Formal Security Analysis of Basic Network-Attached Storage

Avik Chaudhuri Martín Abadi

Department of Computer Science
University of California, Santa Cruz

ABSTRACT

We study formal security properties of network-attached storage (NAS) in an applied pi calculus. We model NAS as an implementation of a specification based on traditional centralized storage. We show the correctness of the implementation by proving that it is fully abstract with respect to the specification. Our result can be viewed as a strong guarantee of security for a basic network-attached storage design.

Categories and Subject Descriptors

F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*; D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security, Verification, Languages, Theory

Keywords

Secure storage, full abstraction, applied pi calculus

1. INTRODUCTION

In recent years, formal techniques have played a significant role in the design and analysis of secure communication protocols. In particular, there has been much research in developing process calculi, type systems, logics, and verification methods [6, 3, 2, 12, 8, 7, 1, 5, 4] to describe and reason about such protocols. In comparison, there has been far less formal work in studying secure storage. At the same time, it may be argued that storage is fast assuming a pervasive role in modern computing—and secure storage, perhaps, becoming as important a subject to understand as secure communication. We aim to bring to the study of secure storage the same level of formal treatment—and through similar techniques, with the hope that they will prove as successful as they have been in the study of secure communication.

In this paper, we model an implementation of storage that separates disk operation services from file system administration. Such systems are called *network-attached storage (NAS) systems* [9], since disks are put on the network, as opposed to being indirectly interfaced via the file system. Clients can request operations directly at the disks; such requests are guided by previous communication with file servers that provide metadata, authorization tokens, *etc.* to ensure correct service. The key advantage of such an implementation over centralized storage is that each disk operation request need not pass through a file server; information provided by the file servers can be reused for multiple disk operation requests. Not surprisingly, this scheme allows remarkable improvements in performance.

However, decoupling file system administration and disk operation services makes NAS harder to analyze. In particular, centralized storage systems employ mechanisms like access control lists to allow straightforward security assurances; it is not immediately obvious whether such guarantees hold in the distributed architecture of NAS. We approach this problem by comparing NAS with a specification based on traditional centralized storage. More precisely, we formulate a strong security criterion for NAS based on the well-known notion of *full abstraction* [17], and prove the correctness of the NAS implementation with respect to the specification under this notion. Full abstraction has been studied as an important concept for understanding the problem of implementing secure systems [1]; it has also been used for establishing security properties of various communication mechanisms [5, 4].

The rest of the paper is organized as follows. Section 2 gives an overview of the formal language we use to describe our models, which is an applied pi calculus. Section 3 presents a specification of storage based on centralized storage systems; Section 4 presents a basic NAS implementation. Section 5 develops some theory and states our main theorem, namely full abstraction of NAS. Section 6 outlines the proofs. Section 7 discusses related work; Section 8 concludes with an discussion of contributions and future work. We include the semantics of the calculus in Appendix A, and a simple type system that identifies well-formed NAS systems in Appendix B.

2. MODELING LANGUAGE

We use an applied polyadic synchronous pi calculus [18, 3] to describe and reason about processes. The syntax is standard, except for the addition of a constructor for generating keyed message authentication codes (MACs), and a destructor to extract messages from them.

Terms M, N ::=

	$m, n, \ldots$	name
$\mid$	$\langle M, N\rangle$	pair
$\mid$	$\mathbf{0}$	zero
$\mid$	$\mathbf{suc}(M)$	successor
$\mid$	$\mathbf{mac}(M, N)$	keyed MAC
$\mid$	$x, y, \ldots$	variable

The term $\mathbf{mac}(M, N)$ stands for the cryptographic hash of message M under key N. The term M can be extracted using a MAC extraction process (see below). Moreover, the terms $\mathbf{mac}(M, N)$ and $\mathbf{mac}(M', N')$ are equal iff $M' = M$ and $N' = N$; thus it is not possible to forge a MAC using an incorrect message or an incorrect key. We use the letters $u, v, \ldots$ to represent names or variables. The notation $\vec{t}$ stands for a vector $t_1, \ldots, t_k$ with $k \geq 0$.

Processes P, Q ::=

	$\overline{u}\langle\vec{M}\rangle.P$	output
$\mid$	$u(\vec{x}).P$	input
$\mid$	$P \mid Q$	composition
$\mid$	$(\nu n)P$	restriction
$\mid$	$!P$	replication
$\mid$	$[M = N]P$ else Q	match branch
$\mid$	0	nil
$\mid$	let $\langle x, y\rangle = M$ in P	split
$\mid$	case M of $\mathbf{0} : P$ $\mathbf{suc}(x) : Q$	integer case
$\mid$	M codes x in P else Q	MAC extraction

Input, split, integer case, and MAC extraction processes bind variables; restriction processes bind names. The sets $\mathsf{fv}(P)$ and $\mathsf{fn}(P)$ collect free variables and free names in P respectively. We write $P\{M/x\}$ for the capture-free substitution of each free occurrence of x in P by M. A process P is closed iff $\mathsf{fv}(P) = \emptyset$. We identify processes upto renaming of bound names and variables.

Intuitively, the constructs of the language have the following meanings:

– An output process $\overline{m}\langle\vec{M}\rangle.P$ is ready to output on channel m. If an interaction occurs, term sequence $\vec{M}$ is communicated on m and then process P runs.

– An input process $m(\vec{x}).P$ is ready to input from channel m. If an interaction occurs in which term sequence $\vec{M}$ is communicated on m, then process $P\{\vec{M}/\vec{x}\}$ runs.

– A composition $P|Q$ behaves as processes P and Q running in parallel. Each may interact with the other on channels known to both, or with the outside world, independently of the other.

– A restriction $(\nu n)P$ is a process that makes a new, private name n, and then behaves as P.

– A replication $!P$ behaves as an infinite number of copies of P running in parallel.

– A match branch $[M = N]P$ else Q behaves as P if terms M and N are the same, as Q otherwise.

– The nil process 0 does nothing.

– A split process let $\langle x, y\rangle = M$ in P behaves as $P\{M_1/x, M_2/y\}$ if term M is $\langle M_1, M_2\rangle$. Otherwise, the process is stuck, that is, it does nothing.

– An integer case process case M of $\mathbf{0} : P$ $\mathbf{suc}(x) : Q$ behaves as P if term M is $\mathbf{0}$, as $Q\{N/x\}$ if M is $\mathbf{suc}(N)$. Otherwise, the process is stuck.

– A MAC extraction process M' codes x in P else Q behaves as $P\{M/x\}$ if term M' is $\mathbf{mac}(M, N)$, as Q otherwise.

The operational semantics of the language is given via a reduction relation and a commitment relation, straightforward variants of those of the spi calculus [6]. For the sake of completeness, the semantics of the language is included in Appendix A.

Some other useful syntactic constructs can be encoded using the basic syntax [18].

– processes with internal choice: $P + Q$;

– parameterized process and function definitions: $P(\vec{x}) = Q$, $f(\vec{x}) = M$;

– process invocations and function calls within processes: $P(\vec{M})$, let $\langle\vec{z}\rangle = f(\vec{M})$ in P;

– booleans, lists, membership conditions: $[M \in N]P$.

We use the following common syntactic abbreviations.

– $u(\vec{x}) = u(\vec{x}).0$, $\overline{u}\langle\vec{M}\rangle = \overline{u}\langle\vec{M}\rangle.0$

– $\langle M_1, M_2, \ldots, M_k\rangle = \langle M_1, \langle M_2, \ldots, M_k\rangle\rangle$ $(k \geq 3)$

– Suppose $\mathcal{I} = \{i_1, \ldots, i_k\}$,
then $(\nu_{i\in\mathcal{I}} n_i)P = (\nu n_{i_1}) \ldots (\nu n_{i_k})P$
$\Pi_{i\in\mathcal{I}} P_i = P_{i_1} \mid \ldots \mid P_{i_k}$
and $\Sigma_{i\in\mathcal{I}} P_i = P_{i_1} + \cdots + P_{i_k}$

– $[M = N]P$ = $[M = N]P$ else 0

– $[M \neq N]P$ else Q = $[M = N]Q$ else P

– let $x = M$ in P = let $\langle x, z\rangle = \langle M, \mathbf{0}\rangle$ in P, where z is fresh.

– let $\langle x_1, x_2, \ldots, x_k\rangle = M$ in P
= let $\langle x_1, z\rangle = M$ in let $\langle x_2, \ldots, x_k\rangle = z$ in P, where z is fresh $(k \geq 3)$.

– M codes $\langle x_1, \ldots, x_k\rangle$ in P else Q
= M codes z in let $\langle x_1, \ldots, x_k\rangle = z$ in P else Q, where z is fresh $(k \geq 2)$.

– M codes x in P = M codes x in P else 0

3. SPECIFYING IDEAL STORAGE

We show the specification of an "ideal" storage system based on a centralized storage model. The aim of such a specification is two-fold. We try to capture the essence of a storage system interacting with clients in a distributed environment, while still keeping the model straightforward to reason about. We also use the model as an abstraction for studying the more detailed network-attached storage implementation in subsequent sections.

