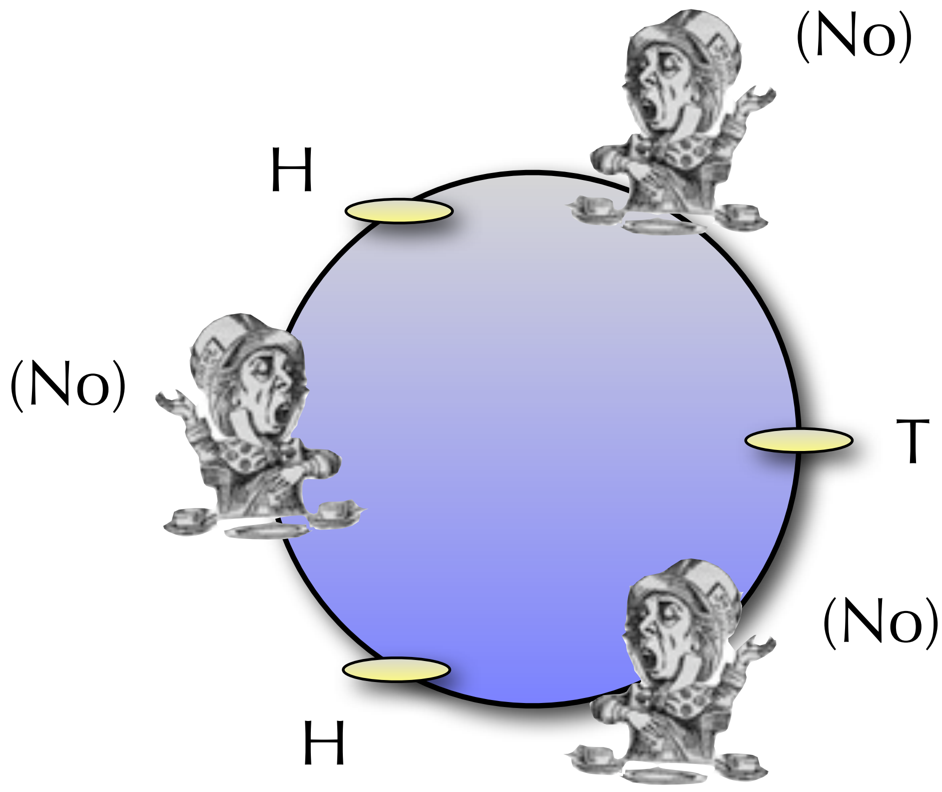
The Dining Cryptographers *algebraically*

