Call by Contract for Cryptographic Protocols*

Jonathan Millen, Joshua Guttman, John Ramsdell,
Justin Sheehy, and Brian Sniffen

The MITRE Corporation
Bedford, MA 01730
USA
jmillen,guttman,ramsdell,justin,bsniffen@mitre.org

Abstract. Call by contract is a way to specify and use interchangeable services in secure protocols, so that protocols and services can be independently designed and verified. A selection algorithm is given to test whether a candidate service is uniformly selectable. To facilitate independent security verification of the calling protocol and its services, contracts and requests also provide an NDA (Non-Disclosure Agreement). Informally, NDAs are confidentiality constraints on parameters.

1 Introduction

A compositional approach to protocol design and analysis is recognized as advantageous. We wish to perform design decomposition in a way that permits independent design and verification of components, and preserves security and correctness goals when the components are recombined. There are many different ways in which composition can be interpreted and implemented. Our version of composition applies to the design of secure protocols. Our objective is to extend verification techniques based on abstract encryption models to protocols that incorporate or implement encapsulated *services*.

One use for such services is to invoke computations that are not included in the vocabulary of operations built into the protocol specification language, such as the special operations of a Trusted Platform Module [7] or some other specialized cryptographic application interface. Another use is to allow flexibility in the choice of means to implement an operation. For a public-key lookup, for example, there may be two alternative services, one that fetches a locally cached certificate, and another that initiates a protocol with a directory server to retrieve one. A service might even establish a shared session key between two parties who have already begun a protocol.

With encapsulation of services, we can create and maintain a library of services that could be shared by all protocol processes running on the same system. The service library can be updated with improved implementations without disturbing existing protocols, and new services can be added at any time for use

* This work was supported by the National Security Agency through US Army CECOM contract W15P7T-05-C-F600.

by future protocols. New services can also be used by older protocols if they extend older services, that is, if their functionality is more general or their input requirements are less strict.

There are two challenges in designing and using services: first, to allow for use of services whose specific interfaces have not been anticipated; second, to assure ourselves that it is safe to use separately designed services when there are security as well as functionality concerns.

Our "encapsulated services" should be distinguished from "Web services" because they can be purely local. Furthermore, the security objectives that we wish to protect are the traditional confidentiality and authentication goals for secure protocols, rather than the more specific access control, execution history [2] or other policies associated with Web services. However, we need not exclude Web services as an option for implementing an encapsulated service. Our focus is on the interface to a calling protocol and on maintaining the security objectives of that protocol.

Service specifications will be expressed in a *logical call-by-contract* form. An algorithm for selecting services that satisfy protocol requests is given. We take into consideration that a service specification may include not only functional guarantees but also confidentiality guarantees, which we call *non-disclosure agreements (NDAs)*. We give some conditions under which authentication and secrecy properties established in a Dolev-Yao idealized encryption context can be carried over to a general service invocation context.

In the remainder of this section, the formal modeling context is presented. Then, in Section 2, we give the abstraction-binding technique that underlies the selection algorithm. In Section 3, we show how abstraction binding is used to select services in response to requests. In Section 4, we investigate how the conventional Dolev-Yao techniques for verification of protocol security can be extended safely to this more general context.

1.1 Modeling Style

Protocols are modeled here as sets of roles, where a role is the trace of a parameterized annotated strand. An annotated strand, as presented in [10], extends the basic strand notion from [13] by labeling nodes not only with messages but also trust management formulas. The vocabulary used in this paper is reviewed in Appendix A. Most of the details are not necessary until we apply a prior strand space result in the latter part of Section 4.

Messages in this model are elements of a free message algebra, presented as expressions constructed from atoms of a few sorts (such as key and text) and operators such as idealized encryption and pairing. Formulas are logical expressions over the same set of atoms. The threat environment is a version of the Dolev-Yao attacker model, in which there is a single attacker who is assumed able to intercept any message, and able to decompose or construct messages using the available operators, but unable to encrypt or extract information from encrypted messages without the appropriate key. Models of this kind have been widely used to analyze the vulnerability of protocols to attacks such as replay

and spoofing, in which the attacker fools an honest participant into revealing an entire key or accepting a key provided by, or previously compromised by, the attacker.

1.2 Trust Management Formulas

The germ of the call-by-contract idea was presaged in the rely-guarantee trust management approach described in [10]. The trust management formula associated with a positive (send) node is a *guarantee*, expressed in terms of the protocol parameters, that must be provable for the values bound to those parameters. The formula associated with a negative (receive) node is a *rely* formula, which may be assumed in proofs of subsequent guarantees. The *local theory* of a node consists of the initial theory T plus the rely formulas of prior nodes in the role. Rely formulas are authentication goals whose validity must be proved by the protocol designer as part of the verification of the protocol.

Note that guarantees can be active, in the sense that they bind parameter values. For example, a guarantee `pubkey(a,ka)` may be evaluated in a node prior to which the parameter `a` is bound but `ka` is unbound, and its effect is to look up the public key of principal `a` in the local theory and bind `ka` to it.

1.3 General Services

A service can be viewed as a guarantee that is provided by a separate role of the same principal, like a subprotocol. This view is illustrated in Fig. 1. A calling strand invokes the service with a call that guarantees some precondition formula γ that implies the service precondition ρ', and the service strand sends a return annotated with another formula γ' that is the service postcondition, which should imply the caller's desired rely condition ρ. The call and return messages convey the sequences of values of the input and output parameters respectively. Call and return operators could be modeled as additional operators in the free algebra which create private messages, in the sense that they are neither constructible nor analyzable by the attacker. (They could also be modeled using the special protected message directions introduced in [8].)