We view storage state as a generic map from terms to terms. In other words, given an identifier (which may be a file handle, block address, *etc.*, encoded as a name, integer, or a more complex data structure), the storage state returns some contents (encoded as a ground term). In what follows, we use the word "file name" to mean arguments to the storage state, with the understanding that the exact implementation of a storage system may use any suitable representation for identifying storage contents. The disk is an interface to the storage state—in other words, it provides well-defined functions that allow external entities to access or modify the storage state via file names. We denote each of these functions by a symbol, and collect these function symbols in $\mathcal{O}$. A term of the form $\mathsf{o}(M_1, \ldots, M_k)$ with $\mathsf{o} \in \mathcal{O}$ is called a "file operation". An operation can be thought of as a symbolic command, that is, a function symbol with argument terms, for example, $\mathsf{write}(M)$; the function symbol itself is separated out when specifying access tuples, as shown later in the section. We let f range over file names, *op* over file operations, and s over storage states. The abstract function $\textsc{do}(op, f, s)$ returns a pair $\langle s', r\rangle$, where s' is the storage state after performing operation op on file f in state s, and r is the result of the operation. A possible specification of semantics is shown below.

$$\begin{aligned} \textsc{do}(\mathsf{read}, f, s) &= \quad \langle s, s(f)\rangle \\ \textsc{do}(\mathsf{write}(M), f, s) &= \quad \langle s[f \mapsto M], \circ\rangle \end{aligned}$$

This specification could, for example, be implemented with storage states as lists:

$$\begin{aligned} & s(f) \equiv \text{case } s \text{ of} \\ & \quad \mathsf{nil} \qquad\qquad\qquad : \bot \\ & \quad \mathsf{cons}(\langle f', M\rangle, s') \quad : \text{if } f = f' \text{ then } M \text{ else } s'(f) \\ & s[f \mapsto M] \equiv \mathsf{cons}(\langle f, M\rangle, s) \end{aligned}$$

For the sake of simplicity, we assume that the file system interface is the same as the disk interface, that is, the same file names and operations are used at both levels. This assumption abstracts away various details of an actual implementation—file names may be (an appropriate encoding of) strings at the file system interface, which might then be translated to integer handles or block numbers to interface with the disk; parts of a file may be stored on different drives; and so on. Typically an implementation would also maintain tables that map identifiers across levels; one could substitute these mappings to obtain a uniform interface at all levels.

We also assume secure communication channels as necessary, by declaring new names in the modeling language (using restriction) and preventing such names from being leaked outside their intended scope. Such channels may be made secure physically, by using cryptography, or by some other means.

In what follows, we describe the components of an ideal storage system in a distributed environment. The system has as participants a group of honest clients (defined later in the section), a file server, and a disk. We allow the system to run in a potentially hostile environment, which we leave unspecified. While analyzing the security of the system, we allow this environment to be an arbitrary process which may contain dishonest clients, *etc.* The file server mediates requests for file operations from *all* clients (honest or dishonest), based on a local table of access tuples. If it finds a request authorized, it forwards the request to the disk via a secure channel, and the disk replies back with the result of the operation.

We assume that each client is identified by a distinct index (example, port number) that it uses to authenticate itself when sending operation requests. To enforce discipline in the way requests are constructed and sent to the server, we demand that honest clients use high-level "macros" to request operations. These macros get compiled into secure low-level code.

Definition 3.1 (Action on port i). *The macro* $\textsc{let}_i\ r = op(f)\ \textsc{in}\ P$*, which expands to*

$$(\nu n)\, \overline{\beta_i}\langle op, f, n\rangle.\, n(r).\, P$$

(where $n \notin \mathsf{fn}(P)$), is used to request a file operation.

Intuitively, a client may request the operation op on file f on port i, get the result in variable r, and continue as process P. The compiled low-level code works as follows: a fresh channel n is created, the request tuple $\langle op, f, n\rangle$ is sent on channel β_i (to the server), the result r received on n, and P continued. By creating and forwarding a fresh "return" channel, and keeping the channel β_i secret, we wish to guarantee that the result is unambiguously associated with the request, and is not leaked to or tampered by the attacker on its way back from the disk. Dishonest clients send requests in much the same way—by sending the tuple $\langle op, f, n\rangle$ on a channel β_j—except that n need not be fresh and neither n nor β_j need be kept secret.

Let $\mathcal{K}$ be the set of client indices. Further, let $\mathcal{I} \subseteq \mathcal{K}$ index *honest clients*, as defined next.

Definition 3.2. *An honest client is any closed process C with the following properties:*

- *it does not contain any explicit occurrence of the names β_k for any $k \in \mathcal{K}$, that is, $\beta_k \notin \mathsf{fn}_{macros}(C)$ for all $k \in \mathcal{K}$, where fn_{macros} collects free names before expanding macros,*
- *all action macros use the same port, that is, if C contains two action macros, one on port i_1 and the other on port i_2, then $i_1 = i_2$.*

In what follows, we shall index honest clients by $\mathcal{I}$, and let an honest client with index $i \in \mathcal{I}$ use only action macros on port i. (There is, however, no such restriction on dishonest clients, as noted above—they are part of an arbitrary attacker.)

Indices are subjects in access control in our model, and an index is indicated by the channel a request is sent on. In other words, when the server receives a request on channel β_k, it decides access rights for subject k. Next we define access tuples, which form the basis of access control.

Definition 3.3. *An access tuple is of the form $\langle k, \mathsf{o}, f\rangle$ where $k \in \mathcal{K}$ and $\mathsf{o} \in \mathcal{O}$.*

We write $\widetilde{op}$ to denote the function symbol in op (for example, $\widetilde{\mathsf{read}} = \mathsf{read}$ and $\widetilde{\mathsf{write}(M)} = \mathsf{write}$ for all M). The access tuple $\langle k, \mathsf{o}, f\rangle$ gives subject k the right to perform on file f any operation op such that $\widetilde{op} = \mathsf{o}$.

We now show the code that specifies an ideal storage system. As mentioned earlier, the participants are a group of honest clients C_i ($i \in \mathcal{I}$), a file server S, and a disk D. The file server mediates requests for file operations from all clients: it listens on channels β_k ($k \in \mathcal{K}$) for requests of the form $\langle op, f, n\rangle$, and enforces access control with the help of a local table T of access tuples. If it finds the request authorized, it forwards the request to the disk on a private channel γ, and the disk replies back directly with the result of operation op on f on the return channel n.

Definition 3.4. *An ideal storage system is of the form $(\nu_{i\in\mathcal{I}}\beta_i)(\Pi_{i\in\mathcal{I}}C_i \mid (\nu\gamma)(S \mid D))$, where*

- *each C_i is an honest client, containing action macros on port i,*
- *S is the* file server $(\nu\delta)(T$
  $\mid \Pi_{k\in\mathcal{K}}\, !\beta_k(op, f, n).\, (\nu t)\, \overline{\delta}\langle k, op, f, t\rangle.$
  $t().\, \overline{\gamma}\langle op, f, n\rangle)$,
- *T is an* access table
  $!\delta(k, op, f, t).\, [\langle k, \widetilde{op}, f\rangle \in \mathcal{A}]\, \overline{t}\langle\rangle$,
  where $\mathcal{A}$ is a set of access tuples,

- *D is the* disk $(\nu d)\ (\overline{d}\langle s_0\rangle$
  $\quad |\ !\gamma(op, f, n).\ d(s).\ let\ \langle s', r\rangle = \text{DO}(op, f, s)$
  $\qquad in\ \overline{n}\langle r\rangle.\ \overline{d}\langle s'\rangle),$
  where s_0 is the initial storage state.

Note that the request channels $\{\beta_i \,|\, i \in \mathcal{I}\}$ are restricted to use within honest clients and the server. It follows that an attacker cannot modify a request made by an honest client, nor can it intercept the response from the disk.

To allow for sufficient concurrency, the table T is modeled as a separate process capable of handling multiple requests from the server S on a private channel δ for resolving access. Access is resolved by checking the membership of the corresponding access tuple in $\mathcal{A}$. Each such request is of the form $\langle k, op, f, t\rangle$, where t is a fresh channel invented to recognize the outcome of the particular test. We do not provide means to change $\mathcal{A}$; in other words, we assume that access rights are initialized once and never modified (see Section 8). This restriction may be removed in various ways; however we do not further deal with this issue in this paper.

The disk D uses a private channel d to pass the storage state within itself. The current state s is first obtained by listening on d, then the abstract function DO is called to get the result r and new state s'; r is sent back on the return channel, and s' emitted on d. A subtle consequence of this protocol is that d serves as a lock to guarantee mutual exclusion of operations.

4. NETWORK-ATTACHED STORAGE

Next we describe an implementation of network-attached storage (NAS). The key difference between this and traditional storage systems is that the tasks of mediating access and servicing requests for file operations are managed by separate entities over a network. In particular, file servers are responsible for providing metadata and issuing *capabilities* to authorized clients; disks are responsible for directly servicing file-operation requests that are certified by capabilities obtained from the server. This separation of tasks can result in performance gains. When clients send requests directly to the disk, authorization checks and metadata processing are not performed for every request; server load is thus reduced.

We study NAS as a refinement of the ideal storage system specified in the previous section. As such, the NAS model builds on ideas and definitions introduced earlier. As in Section 3, we extend the syntax of processes with some macro definitions, meant to be used by clients.

DEFINITION 4.1 (AUTHORIZATION ON PORT i). *The macro* $\text{AUTH}_i\ \kappa$ FOR $op(f)$ IN P, *which expands to*

$$(\nu c)\ \overline{\alpha_i}\langle op, f, c\rangle.\ c(\kappa).\ P$$

(where $c \notin \mathsf{fn}(P)$), is used to request authorization. The variable κ gets bound to a capability *at runtime.*

DEFINITION 4.2 (ACTION USING κ ON PORT i). *The macro* $\text{LET}_i\ r = op(f)$ USING κ IN P, *which expands to*

$$(\nu n)\ \overline{\beta_i}\langle \kappa, op, f, n\rangle.\ n(r).\ P$$

(where $n \notin \mathsf{fn}(P)$), is used to request an authorized file operation.