The Dining Cryptographers

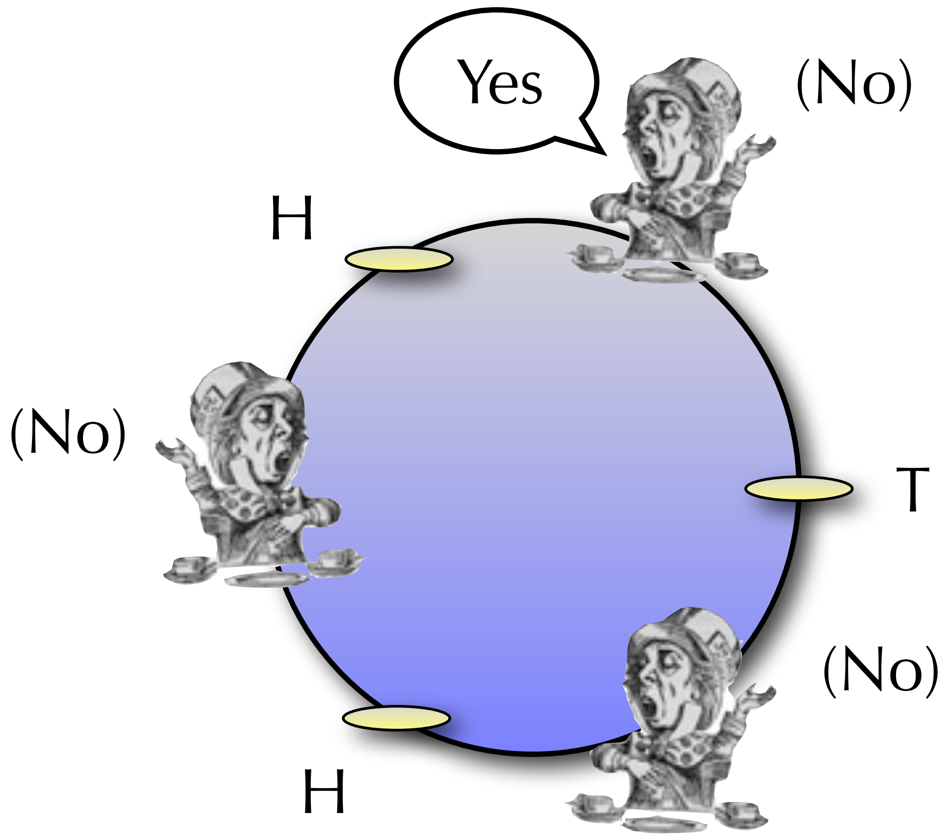
The Dining Cryptographers



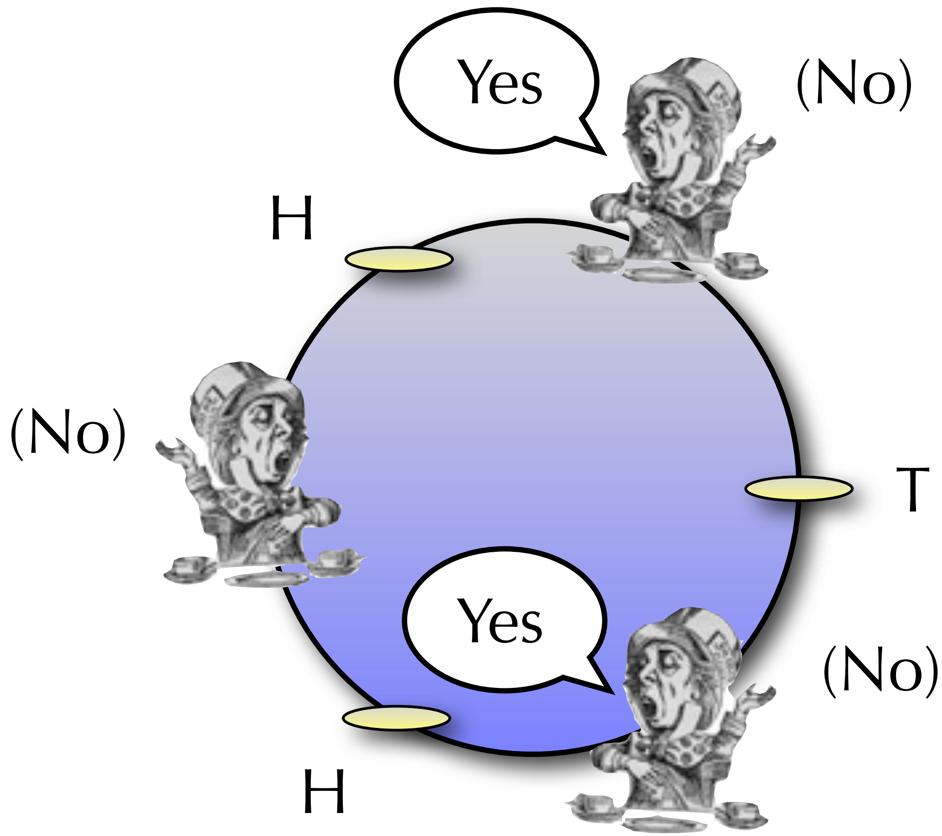
The Dining Cryptographers



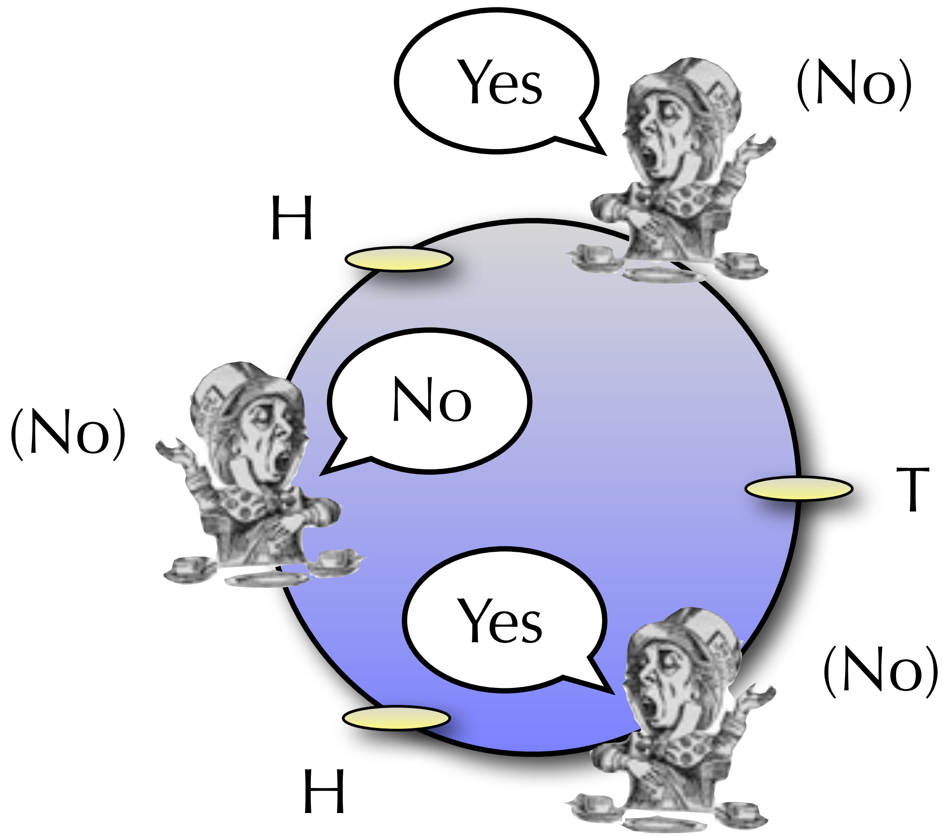
The Dining Cryptographers



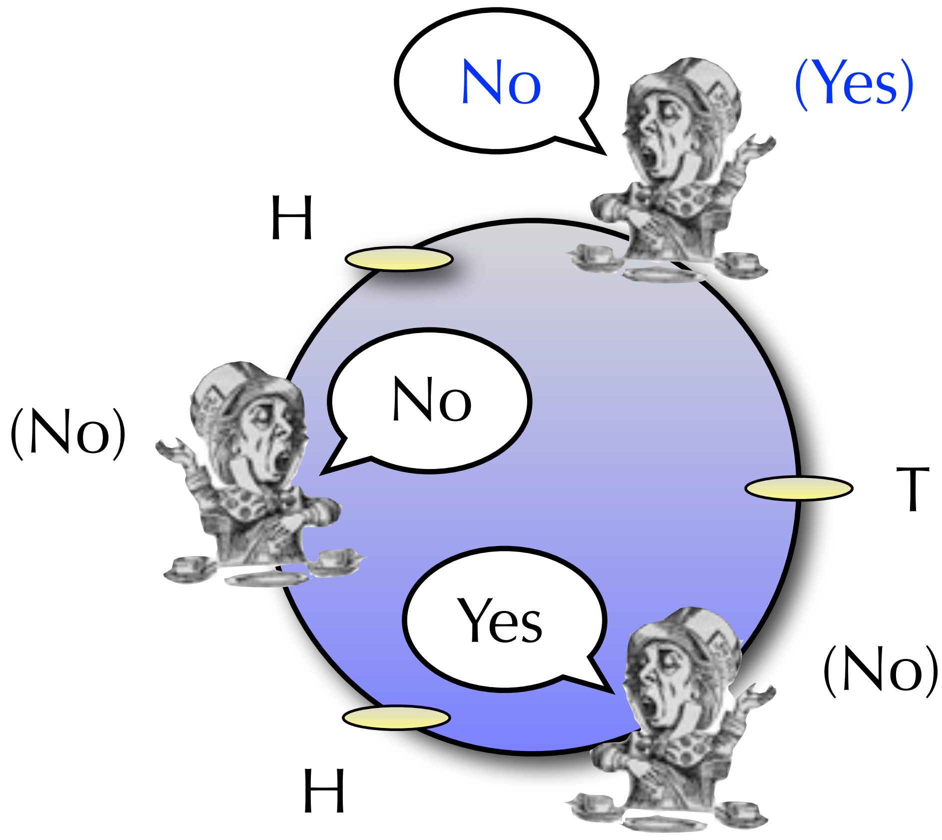
The Dining Cryptographers



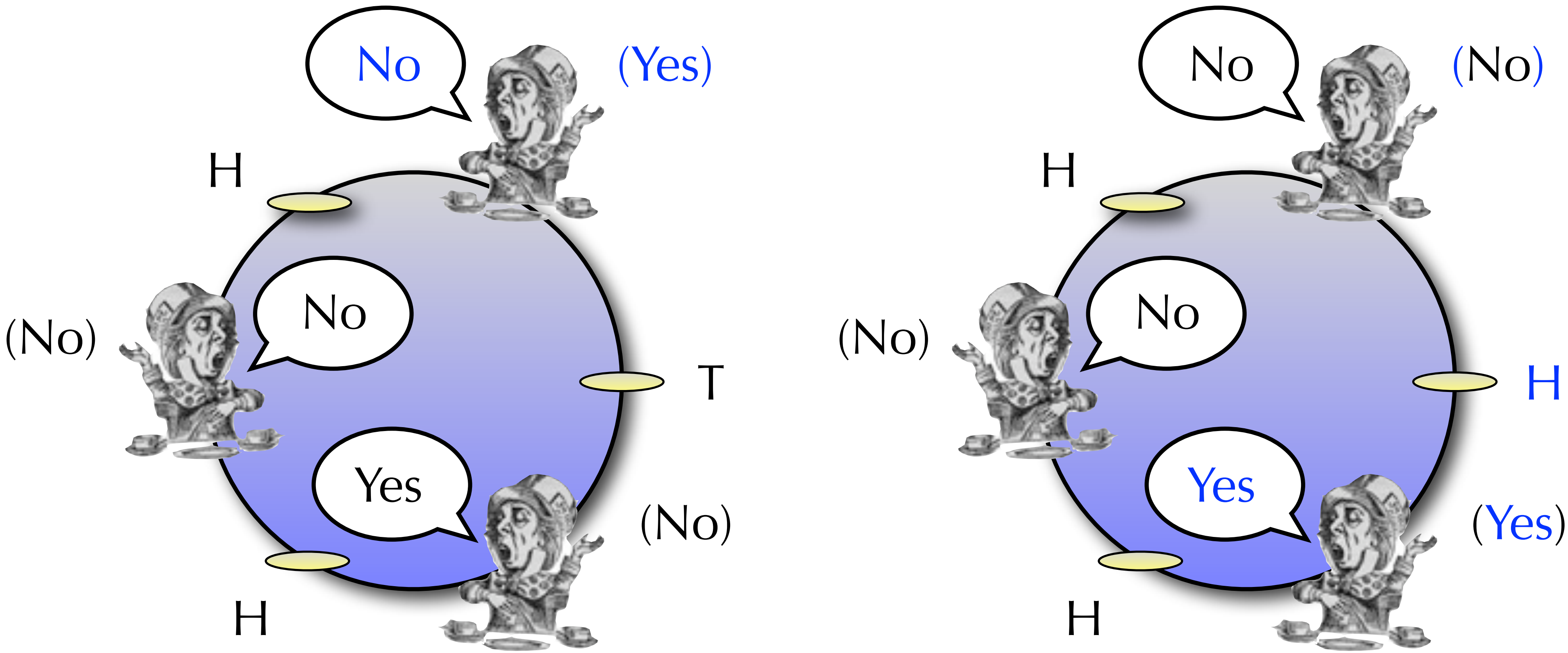
The Dining Cryptographers



The Dining Cryptographers



The Dining Cryptographers



Loop Lego for the Dining Cryptographers

— Assume l is already set.
$r :\in \{0, 1\};$
$\mathbf{reveal}\ l \oplus x \oplus r$

All variables are "Boolean," with *l* and *r* being visible.
Variable *x* is a hidden Boolean, and $\oplus$ is "exclusive or."

Loop Lego for the Dining Cryptographers

Rename the *r* variable to *m for "middle,"* and add a second component.

$$\text{— Assume } l \text{ is already set.}$$
$$m :\in \{0, 1\};$$
$$\textbf{reveal } l \oplus x \oplus m$$

Loop Lego
for the Dining Cryptographers

Rename the *r* variable to *m for "middle,"*
and add a second component.

— Assume l is already set.
$m{:\in}\ \{0,1\};$
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r{:\in}\ \{0,1\};$
reveal $m \oplus y \oplus r$

Loop Lego
for the Dining Cryptographers

Hide the middle variable.

— Assume l is already set.
$m{:}\in \{0,1\};$
$\textbf{reveal}\ l \oplus x \oplus m$

— Assume m is already set.
$r{:}\in \{0,1\};$
$\textbf{reveal}\ m \oplus y \oplus r$

Loop Lego for the Dining Cryptographers

Hide the middle variable.

$$
\begin{array}{l}
[\![\ \textbf{hid}\ m \cdot \\
\quad \text{— Assume } l \text{ is already set.} \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\\
\quad \text{— Assume } m \text{ is already set.} \\
\quad r{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]\!]
\end{array}
$$

Loop Lego for the Dining Cryptographers

Squash up.

$|[$ **hid** $m\ \cdot$

— Assume l is already set.

$m{:}\in \{0,1\};$

reveal $l \oplus x \oplus m$

— Assume m is already set.

$r{:}\in \{0,1\};$

reveal $m \oplus y \oplus r$

$]|$

Loop Lego
for the Dining Cryptographers

Squash up.

$$
\begin{array}{l}
|[\ \textbf{hid}\ m \cdot \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad r{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Bring the **reveal** commands
next to each other.

$$
\begin{array}{l}
|[\ \mathbf{hid}\ m \cdot \\
\quad m{:}\in \{0,1\}; \\
\quad \mathbf{reveal}\ l \oplus x \oplus m \\
\quad r{:}\in \{0,1\}; \\
\quad \mathbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Bring the **reveal** commands next to each other.

$$\begin{array}{l} |[\ \textbf{hid}\ m \cdot \\ \quad m{:}\in \{0,1\}; \\ \quad \textbf{reveal}\ l \oplus x \oplus m \\ \quad r{:}\in \{0,1\}; \\ \quad \textbf{reveal}\ m \oplus y \oplus r \\ ]| \end{array}$$

$$\begin{array}{l} |[\ \textbf{hid}\ m \cdot \\ \quad m{:}\in \{0,1\}; \\ \quad \textbf{reveal}\ l \oplus x \oplus m \\ \quad r{:}\in \{0,1\}; \\ \quad \textbf{reveal}\ m \oplus y \oplus r \\ ]| \end{array}$$

Loop Lego

Bring the **reveal** commands next to each other.

$$
\begin{array}{l}
|[\ \textbf{hid}\ m\ \cdot \\
\quad r :\in \{0,1\}; \\
\quad m :\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

$$
\begin{array}{l}
|[\ \textbf{hid}\ m\ \cdot \\
\quad m :\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad r :\in \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Use *revelation algebra* to alter the expression in the second one.

$$
\begin{array}{l}
|[\ \textbf{hid}\ m \cdot \\
\quad r{:}\in \{0,1\}; \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Use *revelation algebra* to alter the expression in the second one.

$$
\begin{array}{l}
|[\ \textbf{hid}\ m\ \cdot \\
\quad r{:}\in \{0,1\}; \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

$$
\begin{array}{l}
|[\ \textbf{hid}\ m\ \cdot \\
\quad r{:}\in \{0,1\}; \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Use *revelation algebra* to alter the expression in the second one.

$$
\begin{array}{l}
|[\ \textbf{hid}\ m \cdot \\
\quad r{:}\in \{0,1\}; \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad \textbf{reveal}\ l \oplus x \oplus y \oplus r \\
]|
\end{array}
$$

$$
\begin{array}{l}
|[\ \textbf{hid}\ m \cdot \\
\quad r{:}\in \{0,1\}; \\
\quad m{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
\quad \textbf{reveal}\ m \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Move the non-*m*-using commands
out of the local scope.

$$
\begin{array}{l}
|[\ \mathbf{hid}\ m \cdot \\
\quad r{:\in}\ \{0,1\}; \\
\quad m{:\in}\ \{0,1\}; \\
\quad \mathbf{reveal}\ l \oplus x \oplus m \\
\quad \mathbf{reveal}\ l \oplus x \oplus y \oplus r \\
]|
\end{array}
$$

Loop Lego

Move the non-*m*-using commands out of the local scope.

$$
\begin{aligned}
&|[\ \textbf{hid}\ m \cdot \\
&\quad r{:}\in \{0,1\}; \\
&\quad m{:}\in \{0,1\}; \\
&\quad \textbf{reveal}\ l \oplus x \oplus m \\
&\quad \textbf{reveal}\ l \oplus x \oplus y \oplus r \\
&]|
\end{aligned}
$$

$$
\begin{aligned}
&|[\ \textbf{hid}\ m \cdot \\
&\quad r{:}\in \{0,1\}; \\
&\quad m{:}\in \{0,1\}; \\
&\quad \textbf{reveal}\ l \oplus x \oplus m \\
&\quad \textbf{reveal}\ l \oplus x \oplus y \oplus r \\
&]|
\end{aligned}
$$

Loop Lego

Move the non-m-using commands out of the local scope.

$$
\begin{array}{l}
r{:}\in \{0,1\};\\
|[\ \textbf{hid}\ m\ \cdot\\
\quad m{:}\in \{0,1\};\\
\quad \textbf{reveal}\ l \oplus x \oplus m\\
]|\\
\textbf{reveal}\ l \oplus x \oplus y \oplus r
\end{array}
$$

$$
\begin{array}{l}
|[\ \textbf{hid}\ m\ \cdot\\
\quad r{:}\in \{0,1\};\\
\quad m{:}\in \{0,1\};\\
\quad \textbf{reveal}\ l \oplus x \oplus m\\
\quad \textbf{reveal}\ l \oplus x \oplus y \oplus r\\
]|
\end{array}
$$

Loop Lego

Appeal to the *Encryption Lemma*.

$$
\begin{array}{l}
r :\in \{0,1\}; \\
|[\ \textbf{hid}\ m \cdot \\
\quad m :\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x \oplus m \\
]| \\
\textbf{reveal}\ l \oplus x \oplus y \oplus r
\end{array}
$$

Loop Lego

Appeal to the *Encryption Lemma*.

$$
\begin{array}{l}
r{:\in}\ \{0,1\};\\
|[\ \textbf{hid}\ m\ \cdot\\
\quad m{:\in}\ \{0,1\};\\
\quad \textbf{reveal}\ l \oplus x \oplus m\\
]|\\
\textbf{reveal}\ l \oplus x \oplus y \oplus r
\end{array}
$$

$$
\begin{array}{l}
r{:\in}\ \{0,1\};\\
|[\ \textbf{hid}\ m\ \cdot\\
\quad m{:\in}\ \{0,1\};\\
\quad \textbf{reveal}\ l \oplus x \oplus m\\
]|\\
\textbf{reveal}\ l \oplus x \oplus y \oplus r
\end{array}
$$

Loop Lego

$r{:}\in \{0,1\};$
$|[\ \textbf{hid}\ m \cdot$
$\quad m{:}\in \{0,1\};$
$\quad \textbf{reveal}\ l \oplus x \oplus m$
$]|$
$\textbf{reveal}\ l \oplus x \oplus y \oplus r$

Appeal to the *Encryption Lemma*.

$$r{:}\in \{0,1\};$$

$$\textbf{skip};$$

$$\textbf{reveal}\ l \oplus x \oplus y \oplus r$$

Loop Lego

$r{:}\in \{0,1\};$
$|[\ \mathbf{hid}\ m \cdot$
$\quad m{:}\in \{0,1\};$
$\quad \mathbf{reveal}\ l \oplus x \oplus m$
$]|$
$\mathbf{reveal}\ l \oplus x \oplus y \oplus r$

Appeal to the *Encryption Lemma*.

$r{:}\in \{0,1\};$
$\mathbf{skip};$
$\mathbf{reveal}\ l \oplus x \oplus y \oplus r$

Loop Lego

Squash up, and
recall where we started.

$$r :\in \{0, 1\};$$

$$\textbf{reveal}\ l \oplus x \oplus y \oplus r$$

Loop Lego

Squash up, and
recall where we started.

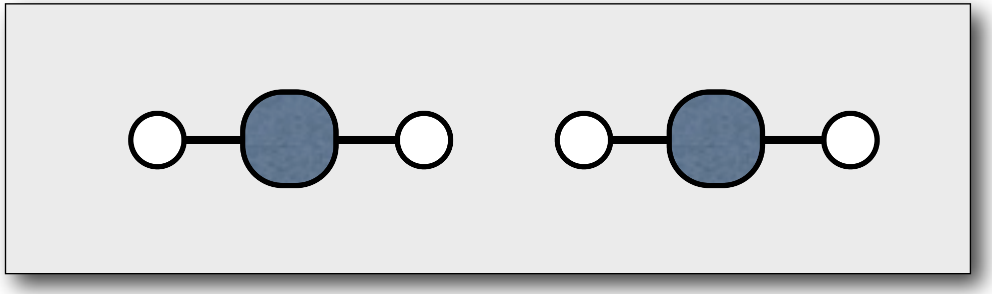
$|[$ **hid** m ·
— Assume l is already set.
$m{:}\in \{0,1\}$;
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r{:}\in \{0,1\}$;
reveal $m \oplus y \oplus r$
$]|$

— Assume l is already set.
$r{:}\in \{0,1\}$;
reveal $l \oplus x \oplus y \oplus r$

Loop Lego

Squash up, and
recall where we started.



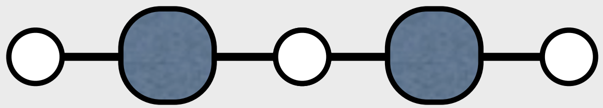
$|[$ **hid** $m \cdot$
— Assume l is already set.
$m :\in \{0, 1\};$
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r :\in \{0, 1\};$
reveal $m \oplus y \oplus r$
$]|$

— Assume l is already set.
$r :\in \{0, 1\};$
reveal $l \oplus x \oplus y \oplus r$

Loop Lego

Squash up, and
recall where we started.



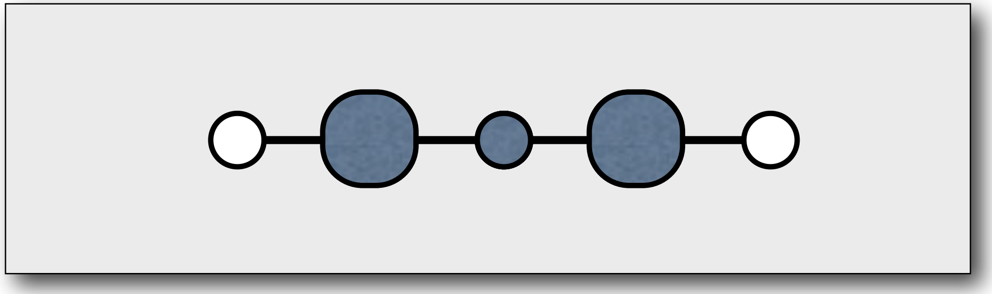
$[\![$ **hid** $m\ \cdot$
— Assume l is already set.
$m{:}\!\in \{0,1\};$
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r{:}\!\in \{0,1\};$
reveal $m \oplus y \oplus r$
$]\!]$

— Assume l is already set.
$r{:}\!\in \{0,1\};$
reveal $l \oplus x \oplus y \oplus r$

Loop Lego

Squash up, and
recall where we started.



$[\![$ **hid** $m \cdot$
— Assume l is already set.
$m{:}\in \{0,1\}$;
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r{:}\in \{0,1\}$;
reveal $m \oplus y \oplus r$
$]\!]$

— Assume l is already set.
$r{:}\in \{0,1\}$;
reveal $l \oplus x \oplus y \oplus r$

Loop Lego

Squash up, and
recall where we started.

$[\![$ **hid** $m\ \cdot$
— Assume l is already set.
$m{:}\in \{0,1\};$
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r{:}\in \{0,1\};$
reveal $m \oplus y \oplus r$
$]\!]$

— Assume l is already set.
$r{:}\in \{0,1\};$
reveal $l \oplus x \oplus y \oplus r$

Loop Lego

$[\![$ **hid** $m \cdot$
— Assume l is already set.
$m{:}\in \{0,1\}$;
reveal $l \oplus x \oplus m$

— Assume m is already set.
$r{:}\in \{0,1\}$;
reveal $m \oplus y \oplus r$
$]\!]$

This is how we will make the loop that treats an arbitrary number of cryptographers.

— Assume l is already set.
$r{:}\in \{0,1\}$;
reveal $l \oplus x[0,N) \oplus r$

Use this to abbreviate exclusive-or of that range.

Loop Lego

$|[\ \mathbf{hid}\ m\ \cdot$

— Assume l is already set.

$m{:}\in \{0,1\};$

$\mathbf{reveal}\ l \oplus x \oplus m$

— Assume m is already set.

$r{:}\in \{0,1\};$

$\mathbf{reveal}\ m \oplus y \oplus r$

$]|$

This is how we will make the loop that treats an arbitrary number of cryptographers.

— Assume l is already set.

$r{:}\in \{0,1\};$

$\mathbf{reveal}\ l \oplus x[0,N) \oplus r$

Use this to abbreviate exclusive-or of that range.

Loop Lego

$|[\ \mathbf{hid}\ m \cdot$
— Assume l is already set.
$m :\in \{0, 1\}$;
reveal $l \oplus x \oplus m$
....
— Assume m is already set.
$r :\in \{0, 1\}$;
reveal $m \oplus y \oplus r$
$]|$

This is how we will make the loop that treats an arbitrary number of cryptographers.

— Assume l is already set.
$r :\in \{0, 1\}$;
reveal $l \oplus x[0, N) \oplus r$

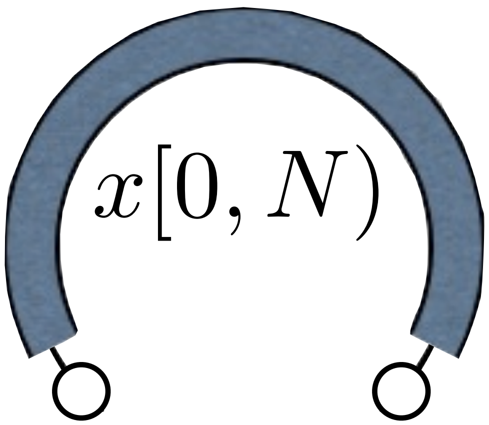
Use this to abbreviate exclusive-or of that range.

Closing the circle

$$\textbf{reveal}\ x[0, N]$$

Parity of $x[0,N]$ revealed — but nothing else.

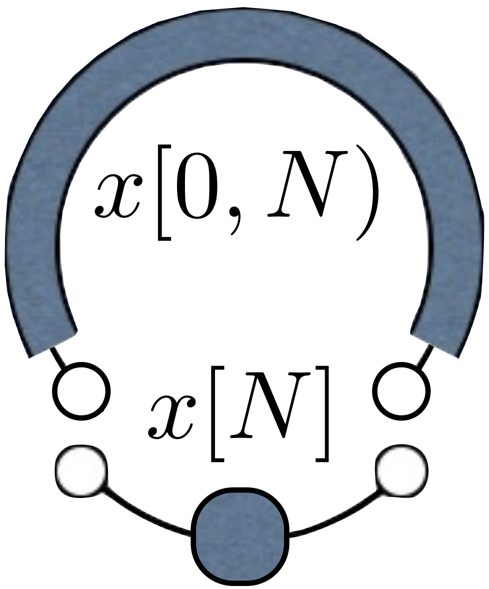
Closing the circle



$$\textbf{reveal}\ x[0, N]$$

Parity of $x[0,N]$ revealed — but nothing else.