We require honest clients to use the authorization macro to obtain capabilities from the file server. A capability is an unforgeable token that certifies that a particular file operation is authorized. In the model, a capability for operation op on f is implemented as a keyed MAC of the form $\mathbf{mac}(\langle k, op, f\rangle, \mathrm{K})$, where K is a secret key shared between the server and the disk and $\langle k, \widetilde{op}, f\rangle$ is the access right that witnesses the capability. A file-operation request is serviced by the disk only if it is accompanied by a correct capability. Verification is easy using the shared secret key.

Intuitively, a client may request authorization for an operation op on f on port i, get back a capability in variable κ, and continue as P. The low-level code works as follows: a fresh private channel c is created, and the request $\langle op, f, c\rangle$ sent to the server on channel α_i; the capability is received back on c, and P continued. Once a capability is obtained for a particular file operation, it may be used any number of times for requesting service at the disk. This is done using action macros on port i. Intuitively, the operation op on f may be requested with a previously acquired capability κ for the same operation, the result stored in variable r and P continued. At the low level, a new return channel n is created and the request $\langle \kappa, op, f, n\rangle$ is sent on β_i to the disk; r is received on n and P continued.

A client is honest if it uses only authorization and action macros to request file operations. Moreover, capabilities are used responsibly—that is, any capability obtained by an honest client via an authorization request may be used only in a subsequent disk service request by itself; conversely, every capability used in a disk service request must be previously obtained by itself via an authorization request.

DEFINITION 4.3. *An honest NAS client C' is any closed process (with macros) with the following properties:*

- *it does not contain explicit occurrences of the names α_k or β_k for any $k \in \mathcal{K}$, that is, $\alpha_k, \beta_k \notin \mathsf{fn}_{macros}(C')$ for all $k \in \mathcal{K}$, where fn_{macros} collects free names before expanding macros,*

- *all authorization and action macros use the same port, that is, if C' contains two macros, one on port i_1 and the other on port i_2, then $i_1 = i_2$,*

- *any capability obtained with an authorization macro may only be used to accompany an enclosed action macro—thus for each subprocess* $\text{AUTH}_i\ \kappa$ FOR $op(f)$ IN P, *the only uses of κ in P are in processes of the form* $\text{LET}_i\ r = op(f)$ USING κ IN Q,

- *the capability accompanying any action macro is bound by an appropriate enclosing authorization macro—thus for every subprocess* $\text{LET}_i\ r = op(f)$ USING κ IN P, *there is some subprocess* $\text{AUTH}_i\ \kappa$ FOR $op(f)$ IN Q, *with Q containing the former, such that* no other *subprocess of Q containing the former binds κ, op, or f.*

The first two conditions are similar to those on honest clients in ideal storage systems. The third condition ensures that capabilities obtained by an honest client are never passed on to any other client, either by direct communication or via the disk; further, capabilities are never used in match-branch or MAC message extraction processes by an honest client. The last condition ensures that any capability accompanying a file-operation request by an honest client is obtained by itself, and is a correct capability for the particular file operation.

Observe that client honesty is a static property—we show some simple well-formedness rules in Appendix B that provide a sufficient condition for client honesty. Of course, dishonest clients may also obtain capabilities from the server and request operations with capabilities at the disk (see below), except that they need not follow any of the rules.

We now show the code implementing a network-attached storage system. The participants are, as in Section 3, a system of honest clients C'_i, a file server S', and a disk D', running in an unspecified environment. The file server mediates access requests from all clients—it listens on channels α_k $(k \in \mathcal{K})$ for requests of the form $\langle op, f, c\rangle$, and enforces access control with the help of a local table T of access tuples. If it finds the request authorized, it sends back a capability to certify this fact. The disk services file-operation requests from all clients: it listens on channels β_k $(k \in \mathcal{K})$ for requests of the form $\langle \kappa, op, f, n\rangle$, verifies that κ is indeed a capability for op on f, and replies back with the result of the operation. We use the abbreviation $[\kappa \text{ cap of } \langle op, f\rangle, \mathrm{K}]P$ for

κ codes $\langle k, \_, \_\rangle$ in $[\kappa = \mathbf{mac}(\langle k, op, f\rangle, \mathrm{K})]P$,

where k is fresh in P.

Definition 4.4. *A network-attached storage system is of the form* $(\nu_{i\in\mathcal{I}}\alpha_i\beta_i)(\Pi_{i\in\mathcal{I}}C'_i \mid (\nu \mathrm{KK}_\perp)(S' \mid D'))$, *where*

- *each C'_i is an honest NAS client containing authorization and action macros on port i,*
- *S' is the* file server $(\nu\delta)(T$
  $\mid \Pi_{k\in\mathcal{K}} \,!\alpha_k(op, f, c).\,(\nu t)\,\overline{\delta}\langle k, op, f, t\rangle.\,t(y).$
  $[y = \top]\,\overline{c}\langle\mathbf{mac}(\langle k, op, f\rangle, \mathrm{K})\rangle$
  $\mathit{else}\ \overline{c}\langle\mathbf{mac}(\langle k, op, f\rangle, \mathrm{K}_\perp)\rangle)$,
- *T is an* access table
  $!\delta(k, op, f, t).\,[\langle k, \widetilde{op}, f\rangle \in \mathcal{A}]\,\overline{t}\langle\top\rangle\ \mathit{else}\ \overline{t}\langle\perp\rangle$,
  where $\mathcal{A}$ is a set of access tuples,
- *D' is the* disk $(\nu d)\,(\overline{d}\langle s_0\rangle$
  $\mid \Pi_{k\in\mathcal{K}}\,!\beta_k(\kappa, op, f, n).$
  $[\kappa\ \mathit{cap\ of}\ \langle op, f\rangle, \mathrm{K}]\ d(s).$
  $\mathit{let}\ \langle s', r\rangle = \textsc{do}(op, f, s)$
  $\mathit{in}\ \overline{n}\langle r\rangle.\,\overline{d}\langle s'\rangle)$,
  where s_0 is the initial storage state.

The key $\mathrm{K}_\perp$ is used by the server to generate fake capabilities. NAS clients cannot distinguish between real and fake capabilities until they use them for disk operations. We assume that s_0 satisfies the following condition: for all file names f, $\mathrm{K}, \mathrm{K}_\perp \notin \mathsf{fn}(s_0(f))$. This ensures that no client can obtain a capability by reading from the initial disk state. Similarly, we assume that $\mathcal{A}$ satisfies the following condition: for all access tuples $\langle k, \mathsf{o}, f\rangle \in \mathcal{A}$, $\mathrm{K}, \mathrm{K}_\perp \notin \mathsf{fn}(\langle k, \mathsf{o}, f\rangle)$.

The code for the access table T does not change much from the previous section; the only difference is that an access lookup always returns with a "capability-like" term. This property is necessary to obtain a faithful map of behaviors between NAS and ideal storage.

5. SECURITY IN NETWORK-ATTACHED STORAGE

We study security properties of network-attached storage by viewing the NAS model of the previous section as an implementation of the ideal storage specification of Section 3. To this end, we define an abstraction function Φ that maps a NAS system to an ideal storage system. The purpose of the abstraction function is much the same as that of a *refinement mapping* [14]—it is used to prove that network-attached storage is a "correct" implementation of ideal storage. However a refinement mapping would not suffice, since a NAS system is not necessarily a refinement of its abstraction in the sense of Lamport [14] (namely, inclusion of behaviors)—indeed, NAS observably uses capabilities while ideal storage does not.

The abstraction function Φ is defined below.

Definition 5.1 (Abstraction). *Let* $\mathrm{NAS} = (\nu_{i\in\mathcal{I}}\alpha_i\beta_i)(\Pi_{i\in\mathcal{I}}C'_i \mid (\nu\mathrm{KK}_\perp)(S' \mid D'))$ *be an arbitrary network-attached storage system. Then* $\Phi\mathrm{NAS} = (\nu_{i\in\mathcal{I}}\beta_i)(\Pi_{i\in\mathcal{I}}\lceil C'_i\rceil \mid (\nu\gamma)(S \mid D))$,
where

$\lceil 0\rceil = 0$
$\lceil(\nu n)P\rceil = (\nu n)\,\lceil P\rceil$
$\lceil P \mid Q\rceil = \lceil P\rceil \mid \lceil Q\rceil$
$\lceil u(\vec{x}).P\rceil = u(\vec{x}).\,\lceil P\rceil$
$\lceil\overline{u}\langle\vec{M}\rangle.P\rceil = \overline{u}\langle\vec{M}\rangle.\,\lceil P\rceil$
$\lceil !P\rceil = !\lceil P\rceil$
$\lceil[M = N]P\ \mathit{else}\ Q\rceil = [M = N]\lceil P\rceil\ \mathit{else}\ \lceil Q\rceil$
$\lceil \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ P\rceil = \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ \lceil P\rceil$
$\lceil \mathit{case}\ M\ \mathit{of}\ \mathbf{0} : P\ \mathbf{suc}(x) : Q\rceil$
$\quad = \mathit{case}\ M\ \mathit{of}\ \mathbf{0} : \lceil P\rceil\ \mathbf{suc}(x) : \lceil Q\rceil$
$\lceil M\ \mathit{codes}\ x\ \mathit{in}\ P\ \mathit{else}\ Q\rceil = M\ \mathit{codes}\ x\ \mathit{in}\ \lceil P\rceil\ \mathit{else}\ \lceil Q\rceil$
$\lceil \textsc{auth}_i\ \kappa\ \textsc{for}\ op(f)\ \textsc{in}\ P\rceil = \lceil P\rceil$
$\lceil \textsc{let}_i\ r = op(f)\ \textsc{using}\ \kappa\ \textsc{in}\ P\rceil$
$\quad = \textsc{let}_i\ r = op(f)\ \textsc{in}\ \lceil P\rceil$

The code for the NAS server-disk subsystem is simply replaced by the code for the ideal server-disk subsystem. To abstract honest NAS clients, we "erase" the use of capability variables in the code—thus NAS authorization macros $\textsc{auth}_i\ \kappa\ \textsc{for}\ op(f)\ \textsc{in}\ P$ are recursively replaced by residues P, and NAS action macros $\textsc{let}_i\ r = op(f)\ \textsc{using}\ \kappa\ \textsc{in}\ P$ are recursively replaced by ideal storage action macros $\textsc{let}_i\ r = op(f)\ \textsc{in}\ P$. Observe that the definition of NAS client honesty precludes incorrect usage of capability variables in action macros, or their usage in other kinds of processes, thus making the abstraction function almost trivial to write. Without these assumptions (as with dishonest clients), however, it might be expected that abstraction would be more involved. Indeed, the proof of the main theorem (Section 6) relies on defining abstraction on arbitrary code employing dishonest clients; in this case, the abstraction function uses dynamic checks to enforce correspondence between the concrete and abstract levels.