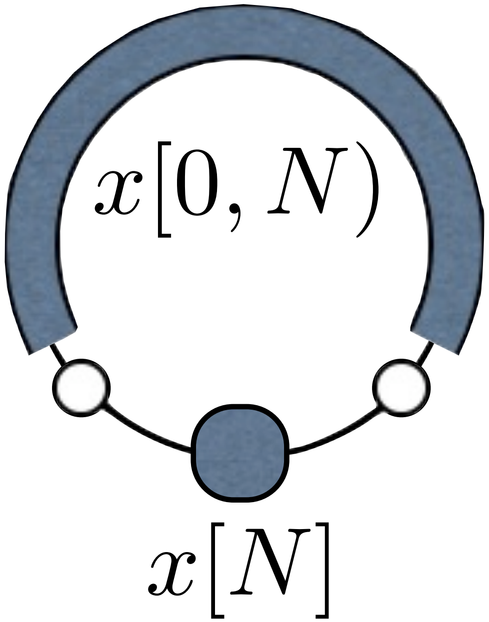
Closing the circle



$$\textbf{reveal}\ x[0, N]$$

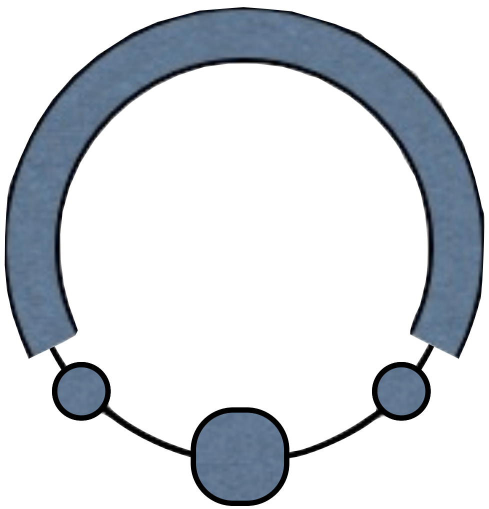
Parity of $x[0,N]$ revealed — but nothing else.

Closing the circle



Parity of $x[0,N]$ revealed — but nothing else.

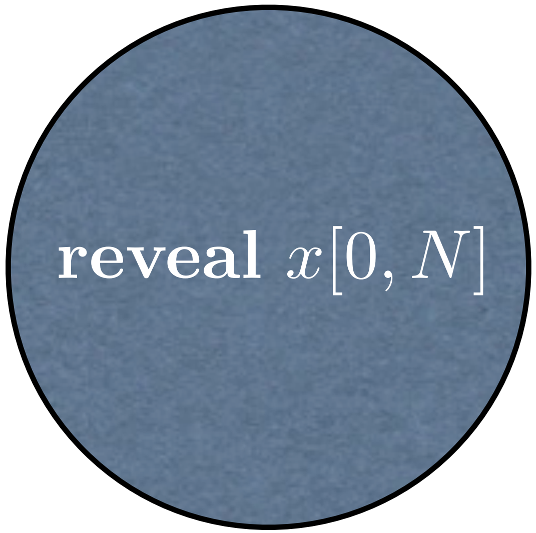
Closing the circle



$$
\begin{array}{l}
|[\ \textbf{hid}\ l, r \cdot \\
\quad l{:}\in \{0,1\}; \\
\quad r{:}\in \{0,1\}; \\
\quad \textbf{reveal}\ l \oplus x[0,N) \oplus r; \\
\quad \textbf{reveal}\ r \oplus x[N] \oplus l \\
]|
\end{array}
$$

Parity of $x[0,N]$ revealed — but nothing else.

Closing the circle



$$
\begin{array}{l}
|[\ \textbf{hid}\ l, r \cdot \\
\quad l :\in \{0, 1\}; \\
\quad r :\in \{0, 1\}; \\
\quad \textbf{reveal}\ l \oplus x[0, N) \oplus r; \\
\quad \textbf{reveal}\ r \oplus x[N] \oplus l \\
]|
\end{array}
$$

Parity of $x[0,N]$ revealed — but nothing else.

How do we treat loops?

First example

How do we treat loops?

~~First~~ example

...and only, in this talk

Iterated revelation

If

$$\textbf{if } h{>}0 \textbf{ then } h{:=}\ h{-}1 \textbf{ fi}$$

reveals whether h *was* non-zero initially,

Iterated revelation

If

$$\mathbf{if}\ h>0\ \mathbf{then}\ h := h-1\ \mathbf{fi}$$

reveals whether *h was* non-zero initially,

Iterated revelation

then does the loop

$$\textbf{while } h>1 \textbf{ do } h := h-2 \textbf{ od}$$

reveal the initial value of $h \div 2$?
Does it reveal more than that?

Iterated revelation

then does the loop

$$\textbf{while } h>1 \textbf{ do } h:=h-2 \textbf{ od}$$

reveal the initial value of $h \div 2$?
Does it reveal more than that?

Iterated revelation

then does the loop

$$\textbf{while } h>1 \textbf{ do } h{:=}\, h-2 \textbf{ od}$$

reveal the initial value of $h \div 2$?
Does it reveal more than that?

Loops are fixed-points, and unique if terminating

$$\begin{array}{lcl} \textbf{while } h{>}1 \textbf{ do} & = & \textbf{reveal } h{\div}2; \\ \quad h{:=}\, h{-}2 & & h{:=}\, h \,\textbf{mod}\, 2 \\ \textbf{od} & & \end{array}$$

Loops are fixed-points, and unique if terminating

$$\begin{array}{lcl} \textbf{while } h{>}1 \textbf{ do} & \stackrel{?}{=} & \textbf{reveal } h{\div}2; \\ \quad h{:=}\, h{-}2 & & h{:=}\, h \,\textbf{mod}\, 2 \\ \textbf{od} & & \end{array}$$

Loops are fixed-points, and unique if terminating

$$\begin{array}{lcl} \textbf{while } h{>}1 \textbf{ do} & = & \textbf{reveal } h{\div}2; \\ \quad h{:=}\, h{-}2 & & h{:=}\, h \,\textbf{mod}\, 2 \\ \textbf{od} & & \end{array}$$

Loops are fixed-points, and unique if terminating

$$\begin{array}{lcl} \textbf{while}\ h{>}1\ \textbf{do} & = & \textbf{reveal}\ h{\div}2; \\ \quad h{:=}\ h{-}2 & & h{:=}\ h \,\mathbf{mod}\, 2 \\ \textbf{od} & & \end{array}$$

just when

$$\begin{array}{lcl} \textbf{if}\ h{>}1\ \textbf{then} & = & \textbf{reveal}\ h{\div}2; \\ \quad h{:=}\ h{-}2; & & h{:=}\ h \,\mathbf{mod}\, 2 \\ \quad \textbf{while}\ h{>}1\ \textbf{do} & & \\ \qquad h{:=}\ h{-}2; & & \\ \quad \textbf{od} & & \\ \textbf{fi} & & \end{array}$$

Loops are fixed-points, and unique if terminating

$$\begin{array}{lcl} \textbf{while } h{>}1 \textbf{ do} & = & \textbf{reveal } h{\div}2; \\ \quad h{:=}\ h{-}2 & & h{:=}\ h \mathbin{\textbf{mod}} 2 \\ \textbf{od} & & \end{array}$$

just when

$$\begin{array}{lcl} \textbf{if } h{>}1 \textbf{ then} & = & \textbf{reveal } h{\div}2; \\ \quad h{:=}\ h{-}2; & & h{:=}\ h \mathbin{\textbf{mod}} 2 \\ \quad \textbf{reveal } h{\div}2; & & \\ \quad h{:=}\ h \mathbin{\textbf{mod}} 2 & & \\ \textbf{fi} & & \end{array}$$

Calculate for equality

$\textbf{reveal}\ h{\div}2;$
$h{:=}\ h\,\textbf{mod}\,2$

$\textbf{if}\ h{>}1\ \textbf{then}$
$\quad h{:=}\ h{-}2;$
$\quad \textbf{reveal}\ h{\div}2;$
$\quad h{:=}\ h\,\textbf{mod}\,2$
$\textbf{fi}$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h \mathbf{\,mod\,} 2$

$\mathbf{if}\ h{>}1\ \mathbf{then}$
$\quad h{:=}\ h{-}2;$
$\quad \mathbf{reveal}\ h{\div}2;$
$\quad h{:=}\ h \mathbf{\,mod\,} 2$
$\mathbf{fi}$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h\ \mathbf{mod}\ 2$

$\mathbf{if}\ h{>}1\ \mathbf{then}$
$\quad h{:=}\ h{-}2;$
$\quad \mathbf{reveal}\ h{\div}2;$
$\quad h{:=}\ h\ \mathbf{mod}\ 2$
$\mathbf{fi}$

Calculate for equality

$$\textbf{reveal } h{\div}2;\\ h{:=}\ h \,\textbf{mod}\, 2$$

$$\begin{array}{l}\textbf{if } h{>}1 \textbf{ then}\\ \quad \textbf{reveal } h{\div}2;\\ \quad h{:=}\ h{-}2;\\ \quad h{:=}\ h \,\textbf{mod}\, 2\\ \textbf{fi}\end{array}$$

Calculate for equality

$\textbf{reveal}\ h{\div}2;$
$h{:=}\ h \bmod 2$

$\textbf{if}\ h{>}1\ \textbf{then}$
$\quad\textbf{reveal}\ (h{-}2){\div}2;$
$\quad h{:=}\ h{-}2;$
$\quad h{:=}\ h \bmod 2$
$\textbf{fi}$

Calculate for equality

$\textbf{reveal}\ h \div 2;$
$h := h \ \textbf{mod}\ 2$

$\textbf{if}\ h > 1\ \textbf{then}$
$\quad \textbf{reveal}\ (h-2) \div 2;$
$\textbf{fi}$
$h := h \ \textbf{mod}\ 2$

Calculate for equality

$\mathbf{reveal}\ h \div 2;$
$h{:=}\ h \ \mathbf{mod}\ 2$

$$
\begin{array}{l}
\mathbf{if}\ h{>}1\ \mathbf{then} \\
\quad \mathbf{reveal}\ (h{-}2){\div}2 \\
\mathbf{fi}; \\
h{:=}\ h \ \mathbf{mod}\ 2
\end{array}
$$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h \ \mathbf{mod}\ 2$

$\mathbf{if}\ h{>}1\ \mathbf{then}$

$\quad\mathbf{reveal}\ (h{-}2){\div}2$

$\mathbf{fi};$
$h{:=}\ h \ \mathbf{mod}\ 2$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h \,\mathbf{mod}\ 2$

$$\begin{array}{l}
\mathbf{if}\ h{>}1\ \mathbf{then} \\
\quad \{h{>}1\} \\
\quad \mathbf{reveal}\ (h{-}2){\div}2 \\
\mathbf{else} \\
\quad \{h{\leq}1\} \\
\\
\mathbf{fi}; \\
h{:=}\ h \,\mathbf{mod}\, 2
\end{array}$$

Calculate for equality

$\textbf{reveal}\ h{\div}2;$
$h{:=}\ h \,\textbf{mod}\, 2$

$$
\begin{array}{l}
\textbf{if}\ h{>}1\ \textbf{then} \\
\quad \{h{>}1\} \\
\quad \textbf{reveal}\ (h{-}2){\div}2 \\
\textbf{else} \\
\quad \{h{\leq}1\} \\
\quad \textbf{reveal}\ h{\div}2 \\
\textbf{fi}; \\
h{:=}\ h \,\textbf{mod}\, 2
\end{array}
$$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h \mathbin{\mathbf{mod}} 2$