The following definition introduces *attackers* formally; these act as environments for our storage systems.

Definition 5.2. *An attacker is an arbitrary closed process.*

We think of an attacker as arbitrary code running in parallel with a system. In particular, it could contain code for dishonest clients, code for exploiting them, colluding with them, and more.

Definition 5.3. *A test is a pair $\langle E, c\rangle$ where E is an attacker and c is a name.*

Since we are primarily interested in safety properties, we use may-testing [19, 6] as a means of observing the system. A test on a system may be viewed as an attacker that tries to induce a ground output action by executing in composition with the system. A successful test, also called an *attack*, is one in which the attacker *may* induce the ground output action.

DEFINITION 5.4 (SUCCESSFUL TEST). *A closed process P passes the test $\langle E, c\rangle$ iff $E \mid P \xrightarrow{\tau}{}^{\star} \xrightarrow{\overline{c}}$.*

The notation $P \xrightarrow{\tau}{}^{\star} \xrightarrow{\overline{c}}$ means that the process P may commit zero or more silent actions followed by an output on c (see Appendix A).

We next define correctness for the NAS implementation in terms of full abstraction. Informally, an implementation is fully abstract if it preserves equivalence, that is, if two concrete systems are equivalent if and only if their abstractions are equivalent. We say that two closed processes are equivalent if they pass the same tests, that is, if no attacker can distinguish between them.

DEFINITION 5.5 (TESTING APPROXIMATION). *Let P and Q be a pair of closed processes. Then $P \sqsubseteq Q$ iff Q passes all tests passed by P.*

Informally, $P \sqsubseteq Q$ means that P is "safer" than Q, in the sense that P does not allow any more observations than Q does.

DEFINITION 5.6 (TESTING EQUIVALENCE). *Let P and Q be two closed processes. Then P is equivalent to Q ($P \simeq Q$) iff P and Q pass the same tests, that is, $P \sqsubseteq Q$ and $Q \sqsubseteq P$.*

We now state our main result, namely full abstraction of NAS.

THEOREM 5.7 (FULL ABSTRACTION). *Let* NAS_1 *and* NAS_2 *be two arbitrary network-attached storage systems. Then* $\mathrm{NAS}_1 \simeq \mathrm{NAS}_2$ *iff* $\Phi\mathrm{NAS}_1 \simeq \Phi\mathrm{NAS}_2$.

Full abstraction is proved in two parts. The first part states that testing approximation is preserved on abstraction, that is, a pair of NAS systems related under testing approximation remain related when they are abstracted.

THEOREM 5.8. *Let* NAS_1 *and* NAS_2 *be a pair of network-attached storage systems. If* $\mathrm{NAS}_1 \sqsubseteq \mathrm{NAS}_2$*, then* $\Phi\mathrm{NAS}_1 \sqsubseteq \Phi\mathrm{NAS}_2$.

This theorem may be read as an adequacy result. The second part is more interesting, and states the converse.

THEOREM 5.9. *Let* NAS_1 *and* NAS_2 *be a pair of network-attached storage systems. If* $\Phi\mathrm{NAS}_1 \sqsubseteq \Phi\mathrm{NAS}_2$*, then* $\mathrm{NAS}_1 \sqsubseteq \mathrm{NAS}_2$.

Full abstraction is a powerful property, and implies many interesting security properties. Recall that two systems are equivalent if no attacker can distinguish them. A system can be proved secure by showing its equivalence to another system, whose security has already been established, or is easy to see. Full abstraction of NAS implies that if two ideal storage systems are indistinguishable, then any corresponding NAS implementations of these systems are indistinguishable as well. We show two examples that illustrate how full abstraction yields specific guarantees for NAS systems. The first example concerns secrecy; the second one concerns authenticity.

EXAMPLE 5.10 (SECRECY). Consider an ideal storage system *Ideal* with one honest client
$\text{LET}_i\ \_ = \mathsf{write}(M)(f)$,
with $\mathcal{A} = \{\langle i, \mathsf{write}, f\rangle\}$ and M a closed term. An attacker cannot know that what has been written to f, since it cannot read it. This is a secrecy property, and can be expressed as $\mathit{Ideal} \simeq \mathit{Ideal}'$, where Ideal' is an ideal storage system with one honest client
$\text{LET}_i\ \_ = \mathsf{write}(\mathbf{0})(f)$,
and the same $\mathcal{A}$. Let NAS' be a trivial implementation of Ideal', with one honest client
$\text{AUTH}_i\ \kappa\ \text{FOR}\ \mathsf{write}(\mathbf{0})(f)\ \text{IN}$
$\text{LET}_i\ \_ = \mathsf{write}(\mathbf{0})(f)\ \text{USING}\ \kappa$.
Theorem 5.7 guarantees that for any implementation NAS such that $\Phi\mathit{NAS} = \mathit{Ideal}$, $\mathit{NAS} \simeq \mathit{NAS}'$ holds, that is, no attacker can get any information about M from NAS.

EXAMPLE 5.11 (AUTHENTICITY). Let *Ideal* consist of two honest clients
$(\nu m)\text{LET}_{i_1}\ \_ = \mathsf{write}(m)(f)\ \text{IN}\ m(x).\ [x \neq \mathbf{0}]\ \overline{c}\langle\rangle$
and
$\text{LET}_{i_2}\ m = \mathsf{read}(f)\ \text{IN}\ \overline{m}\langle\mathbf{0}\rangle$,
with $\mathcal{A} = \{\langle i_1, \mathsf{write}, f\rangle, \langle i_2, \mathsf{read}, f\rangle\}$. An attacker cannot forge a different message on m, since it cannot know m—m is a fresh name that it cannot guess, and is written to a file it cannot read. This is an authenticity property, and may be expressed by stating that an attacker cannot induce output on c. Thus we may write $\mathit{Ideal} \simeq \mathit{Ideal}'$, where Ideal' is an ideal storage system with two honest clients
$\text{LET}_i\ \_ = \mathsf{write}(\mathbf{0})(f)$ and 0,
and the same $\mathcal{A}$. Let NAS' be a trivial implementation of Ideal'. Theorem 5.7 guarantees that for any implementation NAS such that $\Phi\mathit{NAS} = \mathit{Ideal}$, $\mathit{NAS} \simeq \mathit{NAS}'$ holds, so no attacker can forge a different message on s in NAS.

6. OUTLINE OF PROOFS

We state some definitions and lemmas that help to prove the main result.

The concretization function Ψ translates an arbitrary ideal storage system to a network-attached storage system, as follows.

DEFINITION 6.1 (CONCRETIZATION). *Let* Ideal *be any ideal storage system* $(\nu_{i\in\mathcal{I}}\beta_i)(\Pi_{i\in\mathcal{I}} C_i \mid (\nu\gamma)(S \mid D))$. *Then* $\Psi\text{Ideal} = (\nu_{i\in\mathcal{I}}\alpha_i\beta_i)(\Pi_{i\in\mathcal{I}}\lfloor C_i\rfloor \mid (\nu\mathrm{KK}_\perp)(S' \mid D'))$*, where*

$$\begin{aligned}
&\lfloor 0\rfloor = 0\\
&\lfloor(\nu n)P\rfloor = (\nu n)\ \lfloor P\rfloor\\
&\lfloor P \mid Q\rfloor = \lfloor P\rfloor \mid \lfloor Q\rfloor\\
&\lfloor u(\vec{x}).P\rfloor = u(\vec{x}).\ \lfloor P\rfloor\\
&\lfloor \overline{u}\langle\vec{M}\rangle.P\rfloor = \overline{u}\langle\vec{M}\rangle.\ \lfloor P\rfloor\\
&\lfloor !P\rfloor = !\lfloor P\rfloor\\
&\lfloor [M = N]P\ \mathit{else}\ Q\rfloor = [M = N]\lfloor P\rfloor\ \mathit{else}\ \lfloor Q\rfloor\\
&\lfloor \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ P\rfloor = \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ \lfloor P\rfloor\\
&\lfloor \mathit{case}\ M\ \mathit{of}\ \mathbf{0} : P\ \mathbf{suc}(x) : Q\rfloor\\
&\qquad = \mathit{case}\ M\ \mathit{of}\ \mathbf{0} : \lfloor P\rfloor\ \mathbf{suc}(x) : \lfloor Q\rfloor\\
&\lfloor M\ \mathit{codes}\ x\ \mathit{in}\ P\ \mathit{else}\ Q\rfloor = M\ \mathit{codes}\ x\ \mathit{in}\ \lfloor P\rfloor\ \mathit{else}\ \lfloor Q\rfloor\\
&\lfloor \text{LET}_i\ r = \mathit{op}(f)\ \text{IN}\ P\rfloor\\
&\qquad = \text{AUTH}_i\ \kappa\ \text{FOR}\ \mathit{op}(f)\ \text{IN}\\
&\qquad\qquad \text{LET}_i\ r = \mathit{op}(f)\ \text{USING}\ \kappa\ \text{IN}\ \lfloor P\rfloor,
\end{aligned}$$

where κ is fresh in P.

PROPOSITION 6.2 (SANITY). $\Phi \circ \Psi$ *is the identity function on ideal storage systems.*

We define concretization and abstraction functions for environments as well. The concretization function Ψ translates attackers for ideal storage systems to attackers for concretized NAS systems. Let $\mathcal{J} = \mathcal{K} \backslash \mathcal{I}$.

Definition 6.3 (Concretization of attackers). *Let E be an ideal storage attacker. $\Psi E = \lfloor E \rfloor$, where*

$$\begin{aligned}
&\lfloor 0 \rfloor = 0\\
&\lfloor (\nu n) P \rfloor = (\nu n)\ \lfloor P \rfloor\\
&\lfloor P \mid Q \rfloor = \lfloor P \rfloor \mid \lfloor Q \rfloor\\
&\lfloor u(\vec{x}).P \rfloor = u(\vec{x}).\ \lfloor P \rfloor\\
&\lfloor \overline{u}\langle op, f, n\rangle.P \rfloor\\
&\quad = \Sigma_{j \in \mathcal{J}} [u = \beta_j]\ \textsc{auth}_j\ \kappa\ \textsc{for}\ op(f)\ \textsc{in}\\
&\qquad\qquad\qquad \overline{\beta_j}\langle \kappa, op, f, n\rangle.\ \lfloor P \rfloor\\
&\qquad + \overline{u}\langle op, f, n\rangle.\ \lfloor P \rfloor\\
&\lfloor \overline{u}\langle \vec{M}\rangle.P \rfloor = \overline{u}\langle \vec{M}\rangle.\ \lfloor P \rfloor,\ \textit{if}\ |\vec{M}| \neq 3\\
&\lfloor !P \rfloor = !\ \lfloor P \rfloor\\
&\lfloor [M = N] P\ \textit{else}\ Q \rfloor = [M = N] \lfloor P \rfloor\ \textit{else}\ \lfloor Q \rfloor\\
&\lfloor \textit{let}\ \langle x, y\rangle = M\ \textit{in}\ P \rfloor = \textit{let}\ \langle x, y\rangle = M\ \textit{in}\ \lfloor P \rfloor\\
&\lfloor \textit{case}\ M\ \textit{of}\ \mathbf{0} : P\ \mathbf{suc}(x) : Q \rfloor\\
&\quad = \textit{case}\ M\ \textit{of}\ \mathbf{0} : \lfloor P \rfloor\ \mathbf{suc}(x) : \lfloor Q \rfloor\\
&\lfloor M\ \textit{codes}\ x\ \textit{in}\ P\ \textit{else}\ Q \rfloor = M\ \textit{codes}\ x\ \textit{in}\ \lfloor P \rfloor\ \textit{else}\ \lfloor Q \rfloor
\end{aligned}$$

Concretization of attackers closely follows concretization of honest clients, except that we need to keep in mind that attackers may not satisfy the conditions imposed on honest clients. In particular there may be no discipline in the use of the names β_j and capabilities obtained from the server.

Proposition 6.4. *For any network-attached storage system* NAS, ΦNAS *passes* $\langle E', c\rangle$ *iff* NAS *passes* $\langle \Psi E', c\rangle$.

Proof of Theorem 5.8. If ΦNAS_1 passes $\langle E', c\rangle$, then by Proposition 6.4, NAS_1 passes $\langle \Psi E', c\rangle$, which implies that NAS_2 passes $\langle \Psi E', c\rangle$, which implies (again by Proposition 6.4) that ΦNAS_2 passes $\langle E', c\rangle$. ☐

The abstraction function Φ for NAS attackers is given next.

Definition 6.5 (Abstraction of attackers). *Let E be a NAS attacker,* $\mathrm{A} = \{\alpha_j \in \mathsf{fn}(E) \mid j \in \mathcal{J}\}$, *and* $\mathrm{B} = \{\beta_j \in \mathsf{fn}(E) \mid j \in \mathcal{J}\}$. *We abbreviate the code*

$$\begin{aligned}
&\kappa\ \textit{codes}\ \langle j, \_, \_\rangle\ \textit{in}\\
&\quad ([\kappa = \mathbf{mac}(\langle j, op, f\rangle, \mathrm{K}_\perp)]\ \overline{\beta_j}\langle op, f, n\rangle.\ Q\ \textit{else}\ Q)\\
&\textit{else}\ Q
\end{aligned}$$

as $\textsc{sim}(\kappa, op, f, n)/\mathrm{K}_\perp/Q$. *Then* $\Phi E = (\nu \mathrm{K}_\perp)(\lceil E \rceil \mid S_E)$, *where*

$$\begin{aligned}
&\lceil 0 \rceil = 0\\
&\lceil (\nu n) P \rceil = (\nu n)\ \lceil P \rceil\\
&\lceil P \mid Q \rceil = \lceil P \rceil \mid \lceil Q \rceil\\
&\lceil u(\vec{x}).P \rceil = u(\vec{x}).\ \lceil P \rceil\\
&\lceil \overline{u}\langle \kappa, op, f, n\rangle.P \rceil\\
&\quad = \Sigma_{b \in \mathrm{B}} [u = b]\ \textsc{sim}(\kappa, op, f, n)/\mathrm{K}_\perp/\lceil P \rceil\\
&\qquad + \overline{u}\langle \kappa, op, f, n\rangle.\ \lceil P \rceil\\
&\lceil \overline{u}\langle \vec{M}\rangle.P \rceil = \overline{u}\langle \vec{M}\rangle.\ \lceil P \rceil,\ \textit{if}\ |\vec{M}| \neq 4\\
&\lceil !P \rceil = !\ \lceil P \rceil\\
&\lceil [M = N] P\ \textit{else}\ Q \rceil = [M = N] \lceil P \rceil\ \textit{else}\ \lceil Q \rceil\\
&\lceil \textit{let}\ \langle x, y\rangle = M\ \textit{in}\ P \rceil = \textit{let}\ \langle x, y\rangle = M\ \textit{in}\ \lceil P \rceil\\
&\lceil \textit{case}\ M\ \textit{of}\ \mathbf{0} : P\ \mathbf{suc}(x) : Q \rceil\\
&\quad = \textit{case}\ M\ \textit{of}\ \mathbf{0} : \lceil P \rceil\ \mathbf{suc}(x) : \lceil Q \rceil\\
&\lceil M\ \textit{codes}\ x\ \textit{in}\ P\ \textit{else}\ Q \rceil = M\ \textit{codes}\ x\ \textit{in}\ \lceil P \rceil\ \textit{else}\ \lceil Q \rceil\ ,\ \textit{and}
\end{aligned}$$

$$S_E = \Pi_{\alpha_j \in \mathrm{A}}\ !\alpha_j(op, f, c).\ \overline{c}\langle \mathbf{mac}(\langle j, op, f\rangle, \mathrm{K}_\perp)\rangle.$$

The abstract attacker simulates the effects of obtaining capabilities from the NAS server by creating a dummy server that issues fake capabilities (that is, under a different secret key). Given a fake capability, an abstracted attacker can extract the subject whose rights are certified by the associated original capability in the NAS system, and use this information to request the file operation on the subject's port.

The function $\widehat{\ }$ translates terms and actions in network-attached storage systems to "appropriate" terms and actions in ideal storage systems.

Definition 6.6. *The function $\widehat{\ }$ homomorphically maps capabilities issued by the NAS server to similar terms (issued by the dummy server) under a different secret key.*

$$\begin{aligned}
&\widehat{\mathrm{K}} = \mathrm{K}_\perp\\
&\widehat{\langle M, N\rangle} = \langle \widehat{M}, \widehat{N}\rangle\\
&\widehat{\mathbf{suc}(M)} = \mathbf{suc}(\widehat{M})\\
&\widehat{\mathbf{mac}(M, N)} = \mathbf{mac}(\widehat{M}, \widehat{N})\\
&\widehat{\mathsf{o}(M_1, \ldots, M_k)} = \mathsf{o}(\widehat{M_1}, \ldots, \widehat{M_k})\\
&\widehat{s} = s'\ \textit{such that for all}\ f,\ s'(f) = \widehat{s(f)}\\
&\widehat{\star} = \star\ \textit{(otherwise)}
\end{aligned}$$

The following relation shows correspondence between states of a NAS implementation and states of its abstraction as an ideal storage system.