$\mathbf{if}\ h{>}1\ \mathbf{then}$
$\quad \{h{>}1\}$
$\quad \mathbf{reveal}\ h{\div}2$
$\mathbf{else}$
$\quad \{h{\leq}1\}$
$\quad \mathbf{reveal}\ h{\div}2$
$\mathbf{fi};$
$h{:=}\ h \mathbin{\mathbf{mod}} 2$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h\,\mathbf{mod}\ 2$

$\mathbf{if}\ h{>}1\ \mathbf{then}$
$\quad\mathbf{reveal}\ h{\div}2$
$\mathbf{else}$
$\quad\mathbf{reveal}\ h{\div}2$
$\mathbf{fi};$
$h{:=}\ h\,\mathbf{mod}\,2$

Calculate for equality

$\textbf{reveal } h{\div}2;$
$h{:=}\ h \textbf{ mod } 2$

$\textbf{if } h{>}1 \textbf{ then}$
$\quad \textbf{reveal } h{\div}2$
$\textbf{else}$
$\quad \textbf{reveal } h{\div}2$
$\textbf{fi};$
$h{:=}\ h \textbf{ mod } 2$

Calculate for equality

$\textbf{reveal}\ h \div 2;$
$h := h \ \textbf{mod}\ 2$

$$\textbf{reveal}\ h \div 2$$
$$h := h \ \textbf{mod}\ 2$$

Calculate for equality

$\mathbf{reveal}\ h \div 2;$
$h := h \mathbin{\mathbf{mod}} 2$

$\mathbf{reveal}\ h > 1;$
$\mathbf{reveal}\ h \div 2;$
$h := h \mathbin{\mathbf{mod}} 2$

Calculate for equality

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h \mathbin{\mathbf{mod}} 2$

$\mathbf{reveal}\ h{\div}2;$
$h{:=}\ h \mathbin{\mathbf{mod}} 2$

We *have* the equality

$\textbf{reveal}\ h{\div}2;$
$h{:=}\ h \,\textbf{mod}\, 2$

$$\begin{array}{lcl} \textbf{while}\ h{>}1\ \textbf{do} & = & \textbf{reveal}\ h{\div}2; \\ \quad h{:=}\ h{-}2 & & h{:=}\ h \,\textbf{mod}\, 2 \\ \textbf{od} & & \end{array}$$

We *have* the equality

then does the loop

$$\begin{array}{lcl} \textbf{while}\ h{>}1\ \textbf{do} & = & \textbf{reveal}\ h{\div}2; \\ \quad h{:=}\ h{-}2 & & h{:=}\ h \bmod 2 \\ \textbf{od} & & \end{array}$$

reveal the initial value of $h{\div}2$?
Does it reveal more than that?

We *have* the equality

then does the loop

$$\textbf{while } h{>}1 \textbf{ do } h{:=}\, h{-}2 \textbf{ od} \quad = \quad \textbf{reveal } h{\div}2;\ h{:=}\, h \bmod 2$$

reveal the initial value of $h{\div}2$? Yes.
Does it reveal more than that?

We *have* the equality

then does the loop

$$\textbf{while } h{>}1 \textbf{ do} \quad h{:=}\, h{-}2 \quad \textbf{od} \qquad = \qquad \textbf{reveal } h{\div}2;\ h{:=}\, h \,\textbf{mod}\, 2$$

reveal the initial value of *h*÷2? Yes.

Does it reveal more than that? No.

We *have* the equality

then does the loop

$$\textbf{while } h>1 \textbf{ do} \quad h := h-2 \quad \textbf{od} \quad = \quad \textbf{reveal } h\div 2; \; h := h \textbf{ mod } 2$$

reveal the initial value of $h\div 2$? Yes.

Does it reveal more than that? No.

Actually this is quite a delicate question: what's "more"?

Quite a delicate question...

All those delicate questions are answered for the loop, automatically, if they are answerable for its straight-line simplification...

...and in *any* context.

This is why the refinement-based approach has been so popular elsewhere, for >35 years.

Quite a delicate question...

> *All* those delicate questions are answered for the loop, automatically, if they are answerable for its straight-line simplification...
>
> ...and in *any* context.
>
> This is why the refinement-based approach has been so popular elsewhere, for >35 years.

Program development by stepwise refinement. Wirth, CACM 1971.

Quite a delicate question...

> *All* those delicat… …nswered for the loop, automati… …verable for its straight-line s…
>
> ...and in *any*…
>
> This is why the … …pproach has been so popular el… … >35 years.

This is why we need it here too.

Program development by stepwise refinement. Wirth, CACM 1971.

How do we treat loops?

Second example:

unboundedly many cryptographers

How do we treat loops?

Second example:

unboundedly many cryptographers

Conclusions

Conclusions

How do we treat loops?

Second example:

unboundedly many
cryptographers

Conclusions

We have

... a firm semantics

... a supple algebra

... easy automation possible in a simple-minded style

... integration with traditional reasoning

... examples (almost) never done before

Conclusions

... a firm semantics

(not given in the talk)

... a supple algebra

... easy automation possible in a simple-minded style

... integration with traditional reasoning

... examples (almost) never done before

Conclusions

... a firm semantics

(not given in the talk)

... a supple algebra

(illustrated above)

... easy automation possible in a simple-minded style

... integration with traditional reasoning

... examples (almost) never done before

Conclusions

... a firm semantics

(not given in the talk)

... a supple algebra

(illustrated above)

... easy automation possible in a simple-minded style

(suits the hacker mentality)

... integration with traditional reasoning

... examples (almost) never done before

Conclusions

... a firm semantics

(not given in the talk)

... a supple algebra

(illustrated above)

... easy automation possible in a simple-minded style

(suits the hacker mentality)

... integration with traditional reasoning

(scale-up principles)

... examples (almost) never done before

Conclusions

... a firm semantics

(not given in the talk)

... a supple algebra

(illustrated above)

... easy automation possible in a simple-minded style

(suits the hacker mentality)

... integration with traditional reasoning

(scale-up principles)

... examples (almost) never done before

(unbounded Dining Cryptographers)

The Dining Cryptographers, for arbitrary *N*

$$\textbf{reveal}\ x[0,N]$$

$$
\begin{array}{l}
|[\ \textbf{hid}\ l,m,r:\{0,1\}\cdot \\
\quad l{:\in}\ \{0,1\};\ \ m{:=}\ l; \\
\quad n{:=}\ N; \\
\quad \textbf{repeat} \\
\quad\quad n{:=}\ n{-}1; \\
\quad\quad r{:\in}\ \{0,1\}; \\
\quad\quad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\quad\quad m{:=}\ r \\
\quad \textbf{until}\ n{=}0; \\
\quad \textbf{reveal}\ r \oplus x[N] \oplus l \\
]|
\end{array}
$$

The Dining Cryptographers,
for arbitrary *N*

$$\textbf{reveal } x[0, N]$$

$$
\begin{array}{l}
|[\ \textbf{hid}\ l, m, r : \{0,1\} \cdot \\
\quad l :\in \{0,1\};\ m := l; \\
\quad n := N; \\
\quad \textbf{repeat} \\
\qquad n := n - 1; \\
\qquad r :\in \{0,1\}; \\
\qquad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\qquad m := r \\
\quad \textbf{until}\ n = 0; \\
\quad \textbf{reveal}\ r \oplus x[N] \oplus l \\
]|
\end{array}
$$

By using model checking techniques one can verify DC up to 8 and more cryptographers with resulting state spaces for the model of about 10^{36} states, and considerably more cryptographers if the representation of the model is optimised.

Pucella, SIGACT News, 2007

The Dining Cryptographers,
for arbitrary *N*

$\mathbf{reveal}\ x[0, N]$

$$|[\ \mathbf{hid}\ l, m, r : \{0,1\} \cdot$$

$$l :\in \{0,1\};\ m := l;$$

N

To the best of the author's knowledge, this is the first machine-verified proof of privacy of the Dining Cryptographers protocol for an unbounded number of participants and a quantitative metric for privacy.

Coble, 2008

$$\mathbf{until}\ n = 0;$$

$$\mathbf{reveal}\ r \oplus x[N] \oplus l$$

$$]|$$

By using model checking
...erify DC
...and more
...resulting
...model of
...ates, and
...bly more
...ers if the
model is
optimised.

Pucella, SIGACT News, 2007

Making a **repeat** loop via a fixed-point argument

— Assume l is already set.

$r :\in \{0, 1\};$

$\mathbf{reveal}\ l \oplus x[0, N) \oplus r$

“Reasonable” guess
for a suitable loop

$$
\begin{array}{l}
l{:\in}\ \{0,1\};\\
r{:\in}\ \{0,1\};\\
\textbf{reveal}\ l \oplus x[0,N) \oplus r
\end{array}
$$

$$
\begin{array}{l}
n{:=}\ N;\\
l{:\in}\ \{0,1\};\\
\textbf{repeat}\\
\quad n{:=}\ n{-}1;\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal}\ l \oplus x[n] \oplus r;\\
\quad l{:=}\ r\\
\textbf{until}\ n{=}0
\end{array}
$$

But not *quite* right,
because it overwrites *l*

$$l{:\in}\ \{0,1\};$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ l \oplus x[0,N) \oplus r$$

$$n{:=}\ N;$$
$$l{:\in}\ \{0,1\};$$
$$\textbf{repeat}$$
$$\quad n{:=}\ n{-}1;$$
$$\quad r{:\in}\ \{0,1\};$$
$$\quad \textbf{reveal}\ l \oplus x[n] \oplus r;$$
$$\quad l{:=}\ r$$
$$\textbf{until}\ n{=}0$$

So introduce a temporary variable *m*

$$l{:\in}\ \{0,1\};$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ l \oplus x[0,N) \oplus r$$

$$n{:=}\ N;$$
$$l{:\in}\ \{0,1\};$$
$$m{:=}\ l;$$
$$\textbf{repeat}$$
$$\quad n{:=}\ n{-}1;$$
$$\quad r{:\in}\ \{0,1\};$$
$$\quad \textbf{reveal}\ m \oplus x[n] \oplus r;$$
$$\quad m{:=}\ r$$
$$\textbf{until}\ n{=}0$$

So introduce a temporary variable *m*

$$l{:}\in \{0,1\};$$
$$r{:}\in \{0,1\};$$
$$\textbf{reveal}\ l \oplus x[0,N) \oplus r$$

$$n{:=}\ N;$$
$$l{:}\in \{0,1\};$$
$$m{:=}\ l;$$
$$\textbf{repeat}$$
$$\quad n{:=}\ n{-}1;$$
$$\quad r{:}\in \{0,1\};$$
$$\quad \textbf{reveal}\ m \oplus x[n] \oplus r;$$
$$\quad m{:=}\ r$$
$$\textbf{until}\ n{=}0$$