Definition 6.7 (Abstraction relation). *Let* NAS *be a network-attached storage system, and E be an attacker. In any run of $E \mid$ NAS, every state can be written in the form $(\nu \mathrm{K}\mathrm{K}_\perp)(\nu \vec{n})(e \mid (\nu_{i \in \mathcal{I}} \alpha_i \beta_i)(\Pi_{i \in \mathcal{I}} c'_i \mid \varsigma'))$ upto structural equivalence [6], where e, ς', and c'_i ($i \in \mathcal{I}$) are components of the state for the attacker, the server-disk subsystem, and the honest clients, and $\vec{n}$ contains shared bound names.*

The notation close($\mathcal{S}$) *means $\{\langle \sigma P', \widehat{\sigma} P\rangle$ s.t. $\langle P', P\rangle \in \mathcal{S}$; σ maps free variables in P' to ground terms; and $\widehat{\sigma} = \widehat{\ } \circ \sigma\}$. We assume (without loss of generality) that E and each C'_i are written so that distinct variable binding instances bind distinct variables. The relations $R_{C'_i}$, $R_{S'|D'}$ and R_E are defined in Figures 1, 2, and 3. The abstraction relation* $\mathcal{R} =$

$$\begin{aligned}
&\{\langle (\nu \mathrm{K}\mathrm{K}_\perp)(\nu \vec{n})(e \mid (\nu_{i \in \mathcal{I}} \alpha_i \beta_i)(\Pi_{i \in \mathcal{I}} c'_i \mid \varsigma')),\\
&\quad (\nu \mathrm{K}_\perp)(\nu \vec{n})(e' \mid (\nu_{i \in \mathcal{I}} \beta_i)(\Pi_{i \in \mathcal{I}} c_i \mid (\nu \gamma)\varsigma))\rangle
\end{aligned}$$

such that $\langle c'_i, c_i\rangle \in R_{C'_i}$, $\langle \varsigma', \varsigma\rangle \in R_{S'|D'}$, $\langle e, e'\rangle \in R_E$, and $\vec{n}$ contains shared bound names$\}$.

Observe that $\mathcal{R}$ contains the pair $\langle E \mid NAS, \Phi E \mid \Phi NAS\rangle$.

Lemma 6.8. *Let $\mathcal{R}$ be the abstraction relation derived from* NAS *and E, and let $\langle N, I\rangle$ be an arbitrary pair in $\mathcal{R}$. Then*

– *if $N \xrightarrow{\tau} N'$, then there exists I' such that $I \xrightarrow{\tau}{}^\star I'$ and $\langle N', I'\rangle \in \mathcal{R}$; conversely, if $I \xrightarrow{\tau} I'$, then there exists N' such that $N \xrightarrow{\tau}{}^\star N'$ and $\langle N', I'\rangle \in \mathcal{R}$;*

– *if $N \xrightarrow{\overline{c}}$, then $I \xrightarrow{\overline{c}}$; conversely, if $I \xrightarrow{\overline{c}}$, then $N \xrightarrow{\tau}{}^\star \xrightarrow{\overline{c}}$.*

Theorem 6.9 (Testing refinement). *Let* NAS *be a network-attached storage system and $\langle E, c\rangle$ be a test. Then* NAS *passes* $\langle E, c\rangle$ *iff* ΦNAS *passes* $\langle \Phi E, c\rangle$.

Proof. By induction on test runs, using Lemma 6.8. ☐

Observe that $\mathsf{fn}(\Phi E) \subseteq \mathsf{fn}(E)$ follows from the definition of Φ on attackers. Theorem 6.9 is interesting in itself—it can be read as a safety preservation theorem. Informally, it says

$R_{C_i'} = \mathit{close}(\|C_i'\|)$, where the function $\|\_\|$ is as defined below.

$$\begin{array}{l}
\|0\| = \{\langle 0, 0\rangle\} \\
\|(\nu n)P\| = \{\langle(\nu n)P',\ (\nu n)X'\rangle \\
\qquad\qquad\qquad : \langle P', X'\rangle \in \|P\|\} \\
\|P \mid Q\| = \{\langle P'|Q',\ X'|Y'\rangle \\
\qquad\qquad\qquad : \langle P', X'\rangle \in \|P\|, \langle Q', Y'\rangle \in \|Q\|\} \\
\|u(\vec{x}).P\| = \{\langle u(\vec{x}).P, u(\vec{x}).X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|\overline{u}\langle\vec{M}\rangle.P\| = \{\langle\overline{u}\langle\vec{M}\rangle.P, \overline{u}\langle\vec{M}\rangle.X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|!P\| = \{\langle !P|\, \Pi_{l=1}^{r} P_l,\ !X|\, \Pi_{l=1}^{r} X_l\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle, \langle P_1, X_1\rangle, \ldots, \langle P_r, X_r\rangle \in \|P\|\} \\
\|[M = N]P \text{ else } Q\| = \{\langle [M = N]P \text{ else } Q, [M = N]X \text{ else } Y\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|, \langle Q, Y\rangle \in \|Q\|\} \quad \cup \|P\| \cup \|Q\| \\
\|\text{let } \langle x, y\rangle = M \text{ in } P\| = \\
\quad \{\langle \text{let } \langle x, y\rangle = M \text{ in } P, \text{ let } \langle x, y\rangle = M \text{ in } X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|\text{case } M \text{ of } \mathbf{0} : P\ \mathbf{suc}(x) : Q\| = \\
\quad \{\langle \text{case } M \text{ of } \mathbf{0} : P\ \mathbf{suc}(x) : Q, \text{ case } M \text{ of } \mathbf{0} : X\ \mathbf{suc}(x) : Y\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|, \langle Q, Y\rangle \in \|Q\|\} \quad \cup \|P\| \cup \|Q\| \\
\|M \text{ codes } x \text{ in } P \text{ else } Q\| = \\
\quad \{\langle M \text{ codes } x \text{ in } P \text{ else } Q, M \text{ codes } x \text{ in } X \text{ else } Y\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|, \langle Q, Y\rangle \in \|Q\|\} \quad \cup \|P\| \cup \|Q\| \\
\|\textsc{auth}_i\ \kappa \text{ FOR } op(f) \text{ IN } P\| = \\
\quad \{\langle(\nu c)\, \overline{\alpha_i}\langle op, f, c\rangle.\, c(x).\, P,\ X\rangle, \\
\quad\ \langle c(x).\, P,\ X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|\textsc{let}_i\ r = op(f) \text{ USING } \kappa \text{ IN } P\| = \\
\quad \{\langle(\nu n)\, \overline{\beta_i}\langle\kappa, op, f, n\rangle.\, n(r).\, P,\ (\nu n)\, \overline{\beta_i}\langle op, f, n\rangle.\, n(r).\, X\rangle, \\
\quad\ \langle n(r).\, P,\ n(r).\, X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\|
\end{array}$$

Figure 1: Abstraction relation between honest clients

that an observation that can be induced by an attacker on a NAS system can also be induced by an attacker with as much initial knowledge on the abstracted system.

Proof of Theorem 5.9. Assume $\Phi NAS_1 \sqsubseteq \Phi NAS_2$. Suppose NAS_1 passes $\langle E, c\rangle$, then ΦNAS_1 passes $\langle \Phi E, c\rangle$ by Theorem 6.9. By assumption, ΦNAS_2 passes $\langle \Phi E, c\rangle$, and finally by Theorem 6.9, NAS_2 passes $\langle E, c\rangle$. $\square$

The following is a corollary to Theorem 5.9, and can be read as dual to Proposition 6.2. Abstraction followed by concretization may rearrange, prune, or replicate authorization requests in honest clients; the resulting NAS system is equivalent to the original.

Proposition 6.10. *Let* NAS *be a network-attached storage system, then* $\Psi\Phi\text{NAS} \simeq \text{NAS}$.

Proof. Assume the contrary. Then $\Phi\Psi\Phi NAS \not\simeq \Phi NAS$ by Theorem 5.9. But $\Phi\Psi\Phi NAS = \Phi NAS$ by Proposition 6.2—this contradicts the assumption. $\square$

7. RELATED WORK

There has been a lot of interest in the security architecture of network-attached storage ever since its inception [9, 11, 10, 20, 16, 21]. However the design of security in most NAS implementations (NASD [11], SCARED [20], SNAD [16], SNARE [21]) has been semi-formal. Schemes for secure storage have often been described in detail using a mix of cryptography and clever data structures; yet the targeted security properties have seldom been formally articulated or proved. Some exceptions are the work of Mazières and

Let S^0 and S'^0 be parts of server code such that $S = (\nu\delta)S^0$ and $S' = (\nu\delta)S'^0$. Let $D_?$ be part of disk code such that $D = (\nu d)(\overline{d}\langle s_0\rangle \mid D_?)$; define D_s as $\overline{d}\langle s\rangle \mid D_?$. Similarly, suppose $D' = (\nu d)(\overline{d}\langle s_0\rangle \mid D'_?)$; define D'_s as $\overline{d}\langle s\rangle \mid D'_?$. Write $\{k, op, f\}_{\mathrm{Key}}$ for $\mathbf{mac}(\langle k, op, f\rangle, \mathrm{Key})$, and define the abbreviations:
$\textsc{repl}(op, f, s)/n/d = \text{let } \langle s', r\rangle = \textsc{do}(op, f, s) \text{ in } \overline{n}\langle r\rangle.\overline{d}(s')$.
$\textsc{cert}(k, op, f)/y/c = [y = \top]\overline{c}\langle\{k, op, f\}_{\mathrm{K}}\rangle \text{ else } \overline{c}\langle\{k, op, f\}_{\mathrm{K}_\perp}\rangle$.

$$\begin{array}{l}
\text{Let } \|S'\|^E = \\
\ \{\langle(\nu t)\, \overline{\delta}\langle i, op, f, t\rangle.\, t(y).\, \textsc{cert}(i, op, f)/y/c, \quad 0\rangle, \\
\quad \langle(\nu t)\, \overline{\delta}\langle j, op, f, t\rangle.\, t(y).\textsc{cert}(j, op, f)/y/c, \quad \overline{c}\langle\{j, op, f\}_{\mathrm{K}_\perp}\rangle\rangle, \\
\quad \langle(\nu t)\, t(y).\, \textsc{cert}(i, op, f)/y/c \mid [\langle i, \widetilde{op}, f\rangle \in \mathcal{A}]\, \overline{t}\langle\top\rangle \text{ else } \overline{t}\langle\perp\rangle, \quad 0\rangle, \\
\quad \langle(\nu t)\, t(y).\, \textsc{cert}(j, op, f)/y/c \mid [\langle j, \widetilde{op}, f\rangle \in \mathcal{A}]\, \overline{t}\langle\top\rangle \text{ else } \overline{t}\langle\perp\rangle, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \overline{c}\langle\{j, op, f\}_{\mathrm{K}_\perp}\rangle\rangle, \\
\quad \langle\textsc{cert}(i, op, f)/y/c, \quad 0\rangle, \\
\quad \langle\textsc{cert}(j, op, f)/y/c, \quad \overline{c}\langle\{j, op, f\}_{\mathrm{K}_\perp}\rangle\rangle, \\
\quad \langle\overline{c}\langle\{i, op, f\}_{\mathrm{K}}\rangle, \quad 0\rangle, \\
\quad \langle\overline{c}\langle\{j, op, f\}_{\mathrm{K}}\rangle, \quad \overline{c}\langle\{j, op, f\}_{\mathrm{K}_\perp}\rangle\rangle, \\
\quad \langle\overline{c}\langle\{i, op, f\}_{\mathrm{K}_\perp}\rangle, \quad 0\rangle, \\
\quad \langle\overline{c}\langle\{j, op, f\}_{\mathrm{K}_\perp}\rangle, \quad \overline{c}\langle\{j, op, f\}_{\mathrm{K}_\perp}\rangle\rangle \ : \ i \in \mathcal{I}, j \in \{j' \mid \alpha_{j'} \in \mathrm{A}\}\} \\
\\
\|D'\|^S = \\
\ \{\langle[\kappa \text{ cap of } \langle op, f\rangle, \mathrm{K}]\, d(s).\, \textsc{repl}(op, f, s)/n/d, \\
\qquad\qquad\qquad\qquad (\nu t)\, t().\, \overline{\gamma}\langle\widehat{op}, \widehat{f}, n\rangle \mid [\langle k, \widetilde{\widehat{op}}, \widehat{f}\rangle \in \mathcal{A}]\, \overline{t}\langle\rangle\rangle \\
\qquad\quad : \ \kappa = \{k, op, f\}_{\mathrm{K}} \Leftrightarrow \langle k, \widetilde{op}, f\rangle \in \mathcal{A}\} \\
\\
\|D'\| = \\
\ \{\langle D'_s, D_{\hat{s}}\rangle, \quad \langle\textsc{repl}(op, f, s)/n/d\, |D'_?, \quad \textsc{repl}(op, f, s)/n/d\, |D_?\rangle\} \\
\\
R_{S'|D'} = \\
\ \{\langle(\nu\delta)(\Pi_p s'_p|S'^0) \mid (\nu d)(\Pi_q d'_q|d'_r), \\
\qquad\qquad\qquad\qquad (\Pi_p s_p|S_E) \mid (\nu\delta)(\Pi_q s_q|S^0) \mid (\nu d)(d_r)\rangle \\
\qquad\quad : \ \langle s'_p, s_p\rangle \in \mathit{close}(\|S'\|^E), \langle d'_q, s_q\rangle \in \mathit{close}(\|D'\|^S), \\
\qquad\qquad \langle d'_r, d_r\rangle \in \mathit{close}(\|D'\|) \text{ and } p, q, r \text{ index finite sets}\}
\end{array}$$

Figure 2: Abstraction relation between server-disk subsystems

$R_E = \mathit{close}(\|E\|)$, where the function $\|\_\|$ is defined as below.

$$\begin{array}{l}
\|0\| = \{\langle 0, 0\rangle\} \\
\|(\nu n)P\| = \{\langle(\nu n)P',\ (\nu n)X'\rangle \\
\qquad\qquad\qquad : \langle P', X'\rangle \in \|P\|\} \\
\|P \mid Q\| = \{\langle P'|Q',\ X'|Y'\rangle \\
\qquad\qquad\qquad : \langle P', X'\rangle \in \|P\|, \langle Q', Y'\rangle \in \|Q\|\} \\
\|u(\vec{x}).P\| = \{\langle u(\vec{x}).P, u(\vec{x}).X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|\overline{u}\langle\kappa, op, f, n\rangle.P\| = \\
\quad \{\langle\overline{u}\langle\kappa, op, f, n\rangle.P, \\
\qquad \Sigma_{b \in \mathrm{B}}[u = b]\textsc{sim}(\kappa, op, f, n)/\mathrm{K}_\perp/X \\
\qquad + \overline{u}\langle\kappa, op, f, n\rangle.X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|\overline{u}\langle\vec{M}\rangle.P\| = \{\langle\overline{u}\langle\vec{M}\rangle.P, \overline{u}\langle\vec{M}\rangle.X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \qquad \text{if } |\vec{M}| \neq 4 \\
\|!P\| = \{\langle !P|\, \Pi_{l=1}^{r} P_l,\ !X|\, \Pi_{l=1}^{r} X_l\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle, \langle P_1, X_1\rangle, \ldots, \langle P_r, X_r\rangle \in \|P\|\} \\
\|[M = N]P \text{ else } Q\| = \{\langle [M = N]P \text{ else } Q, [M = N]X \text{ else } Y\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|, \langle Q, Y\rangle \in \|Q\|\} \quad \cup \|P\| \cup \|Q\| \\
\|\text{let } \langle x, y\rangle = M \text{ in } P\| = \\
\quad \{\langle \text{let } \langle x, y\rangle = M \text{ in } P, \text{ let } \langle x, y\rangle = M \text{ in } X\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|\} \quad \cup \|P\| \\
\|\text{case } M \text{ of } \mathbf{0} : P\ \mathbf{suc}(x) : Q\| = \\
\quad \{\langle \text{case } M \text{ of } \mathbf{0} : P\ \mathbf{suc}(x) : Q, \text{ case } M \text{ of } \mathbf{0} : X\ \mathbf{suc}(x) : Y\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|, \langle Q, Y\rangle \in \|Q\|\} \quad \cup \|P\| \cup \|Q\| \\
\|M \text{ codes } x \text{ in } P \text{ else } Q\| = \\
\quad \{\langle M \text{ codes } x \text{ in } P \text{ else } Q, M \text{ codes } x \text{ in } X \text{ else } Y\rangle \\
\qquad\qquad\qquad : \langle P, X\rangle \in \|P\|, \langle Q, Y\rangle \in \|Q\|\} \quad \cup \|P\| \cup \|Q\|
\end{array}$$

Figure 3: Abstraction relation between attackers

Shasha on a formal notion of data integrity for a system with untrusted remote storage (SUNDR [15]), and Gobioff's analysis of the NASD protocol using belief logics [10].

Our approach towards formalizing "what security means" in the context of NAS derives various ideas from verification and programming languages theory. Proofs of correctness of the NAS implementation with respect to the abstracted ideal storage specification rely in part on techniques similar to refinement mappings [14]. Full abstraction as a security property has been studied in [1], and used, for example, in proving the correctness of secure implementations of channel abstractions [5] and authentication primitives [4], and also in the analysis of programming language security guarantees [1, 13].

8. CONCLUSION

In this paper we show a formal approach to modeling secure storage systems. We state and prove the correctness of NAS in terms of a powerful notion of security, namely full abstraction, by comparing the implementation with a specification of traditional centralized storage.

Our result is quite general, and says that NAS systems are as secure as their abstractions as "ideal" centralized storage systems. This result may be applied to derive specific guarantees for specific schemes. Furthermore, by making minimal assumptions on the behavior of clients, our result scales to systems where interaction with storage devices is only a part of client behavior. One might imagine that security analysis for such composite systems would be made simpler by "plugging in" abstracted ideal storage subsystems wherever NAS subsystems were used.

We have only begun our study in this direction; in particular, the NAS design we analyze is fairly basic. One immediate line of future work includes allowing multiple disks, dynamic administration of access, revocation of capabilities, *etc.*, in the NAS implementation to scale up the analysis to more realistic systems. Along similar lines, it would be interesting to investigate richer models for access control and data representation, with varying trust assumptions on file servers and disks.

Acknowledgments

We thank Mathieu Baudet and Cédric Fournet for their useful comments. This work was supported in part by the National Science Foundation under Grants CCR-0204162 and CCR-0208800.

9. REFERENCES

[1] M. Abadi. Protection in programming-language translations. In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 868–883. Springer-Verlag, 1998.

[2] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115. ACM Press, 2001.

[4] M. Abadi, C. Fournet, and G. Gonthier. Authentication primitives and their compilation. In *POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 302–315. ACM Press, 2000.

[5] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. *Information and Computation*, 174(1):37–83, Apr. 2002.

[6] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report SRC-RR-149, Digital Systems Research Center, Palo Alto CA, January 1998.

[7] B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, May 2004.

[8] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[9] G. A. Gibson, D. P. Nagle, K. Amiri, F. W. Chang, E. Feinberg, H. G. C. Lee, B. Ozceri, E. Riedel, and D. Rochberg. A case for network-attached secure disks. Technical Report CMU–CS-96-142, 1996.

[10] H. Gobioff. *Security for a High Performance Commodity Storage Subsystem*. PhD thesis, Carnegie Mellon University, 1999.

[11] H. Gobioff, G. Gibson, and J. Tygar. Security for network attached storage devices. Technical Report CMU-CS-97-185, Carnegie Mellon University, October 1997.