Concentrate on the
repeat on its own

$$
\begin{array}{l}
n{:=}\ N; \\
l{:\in}\ \{0,1\}; \\
m{:=}\ l; \\
\textbf{repeat} \\
\quad n{:=}\ n{-}1; \\
\quad r{:\in}\ \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\quad m{:=}\ r \\
\textbf{until}\ n{=}0
\end{array}
$$

Concentrate on the **repeat** on its own

— Assume $n>0$
$r{:\in}\ \{0,1\}$;
reveal $m \oplus x[0,n) \oplus r$;
$m{:=}\ r$;
$n{:=}\ 0$

repeat
$n{:=}\ n{-}1$;
$r{:\in}\ \{0,1\}$;
reveal $m \oplus x[n] \oplus r$;
$m{:=}\ r$
until $n{=}0$

Apply the fixed-point functional

$$
\begin{array}{l}
n{:=}\ n{-}1; \\
r{:\in}\ \{0,1\}; \\
\textbf{reveal}\ m \oplus x[n] \oplus r; \\
m{:=}\ r; \\
\textbf{if}\ n{>}0\ \textbf{then} \\
\quad r{:\in}\ \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r; \\
\quad m{:=}\ r; \\
\quad n{:=}\ 0 \\
\textbf{fi}
\end{array}
$$

Apply the fixed-point functional

$n{:=}\ n{-}1;$ loop body

$r{:\in}\ \{0,1\};$

$\textbf{reveal}\ m \oplus x[n] \oplus r;$

loop conditional

$\textbf{if}\ n{>}0\ \textbf{then}$

$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$ loop specification

$\textbf{fi}$

It must *be* a fixed-point

loop body

$n := n - 1;$
$r :\in \{0, 1\};$
$\textbf{reveal}\ m \oplus x[n] \oplus r;$

loop conditional

$\textbf{if}\ n > 0\ \textbf{then}$

$r :\in \{0, 1\};$
$\textbf{reveal}\ m \oplus x[0, n) \oplus r;$
$m := r;$
$n := 0$

loop specification

$\textbf{fi}$

It must *be* a fixed-point

$r{:\in}\ \{0,1\};$
reveal $m \oplus x[0,n) \oplus r;$
$m{:=}\ r;$
$n{:=}\ 0$ loop specification

$n{:=}\ n{-}1;$
$r{:\in}\ \{0,1\};$
reveal $m \oplus x[n] \oplus r;$
$m{:=}\ r;$
if $n{>}0$ **then**

$r{:\in}\ \{0,1\};$
reveal $m \oplus x[0,n) \oplus r;$
$m{:=}\ r;$
$n{:=}\ 0$ loop specification

fi

We calculate as follows...
Expand the **if**

$$
\begin{array}{l}
n{:=}\ n{-}1;\\
r{:\in}\ \{0,1\};\\
\textbf{reveal}\ m \oplus x[n] \oplus r;\\
m{:=}\ r;\\
\textbf{if}\ n{>}0\ \textbf{then}\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
\quad m{:=}\ r;\\
\quad n{:=}\ 0\\
\textbf{fi}
\end{array}
$$

We calculate as follows...
Expand the **if**

$$n{:=}\ n{-}1;$$
$$r{:\in}\ \{0,1\};$$
$$\mathbf{reveal}\ m \oplus x[n] \oplus r;$$
$$m{:=}\ r;$$

$$\mathbf{if}\ n{>}0\ \mathbf{then}$$
$$\quad n{:=}\ n{-}1;$$
$$\quad r{:\in}\ \{0,1\};$$
$$\quad \mathbf{reveal}\ m \oplus x[n] \oplus r;$$
$$\quad m{:=}\ r;$$
$$\quad r{:\in}\ \{0,1\};$$
$$\quad \mathbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$\quad m{:=}\ r;$$
$$\quad n{:=}\ 0$$
$$\mathbf{fi}$$

Use n=1 in the **else**

$\ldots$ **else**

$n := n-1;$
$r :\in \{0,1\};$
reveal $m \oplus x[n] \oplus r;$
$m := r$
fi

if $n{>}1$ **then**
$n := n-1;$
$r :\in \{0,1\};$
reveal $m \oplus x[n] \oplus r;$
$m := r;$
$r :\in \{0,1\};$
reveal $m \oplus x[0,n) \oplus r;$
$m := r;$
$n := 0$
else $\ldots$

Use n=1 in the **else**

$$
\begin{array}{l}
\textbf{if } n{>}1 \textbf{ then}\\
\quad n{:=}\ n{-}1;\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal } m \oplus x[n] \oplus r;\\
\quad m{:=}\ r;\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal } m \oplus x[0,n) \oplus r;\\
\quad m{:=}\ r;\\
\quad n{:=}\ 0\\
\textbf{else} \ldots
\end{array}
$$

$$
\begin{array}{l}
\ldots \textbf{else}\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal } m \oplus x[n] \oplus r;\\
\quad m{:=}\ r\\
\quad n{:=}\ n{-}1;\\
\textbf{fi}
\end{array}
$$

Introduce local hidden r'

$$\ldots\ \textbf{else}$$

$$r{:\in}\ \{0,1\};$$

$$\textbf{reveal}\ m \oplus x[0] \oplus r;$$

$$m{:=}\ r;$$

$$n{:=}\ 0$$

$$\textbf{fi}$$

$$\textbf{if}\ n{>}1\ \textbf{then}$$

$$n{:=}\ n{-}1;$$

$$r{:\in}\ \{0,1\};$$

$$\textbf{reveal}\ m \oplus x[n] \oplus r;$$

$$m{:=}\ r;$$

$$r{:\in}\ \{0,1\};$$

$$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$$

$$m{:=}\ r;$$

$$n{:=}\ 0$$

$$\textbf{else}\ \ldots$$

Introduce local hidden r'

$\ldots$ **else**

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[0] \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$

fi

if $n{>}1$ **then**

$n{:=}\ n{-}1;$

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[0,n) \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$

else $\ldots$

Reorder; eliminate superfluous assignment

$$\ldots \textbf{else}$$

$$
\begin{aligned}
& r{:}\in \{0,1\};\\
& \textbf{reveal}\ m \oplus x[0] \oplus r;\\
& m{:}=r;\\
& n{:}=0\\
& \textbf{fi}
\end{aligned}
$$

$$
\begin{aligned}
& \textbf{if}\ n{>}1\ \textbf{then}\\
& \quad n{:}=n{-}1;\\
& \quad |[\ \textbf{hid}\ r' \cdot\\
& \qquad r'{:}\in \{0,1\};\ \textbf{reveal}\ m \oplus x[n] \oplus r';\\
& \qquad m{:}=r'\ \ ]|;\\
& \quad r{:}\in \{0,1\};\\
& \quad \textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
& \quad m{:}=r;\\
& \quad n{:}=0\\
& \textbf{else} \ldots
\end{aligned}
$$

Reorder; eliminate superfluous assignment

$r :\in \{0,1\};$
if $n>1$ **then**
  $n := n-1;$
  $|[$ **hid** $r' \cdot$
    $r' :\in \{0,1\};$
    reveal $m \oplus x[n] \oplus r';$
  reveal $m \oplus x[0,n) \oplus r;$
  $]|;$
else . . .

. . . **else**
  reveal $m \oplus x[0] \oplus r;$
fi
$m := r;$
$n := 0$

Reveal calculus;
use n=1 in **else**

$$
\begin{array}{l}
\ldots\,\textbf{else} \\
\quad \textbf{reveal}\ m \oplus x[0] \oplus r \\
\textbf{fi} \\
m{:=}\ r; \\
n{:=}\ 0
\end{array}
$$

$$
\begin{array}{l}
r{:\in}\ \{0,1\}; \\
\textbf{if}\ n{>}1\ \textbf{then} \\
\quad n{:=}\ n{-}1; \\
\quad |[\ \textbf{hid}\ r' \cdot \\
\qquad r'{:\in}\ \{0,1\}; \\
\qquad \textbf{reveal}\ m \oplus x[n] \oplus r'; \\
\qquad \textbf{reveal}\ r' \oplus x[0,n) \oplus r \\
\quad]| \\
\textbf{else}\ldots
\end{array}
$$

Reveal calculus;
use n=1 in **else**

$r{:}\in \{0,1\};$
$\textbf{if}\ n{>}1\ \textbf{then}$
$\quad n{:=}\ n{-}1;$
$\quad [\![\ \textbf{hid}\ r' \cdot$
$\qquad r'{:}\in \{0,1\};$
$\qquad \textbf{reveal}\ m \oplus x[n] \oplus r';$
$\qquad \textbf{reveal}\ r' \oplus x[0,n) \oplus r$
$\quad]\!]$
$\textbf{else} \ldots$

$\ldots \textbf{else}$
$\quad \textbf{reveal}\ m \oplus x[0] \oplus r$
$\textbf{fi}$
$m{:=}\ r;$
$n{:=}\ 0$

Reorder; remove assignment to *n*

$$
\begin{array}{l}
r\!:\in \{0,1\};\\
\textbf{if } n{>}1 \textbf{ then}\\
\quad n\!:= n{-}1;\\
\quad [\![\ \textbf{hid } r' \cdot\\
\quad\quad r'\!:\in \{0,1\};\\
\quad\quad \textbf{reveal } m \oplus x[n] \oplus r';\\
\quad\quad \textbf{reveal } m \oplus x[0,n] \oplus r\\
\quad]\!]\\
\textbf{else} \ldots
\end{array}
\qquad
\begin{array}{l}
\ldots \textbf{else}\\
\quad \textbf{reveal } m \oplus x[0,n) \oplus r\\
\textbf{fi}\\
m\!:= r;\\
n\!:= 0
\end{array}
$$

Reorder; remove assignment to *n*

$$
\begin{array}{l}
\ldots\,\textbf{else} \\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r \\
\textbf{fi} \\
m{:=}\ r; \\
n{:=}\ 0
\end{array}
$$

$$
\begin{array}{l}
r{:\in}\ \{0,1\}; \\
\textbf{if}\ n{>}1\ \textbf{then} \\
\quad |[\ \textbf{hid}\ r' \cdot \\
\qquad r'{:\in}\ \{0,1\}; \\
\qquad \textbf{reveal}\ m \oplus x[n] \oplus r'; \\
\qquad \textbf{reveal}\ m \oplus x[0,n] \oplus r \\
\quad]| \\
\textbf{else}\ldots
\end{array}
$$

Reorder

$$\ldots \textbf{else}$$
$$\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r$$
$$\textbf{fi}$$
$$m{:=}\ r;$$
$$n{:=}\ 0$$

$$r{:\in}\ \{0,1\};$$
$$\textbf{if}\ n{>}1\ \textbf{then}$$
$$\quad |[\ \textbf{hid}\ r'\ \cdot$$
$$\qquad r'{:\in}\ \{0,1\};$$
$$\qquad \textbf{reveal}\ m \oplus x[n{-}1] \oplus r';$$
$$\qquad \textbf{reveal}\ m \oplus x[0,n) \oplus r$$
$$\quad]|$$
$$\textbf{else} \ldots$$