[12] A. D. Gordon and A. Jeffrey. Typing correspondence assertions for communication protocols. *Theoritical Computer Science*, 300(1-3):379–409, 2003.

[13] A. Kennedy. Untrustworthy programming languages. Talk at the Joint Queen Mary/Imperial College Seminar series, May 2005. Slides available at http://research.microsoft.com/~akenn/sec/index.html.

[14] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[15] D. Mazières and D. Shasha. Building secure file systems out of byzantine storage. In *PODC '02: Proceedings of the 21st Symposium on Principles of Distributed Computing*, pages 108–117. ACM Press, 2002.

[16] E. L. Miller, D. D. E. Long, W. E. Freeman, and B. Reed. Strong security for network-attached storage. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 1–13. USENIX Association, 2002.

[17] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.

[18] R. Milner. The polyadic pi-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[19] R. D. Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1–2):83–133, Nov. 1984.

[20] B. C. Reed, E. G. Chron, R. C. Burns, and D. D. E. Long. Authenticating network-attached storage. *IEEE Micro*, 20(1):49–57, Jan. 2000.

[21] Y. Zhu and Y. Hu. SNARE: A strong security scheme for network-attached storage. In *SRDS '03: Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 250–259. IEEE Computer Society, 2003.

APPENDIX

The appendix includes a formal semantics of the calculus (directly based on one for the spi calculus [6]) and a simple type system for establishing honesty of NAS clients.

A. FORMAL SEMANTICS OF THE CALCULUS

Reduction is defined on closed processes.

$$\begin{array}{ll}
!P \;>\; P \mid !P & \\
[M = M]P \text{ else } Q \;>\; P & \\
[M = N]P \text{ else } Q \;>\; Q & \text{if } M \neq N \\
\text{let } \langle x, y\rangle = \langle M, N\rangle \text{ in } P \;>\; P\{M/x, N/y\} & \\
\text{case } \mathbf{0} \text{ of } \mathbf{0} : P \ \mathbf{suc}(x) : Q \;>\; P & \\
\text{case } \mathbf{suc}(M) \text{ of } \mathbf{0} : P \ \mathbf{suc}(x) : Q \;>\; Q\{M/x\} & \\
\mathbf{mac}(M, N) \text{ codes } x \text{ in } P \text{ else } Q \;>\; P\{M/x\} & \\
M' \text{ codes } x \text{ in } P \text{ else } Q \;>\; Q & \text{if } M' \neq \mathbf{mac}(M, N)
\end{array}$$

The grammar of processes is extended to a grammar of agents, which include processes, abstractions, and concretions.

$$\begin{array}{lcl}
\text{Abstractions } F & ::= & (\vec{x}).P \\
\text{Concretions } C & ::= & (\nu\vec{n})\langle\vec{M}\rangle.P \\
\text{Agents } A & ::= & P \mid F \mid C
\end{array}$$

The following syntactic rearrangements are defined to extend the scope of abstractions and concretions beyond restriction and composition constructs.

$$\begin{array}{l}
(\nu m)(\vec{x}).P \triangleq (\vec{x}).(\nu m)P \\
R \mid (\vec{x}).P \triangleq (\vec{x}).(R \mid P) \quad \text{if } \vec{x} \not\subseteq \mathsf{fv}(R) \\
(\nu m)(\nu\vec{n})\langle\vec{M}\rangle.P \triangleq \\
\qquad \begin{cases} (\nu m, \vec{n})\langle\vec{M}\rangle.P & \text{if } m \in \mathsf{fn}(\vec{M})\backslash\vec{n} \\ (\nu\vec{n})\langle\vec{M}\rangle.(\nu m)P & \text{if } m \notin \mathsf{fn}(\vec{M}) \cup \vec{n} \end{cases} \\
R \mid (\nu\vec{n})\langle\vec{M}\rangle.P \triangleq (\nu\vec{n})\langle\vec{M}\rangle.(R \mid P) \quad \text{if } \vec{n} \cap \mathsf{fn}(R) = \emptyset
\end{array}$$

Abstractions and concretions reduce to give back processes. Suppose $F = (\vec{x}).P$ and $C = (\nu\vec{n})\langle\vec{M}\rangle.Q$ such that $\vec{n} \cap \mathsf{fn}(P) = \emptyset$, then

$$\begin{array}{lcl}
F \mathbin{@} C & \triangleq & (\nu\vec{n})(P\{\vec{M}/\vec{x}\} \mid Q) \\
C \mathbin{@} F & \triangleq & (\nu\vec{n})(Q \mid P\{\vec{M}/\vec{x}\})
\end{array}$$

An action may be silent, input on a name, or output on a name.

$$\begin{array}{llll}
\text{Actions } \alpha & ::= & \tau & \text{(silent)} \\
& \mid & m & \text{(input)} \\
& \mid & \overline{m} & \text{(ground output)}
\end{array}$$

The commitment relation $\longrightarrow \subseteq \text{Processes} \times \text{Actions} \times \text{Agents}$ is shown in Figure 4.

$$m(\vec{x}).P \xrightarrow{m} (\vec{x}).P \qquad \overline{m}\langle\vec{M}\rangle.P \xrightarrow{\overline{m}} (\nu)\langle\vec{M}\rangle.P$$

$$\frac{P \xrightarrow{m} F \qquad Q \xrightarrow{\overline{m}} C}{P \mid Q \xrightarrow{\tau} F \mathbin{@} C} \qquad \frac{P \xrightarrow{\overline{m}} C \qquad Q \xrightarrow{m} F}{P \mid Q \xrightarrow{\tau} C \mathbin{@} F}$$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q} \qquad \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A}$$

$$\frac{P \xrightarrow{\alpha} A \qquad \alpha \notin \{n, \overline{n}\}}{(\nu n)P \xrightarrow{\alpha} (\nu n)A} \qquad \frac{P > Q \qquad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

Figure 4: The commitment relation

B. TYPING HONESTY OF NAS CLIENTS

We define typing judgments of the form $\rho, i \vdash P$, where ρ is a function from variables to terms, $i \in \mathcal{K}$, and P is a process with NAS authorization and action macros. Let $\mathsf{occ}(\rho) = \mathsf{dom}(\rho) \cup \mathsf{fn}(\mathsf{range}(\rho)) \cup \mathsf{fv}(\mathsf{range}(\rho))$, and $\mathsf{occ}(M) = \mathsf{fn}(M) \cup \mathsf{fv}(M)$. The typing rules are shown in Figure 5.

We define NAS client honesty as a well-typedness property: *A closed process P is an honest NAS client iff $\mathsf{fn}(P) \cap \{\alpha_k, \beta_k \mid k \in \mathcal{K}\} = \emptyset$ and $\emptyset, i \vdash P$ for some $i \in \mathcal{K}$.*

$$\rho, i \vdash 0 \qquad \frac{\rho, i \vdash P \qquad \rho, i \vdash Q}{\rho, i \vdash P \mid Q} \qquad \frac{\rho, i \vdash P}{\rho, i \vdash !P}$$

$$\frac{\vec{x} \cap \mathsf{occ}(\rho) = \emptyset,\ \mathsf{occ}(u) \cap \mathsf{dom}(\rho) = \emptyset \qquad \rho, i \vdash P}{\rho, i \vdash u(\vec{x}).P}$$

$$\frac{\mathsf{occ}(u, \vec{M}) \cap \mathsf{dom}(\rho) = \emptyset \qquad \rho, i \vdash P}{\rho, i \vdash \overline{u}\langle\vec{M}\rangle.P}$$

$$\frac{n \notin \mathsf{occ}(\rho) \qquad \rho, i \vdash P}{\rho, i \vdash (\nu n)P}$$

$$\frac{\mathsf{occ}(M, N) \cap \mathsf{dom}(\rho) = \emptyset \qquad \rho, i \vdash P \qquad \rho, i \vdash Q}{\rho, i \vdash [M = N]P \text{ else } Q}$$

$$\frac{\{x, y\} \cap \mathsf{occ}(\rho) = \emptyset,\ \mathsf{occ}(M) \cap \mathsf{dom}(\rho) = \emptyset \qquad \rho, i \vdash P}{\rho, i \vdash \text{let } \langle x, y\rangle = M \text{ in } P}$$

$$\frac{x \notin \mathsf{occ}(\rho),\ \mathsf{occ}(M) \cap \mathsf{dom}(\rho) = \emptyset \qquad \rho, i \vdash P \quad \rho, i \vdash Q}{\rho, i \vdash \text{case } M \text{ of } \mathbf{0} : P \ \mathbf{suc}(x) : Q}$$

$$\frac{x \notin \mathsf{occ}(\rho),\ \mathsf{occ}(M) \cap \mathsf{dom}(\rho) = \emptyset \qquad \rho, i \vdash P \quad \rho, i \vdash Q}{\rho, i \vdash M \text{ codes } x \text{ in } P \text{ else } Q}$$

$$\frac{\kappa \notin \mathsf{occ}(\rho),\ \mathsf{occ}(op, f) \cap \mathsf{dom}(\rho) = \emptyset \quad \rho[\kappa \mapsto \langle op, f\rangle], i \vdash P}{\rho, i \vdash \text{AUTH}_i\ \kappa \text{ FOR } op(f) \text{ IN } P}$$

$$\frac{r \notin \mathsf{occ}(\rho) \qquad \rho(\kappa) = \langle op, f\rangle \qquad \rho, i \vdash P}{\rho, i \vdash \text{LET}_i\ r = op(f) \text{ USING } \kappa \text{ IN } P}$$

Figure 5: Typing rules for honest NAS clients