Reorder

$$
\begin{array}{l}
r :\in \{0,1\};\\
\textbf{if } n>1 \textbf{ then}\\
\quad [\![\textbf{ hid } r' \cdot\\
\qquad r' :\in \{0,1\};\\
\qquad \textbf{reveal } m \oplus x[n-1] \oplus r';\\
\quad]\!]\\
\\
\textbf{reveal } m \oplus x[0,n) \oplus r\\
m := r;\\
n := 0
\end{array}
$$

Encryption lemma

$$
\begin{array}{l}
r{:\in}\ \{0,1\};\\
\textbf{if}\ n{>}1\ \textbf{then}\\
\quad [\![\ \textbf{hid}\ r' \cdot\\
\qquad r'{:\in}\ \{0,1\};\\
\qquad \textbf{reveal}\ m \oplus x[n{-}1] \oplus r'\\
\quad]\!]\\
\textbf{fi};\\
\textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
m{:=}\ r;\\
n{:=}\ 0
\end{array}
$$

Encryption lemma

$$r :\in \{0, 1\};$$

$$\textbf{reveal}\ m \oplus x[0, n) \oplus r;$$
$$m := r;$$
$$n := 0$$

Variable *n* is visible

$$r{:\in}\ \{0,1\};$$
$$\textbf{if}\ n{>}1\ \textbf{then skip fi};$$
$$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$m{:=}\ r;$$
$$n{:=}\ 0$$

Variable *n* is visible

$$r{:}\in \{0,1\};$$
$$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$m{:}= r;$$
$$n{:}= 0$$

And we started with...

$$r{:}\in \{0,1\};$$
$$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$m{:}=r;$$
$$n{:}=0$$

— Assume $n>0$

$r{:\in}\ \{0,1\};$
$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$
$m{:=}\ r;$
$n{:=}\ 0$

And we started with...

$$
\begin{array}{l}
r{:\in}\ \{0,1\}; \\
\textbf{reveal}\ m \oplus x[0,n) \oplus r; \\
m{:=}\ r; \\
n{:=}\ 0
\end{array}
$$

The complete program,
with the circle closed

reveal $x[0, N]$

$$
\begin{array}{l}
[\![\ \textbf{hid}\ l, m, r : \{0,1\}\cdot \\
\quad l{:\in}\ \{0,1\};\ \ m{:=}\ l; \\
\quad n{:=}\ N; \\
\quad \textbf{repeat} \\
\qquad n{:=}\ n{-}1; \\
\qquad r{:\in}\ \{0,1\}; \\
\qquad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\qquad m{:=}\ r \\
\quad \textbf{until}\ n{=}0; \\
\quad \textbf{reveal}\ r \oplus x[N] \oplus l \\
\ ]\!]
\end{array}
$$

Conclusions

We have

... a firm semantics
... a supple algebra
... easy automation in a simple-minded style
... integration with traditional reasoning
... examples (almost) never done before

Cryptographers' derivation: reprise

Making a **repeat** loop via a fixed-point argument

— Assume l is already set.
$r :\in \{0, 1\};$
$\textbf{reveal}\ l \oplus x[0, N) \oplus r$

“Reasonable” guess
for a suitable loop

$l{:\in}\ \{0,1\};$
$r{:\in}\ \{0,1\};$
reveal $l \oplus x[0,N) \oplus r$

$n{:=}\ N;$
$l{:\in}\ \{0,1\};$
repeat
  $n{:=}\ n{-}1;$
  $r{:\in}\ \{0,1\};$
  reveal $l \oplus x[n] \oplus r;$
  $l{:=}\ r$
until $n{=}0$

But not *quite* right,
because it overwrites *l*

$$l{:\in}\ \{0,1\};$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ l \oplus x[0,N) \oplus r$$

$$n{:=}\ N;$$
$$l{:\in}\ \{0,1\};$$
$$\textbf{repeat}$$
$$\quad n{:=}\ n{-}1;$$
$$\quad r{:\in}\ \{0,1\};$$
$$\quad \textbf{reveal}\ l \oplus x[n] \oplus r;$$
$$\quad l{:=}\ r$$
$$\textbf{until}\ n{=}0$$

So introduce a temporary variable *m*

$$l{:\in}\ \{0,1\};$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ l \oplus x[0,N) \oplus r$$

$$n{:=}\ N;$$
$$l{:\in}\ \{0,1\};$$
$$m{:=}\ l;$$
$$\textbf{repeat}$$
$$\quad n{:=}\ n{-}1;$$
$$\quad r{:\in}\ \{0,1\};$$
$$\quad \textbf{reveal}\ m \oplus x[n] \oplus r;$$
$$\quad m{:=}\ r$$
$$\textbf{until}\ n{=}0$$

So introduce a temporary variable *m*

$l{:}\in \{0,1\};$
$r{:}\in \{0,1\};$
$\textbf{reveal}\ l \oplus x[0,N) \oplus r$

$$
\begin{aligned}
&n{:}=N;\\
&l{:}\in \{0,1\};\\
&m{:}=l;\\
&\textbf{repeat}\\
&\quad n{:}=n-1;\\
&\quad r{:}\in \{0,1\};\\
&\quad \textbf{reveal}\ m \oplus x[n] \oplus r;\\
&\quad m{:}=r\\
&\textbf{until}\ n=0
\end{aligned}
$$

Concentrate on the **repeat** on its own

$$
\begin{aligned}
&n{:=}\ N;\\
&l{:\in}\ \{0,1\};\\
&m{:=}\ l;\\
&\textbf{repeat}\\
&\quad n{:=}\ n{-}1;\\
&\quad r{:\in}\ \{0,1\};\\
&\quad \textbf{reveal}\ m \oplus x[n] \oplus r;\\
&\quad m{:=}\ r\\
&\textbf{until}\ n{=}0
\end{aligned}
$$

Concentrate on the
repeat on its own

$$
\begin{array}{l}
\textbf{repeat} \\
\quad n{:=}\ n{-}1; \\
\quad r{:\in}\ \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\quad m{:=}\ r \\
\textbf{until}\ n{=}0
\end{array}
$$

Concentrate on the **repeat** on its own

$$
\begin{array}{l}
\text{— Assume } n{>}0 \\
r{:\in}\ \{0,1\}; \\
\textbf{reveal}\ m \oplus x[0,n) \oplus r; \\
m{:=}\ r; \\
n{:=}\ 0
\end{array}
$$

$$
\begin{array}{l}
\textbf{repeat} \\
\quad n{:=}\ n{-}1; \\
\quad r{:\in}\ \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\quad m{:=}\ r \\
\textbf{until}\ n{=}0
\end{array}
$$

Apply the fixed-point functional

$$
\begin{array}{l}
n{:=}\ n{-}1;\\
r{:\in}\ \{0,1\};\\
\textbf{reveal}\ m \oplus x[n] \oplus r;\\
m{:=}\ r;\\
\textbf{if}\ n{>}0\ \textbf{then}\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
\quad m{:=}\ r;\\
\quad n{:=}\ 0\\
\textbf{fi}
\end{array}
$$

Apply the fixed-point functional

$n := n - 1;$ loop body
$r :\in \{0, 1\};$
$\mathbf{reveal}\ m \oplus x[n] \oplus r;$
$m := r;$

$\mathbf{if}\ n > 0\ \mathbf{then}$
$\quad r :\in \{0, 1\};$
$\quad \mathbf{reveal}\ m \oplus x[0, n) \oplus r;$
$\quad m := r;$
$\quad n := 0$
$\mathbf{fi}$

Apply the fixed-point functional

$n{:=}\ n{-}1;$ loop body

$r{:\in}\ \{0,1\};$

$\textbf{reveal}\ m \oplus x[n] \oplus r;$

$m{:=}\ r;$

$\textbf{if}\ n{>}0\ \textbf{then}$

$r{:\in}\ \{0,1\};$

$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$ loop specification

$\textbf{fi}$

Apply the fixed-point functional

$n{:=}\ n{-}1;$

$r{:\in}\ \{0,1\};$

$\textbf{reveal}\ m \oplus x[n] \oplus r;$

loop body

loop conditional

$\textbf{if}\ n{>}0\ \textbf{then}$

$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$

loop specification

fi

It must *be* a fixed-point

loop body

$n{:=}\ n{-}1;$
$r{:\in}\ \{0,1\};$
$\textbf{reveal}\ m \oplus x[n] \oplus r;$

loop conditional

$\textbf{if}\ n{>}0\ \textbf{then}$

$r{:\in}\ \{0,1\};$
$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$
$m{:=}\ r;$
$n{:=}\ 0$

loop specification

$\textbf{fi}$

It must *be* a fixed-point

$r{:}\in \{0,1\};$
reveal $m \oplus x[0,n) \oplus r;$
$m{:}= r;$
$n{:}= 0$ loop specification

$n{:}= n{-}1;$
$r{:}\in \{0,1\};$
reveal $m \oplus x[n] \oplus r;$
$m{:}= r;$
if $n{>}0$ **then**

$r{:}\in \{0,1\};$
reveal $m \oplus x[0,n) \oplus r;$
$m{:}= r;$
$n{:}= 0$ loop specification

fi

We calculate as follows...
Expand the **if**

$$
\begin{array}{l}
n{:=}\ n{-}1;\\
r{:\in}\ \{0,1\};\\
\textbf{reveal}\ m \oplus x[n] \oplus r;\\
m{:=}\ r;\\
\textbf{if}\ n{>}0\ \textbf{then}\\
\quad r{:\in}\ \{0,1\};\\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
\quad m{:=}\ r;\\
\quad n{:=}\ 0\\
\textbf{fi}
\end{array}
$$

We calculate as follows...
Expand the **if**

$$\begin{array}{l} \textbf{if}\ n{>}0\ \textbf{then} \\ \quad n{:=}\ n{-}1; \\ \quad r{:}{\in}\ \{0,1\}; \\ \quad \textbf{reveal}\ m \oplus x[n] \oplus r; \\ \quad m{:=}\ r; \\ \quad r{:}{\in}\ \{0,1\}; \\ \quad \textbf{reveal}\ m \oplus x[0,n) \oplus r; \\ \quad m{:=}\ r; \\ \quad n{:=}\ 0 \\ \textbf{fi} \end{array}$$

$$\begin{array}{l} n{:=}\ n{-}1; \\ r{:}{\in}\ \{0,1\}; \\ \textbf{reveal}\ m \oplus x[n] \oplus r; \\ m{:=}\ r; \end{array}$$

We calculate as follows...
Expand the **if**

$$
\begin{array}{l}
n{:=}\ n{-}1; \\
r{:\in}\ \{0,1\}; \\
\textbf{reveal}\ m \oplus x[n] \oplus r; \\
m{:=}\ r; \\
\textbf{if}\ n{>}0\ \textbf{then} \\
\quad r{:\in}\ \{0,1\}; \\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r; \\
\quad m{:=}\ r; \\
\quad n{:=}\ 0 \\
\textbf{fi}
\end{array}
$$

We calculate as follows...
Expand the **if**

$$\begin{array}{l} \textbf{if } n{>}0 \textbf{ then} \\ \quad n{:=}\ n{-}1; \\ \quad r{:\in}\ \{0,1\}; \\ \quad \textbf{reveal } m \oplus x[n] \oplus r; \\ \quad m{:=}\ r; \\ \quad r{:\in}\ \{0,1\}; \\ \quad \textbf{reveal } m \oplus x[0,n) \oplus r; \\ \quad m{:=}\ r; \\ \quad n{:=}\ 0 \\ \textbf{fi} \end{array}$$

$$\begin{array}{l} n{:=}\ n{-}1; \\ r{:\in}\ \{0,1\}; \\ \textbf{reveal } m \oplus x[n] \oplus r; \\ m{:=}\ r; \end{array}$$

Use n=1 in the **else**

$$\ldots\ \textbf{else}$$

$$n{:=}\ n{-}1;$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ m \oplus x[n] \oplus r;$$
$$m{:=}\ r$$
$$\textbf{fi}$$

$$\textbf{if}\ n{>}1\ \textbf{then}$$
$$n{:=}\ n{-}1;$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ m \oplus x[n] \oplus r;$$
$$m{:=}\ r;$$
$$r{:\in}\ \{0,1\};$$
$$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$m{:=}\ r;$$
$$n{:=}\ 0$$
$$\textbf{else}\ \ldots$$

Use n=1 in the **else**

$\ldots$ **else**

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[n] \oplus r;$

$m{:=}\ r$

$n{:=}\ n{-}1;$

fi

if $n{>}1$ **then**

$n{:=}\ n{-}1;$

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[n] \oplus r;$

$m{:=}\ r;$

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[0,n) \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$

else $\ldots$

Introduce local hidden r'

$\textbf{if}\ n{>}1\ \textbf{then}$
$\quad n{:=}\ n{-}1;$
$\quad r{:\in}\ \{0,1\};$
$\quad \textbf{reveal}\ m \oplus x[n] \oplus r;$
$\quad m{:=}\ r;$
$\quad r{:\in}\ \{0,1\};$
$\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r;$
$\quad m{:=}\ r;$
$\quad n{:=}\ 0$
$\textbf{else} \ldots$

$\ldots\textbf{else}$
$\quad r{:\in}\ \{0,1\};$
$\quad \textbf{reveal}\ m \oplus x[0] \oplus r;$
$\quad m{:=}\ r;$
$\quad n{:=}\ 0$
$\textbf{fi}$

$\ldots$ **else**

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[0] \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$

fi

Introduce local hidden r'

if $n{>}1$ **then**

$n{:=}\ n{-}1;$

$r{:\in}\ \{0,1\};$

reveal $m \oplus x[0,n) \oplus r;$

$m{:=}\ r;$

$n{:=}\ 0$

else $\ldots$

Reorder; eliminate superfluous assignment

$\ldots$ **else**

$r{:}\in \{0,1\}$;

reveal $m \oplus x[0] \oplus r$;

$m{:}= r$;

$n{:}= 0$

fi

if $n{>}1$ **then**

$n{:}= n{-}1$;

$|[$ **hid** $r' \cdot$

$r'{:}\in \{0,1\}$; **reveal** $m \oplus x[n] \oplus r'$;

$m{:}= r'$ $]|$;

$r{:}\in \{0,1\}$;

reveal $m \oplus x[0,n) \oplus r$;

$m{:}= r$;

$n{:}= 0$

else $\ldots$

Reorder; eliminate superfluous assignment

$$
\begin{array}{l}
r{:\in}\ \{0,1\};\\
\textbf{if}\ n{>}1\ \textbf{then}\\
\quad n{:=}\ n{-}1;\\
\quad [\![\ \textbf{hid}\ r'\ \cdot\\
\quad\quad r'{:\in}\ \{0,1\};\\
\quad\quad \textbf{reveal}\ m \oplus x[n] \oplus r';\\
\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
\quad]\!];\\
\textbf{else}\ \ldots
\end{array}
$$

$$
\begin{array}{l}
\ldots\ \textbf{else}\\
\quad\quad \textbf{reveal}\ m \oplus x[0] \oplus r;\\
\quad \textbf{fi}\\
\quad m{:=}\ r;\\
\quad n{:=}\ 0
\end{array}
$$

Reveal calculus;
use n=1 in **else**

$$
\begin{array}{l}
r:\in \{0,1\};\\
\textbf{if } n>1 \textbf{ then}\\
\quad n:= n-1;\\
\quad |[\ \textbf{hid } r' \cdot\\
\quad\quad r':\in \{0,1\};\\
\quad\quad \textbf{reveal } m \oplus x[n] \oplus r';\\
\quad\quad \textbf{reveal } r' \oplus x[0,n) \oplus r\\
\quad]|\\
\textbf{else} \ldots
\end{array}
$$

$$
\begin{array}{l}
\ldots \textbf{else}\\
\quad \textbf{reveal } m \oplus x[0] \oplus r\\
\textbf{fi}\\
m:= r;\\
n:= 0
\end{array}
$$

Reveal calculus;
use n=1 in **else**

$r{:\in}\ \{0,1\};$
$\textbf{if}\ n{>}1\ \textbf{then}$
$\quad n{:=}\ n{-}1;$
$\quad |[\ \textbf{hid}\ r'\ \cdot$
$\qquad r'{:\in}\ \{0,1\};$
$\qquad \textbf{reveal}\ m \oplus x[n] \oplus r';$
$\qquad \textbf{reveal}\ r' \oplus x[0,n) \oplus r$
$\quad]|$
$\textbf{else} \ldots$

$\ldots\textbf{else}$
$\quad \textbf{reveal}\ m \oplus x[0] \oplus r$
$\textbf{fi}$
$m{:=}\ r;$
$n{:=}\ 0$

Reorder; remove assignment to *n*

$r{:\in}\ \{0,1\};$

if $n{>}1$ **then**

$\quad n{:=}\ n{-}1;$

$\quad |[\ $ **hid** $r'\ \cdot$

$\qquad r'{:\in}\ \{0,1\};$

$\qquad$ **reveal** $m \oplus x[n] \oplus r';$

$\qquad$ **reveal** $m \oplus x[0,n] \oplus r$

$\quad]|$

else ...

... **else**

$\quad$ **reveal** $m \oplus x[0,n) \oplus r$

fi

$m{:=}\ r;$

$n{:=}\ 0$

Reorder; remove assignment to *n*

$$\ldots \textbf{else}$$
$$\quad \textbf{reveal } m \oplus x[0,n) \oplus r$$
$$\textbf{fi}$$
$$m{:=}\ r;$$
$$r{:\in}\ \{0,1\}; \qquad n{:=}\ 0$$
$$\textbf{if } n{>}1 \textbf{ then}$$
$$\quad [\![\ \textbf{hid } r' \cdot$$
$$\qquad r'{:\in}\ \{0,1\};$$
$$\qquad \textbf{reveal } m \oplus x[n] \oplus r';$$
$$\qquad \textbf{reveal } m \oplus x[0,n] \oplus r$$
$$\quad]\!]$$
$$\textbf{else} \ldots$$

Reorder

$$\ldots\ \textbf{else}$$
$$\quad \textbf{reveal}\ m \oplus x[0,n) \oplus r$$
$$\textbf{fi}$$
$$m{:=}\ r;$$
$$n{:=}\ 0$$

$$r{:\in}\ \{0,1\};$$
$$\textbf{if}\ n{>}1\ \textbf{then}$$
$$\quad [\![\ \textbf{hid}\ r' \cdot$$
$$\qquad r'{:\in}\ \{0,1\};$$
$$\qquad \textbf{reveal}\ m \oplus x[n{-}1] \oplus r';$$
$$\qquad \textbf{reveal}\ m \oplus x[0,n) \oplus r$$
$$\quad]\!]$$
$$\textbf{else}\ \ldots$$

Reorder

$$
\begin{array}{l}
r{:\in}\ \{0,1\};\\
\textbf{if}\ n{>}1\ \textbf{then}\\
\quad |[\ \textbf{hid}\ r'\ \cdot\\
\qquad r'{:\in}\ \{0,1\};\\
\qquad \textbf{reveal}\ m \oplus x[n{-}1] \oplus r';\\
\quad]|\\
\\
\textbf{reveal}\ m \oplus x[0,n) \oplus r\\
m{:=}\ r;\\
n{:=}\ 0
\end{array}
$$

Encryption lemma

$$
\begin{array}{l}
r{:\in}\ \{0,1\};\\
\textbf{if}\ n{>}1\ \textbf{then}\\
\quad |[\ \textbf{hid}\ r'\ \cdot\\
\qquad r'{:\in}\ \{0,1\};\\
\qquad \textbf{reveal}\ m \oplus x[n{-}1] \oplus r'\\
\quad]|\\
\textbf{fi};\\
\textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
m{:=}\ r;\\
n{:=}\ 0
\end{array}
$$

Encryption lemma

$$r{:\in}\ \{0,1\};$$

$$\textbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$m{:=}\ r;$$
$$n{:=}\ 0$$

Variable *n* is visible

$$
\begin{aligned}
& r{:}{\in}\ \{0,1\};\\
& \textbf{if}\ n{>}1\ \textbf{then skip fi};\\
& \textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
& m{:}{=}\ r;\\
& n{:}{=}\ 0
\end{aligned}
$$

Variable *n* is visible

$$r{:}\in \{0,1\};$$
$$\mathbf{reveal}\ m \oplus x[0,n) \oplus r;$$
$$m{:}= r;$$
$$n{:}= 0$$

And we started with...

$$r {:\in} \{0, 1\};$$
$$\textbf{reveal}\ m \oplus x[0, n) \oplus r;$$
$$m {:=} r;$$
$$n {:=} 0$$

And we started with...

$$
\begin{aligned}
&\text{— Assume } n>0\\
&r{:\in}\ \{0,1\};\\
&\textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
&m{:=}\ r;\\
&n{:=}\ 0
\end{aligned}
$$

$$
\begin{aligned}
&r{:\in}\ \{0,1\};\\
&\textbf{reveal}\ m \oplus x[0,n) \oplus r;\\
&m{:=}\ r;\\
&n{:=}\ 0
\end{aligned}
$$

The complete program,
with the circle closed

reveal $x[0, N]$

$$
\begin{array}{l}
[\![\ \textbf{hid}\ l, m, r : \{0,1\} \cdot \\
\quad l :\in \{0,1\};\ \ m := l; \\
\quad n := N; \\
\quad \textbf{repeat} \\
\qquad n := n-1; \\
\qquad r :\in \{0,1\}; \\
\qquad \textbf{reveal}\ m \oplus x[n] \oplus r; \\
\qquad m := r \\
\quad \textbf{until}\ n=0; \\
\quad \textbf{reveal}\ r \oplus x[N] \oplus l \\
]\!]
\end{array}
$$