A service has a precondition and postcondition, as in the terminology of the Eiffel "design by contract" concept. Our focus, however, is not on how the service implements its contract, but rather on how a service selection mechanism can adapt a caller rely formula ρ to a different but sufficient service postcondition γ' that was stated with a different set of formal parameters. We also need to match the caller guarantee to the service precondition.

A service is *selectable* for a call request if there exists a parameter mapping that assigns values to service inputs from caller inputs, and caller outputs from service outputs, such that the service precondition and the caller postcondition are both satisfied.

For example, suppose that the caller requirement has `certAuth(z)` as a precondition, and its postcondition is `pubKey(a,ka)`, so that its parameter set is $P_c = \{\texttt{a,ka,z}\}$. The caller has bound `a` and `z` and needs a value for `ka`. It might

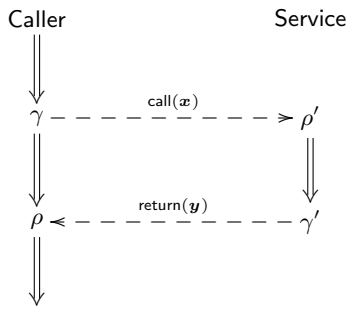


Fig. 1. Calling a service

be satisfied by a service that asks for the precondition `certAuth(ca)` and offers the postcondition `pkCert(pk,p,ca)`, and its parameters are $P_s = \{\texttt{pk,p,ca}\}$, where `ca` is a certification authority. To match the service to the call, we must find the mapping `p` $\mapsto$ `a`, `ka` $\mapsto$ `pk`, and `ca` $\mapsto$ `z`. This match is justified using an inference rule like:

```
pubKey(A,K) :- pkCert(K,A,C), certAuth(C).
```

This rule is presented in the Horn clause format used in Datalog and Prolog, for reasons given in the next section. The capital letters used as arguments are logical variables.

Where does this inference rule come from? We assume that there is a *selector theory* with conversion rules useful for translating between caller and service predicate vocabularies.

At this point it is possible to point out the differences between the service idea and the prior related concepts of guarantees and subprotocols. First, guarantees are established logically, by inference from the caller's local theory, while the mechanism by which a service establishes its postcondition is not specified. A service postcondition might result from any computation, or from a separate protocol.

A service is different from a subprotocol because it might just be a local computation, but even when it is a protocol, the calling mechanism is different. A subprotocol call identifies a subprotocol with a known name and parameters, while our services are selectable through a flexible matching mechanism that can search through a list of many subprotocols to find one that can be made to fit through a suitable parameter mapping.

1.4 Non-Disclosure Agreements

One way in which Fig. 1 is incomplete is that it does not show that the service may participate in a protocol that communicates with other principals.

Parameter values that it shares with the caller might be transmitted in protocol messages. A caller might want to insist that some shared parameter values be kept secret, either by not sending them out at all, or by sending them only to (strands owned by) some designated principals. Specifications of this kind must be provided by the caller requirement and by the service contract. Our formulation of this kind of specification is an "NDA" and it will be discussed in Section 4.

2 Abstraction-Binding Inferences

Our objective is not merely to define the call-by-contract relationship, but also to describe a practical way to implement the selection mechanism that matches services to requests. The selection mechanism takes advantage of Datalog, because we have already been using Datalog to support the rely-guarantee trust management approach. It has been implemented as part of the runtime system for a high-level protocol programming language, CPPL [8].

Datalog is a restricted form of first-order logic in which each formula is a function-free Horn clause, and every variable in the head of a clause must appear in the body of the clause. A clause that meets this condition is called a *safe* clause.

Datalog [5] forms the foundation of many deductive database systems, as well as at least one trust management language [12]. A Datalog *literal* has the form *predicate-symbol*$(t_1, ..., t_k)$, where each argument t_i is either an arity-zero function symbol (i.e., a constant) or a logical variable. The predicate symbols and constants are application-specific, and begin with lower-case letters. Variables begin with capital letters. In Datalog, a theory is a set of safe Horn clauses of the form *head* :- *body* where the head is a literal and the body is a (possibly empty) sequence of literals. A clause with an empty body is a *fact*, and a clause with at least one literal in the body is a *rule*.

The conditions on asserted clauses guarantee that the set of all facts that can be derived from a Datalog theory is finite, and there is a terminating algorithm to find all provable instances of a given literal presented as a query. This is the essential feature of Datalog for our purposes.

In order to apply Datalog to node formulas, we restrict ourselves to formulas that are expressible as conjunctions of Datalog ground literals. We use Datalog constants to represent atoms, both parameters and values. Thus, principal symbols such as `p, a, ca` and key symbols like `pk, ka` appear in literals as atoms, and are used as arguments of predicates such as `pubKey` and `certAuth`.

To apply the Datalog engine, we may *abstract* a formula by replacing some or all of its parameters with fresh, distinct variables. Other atoms are left unchanged. The abstraction step can be represented with a substitution α, with a domain given in context.

Now, suppose p and q are formulas, and T is a theory consisting of a set of rules. We wish to check whether some instance of q follows from p in the context

of T. For now, assume that q is a single literal. The Datalog engine is used as follows:

1. assert T and p;
2. present the abstracted literal $q\alpha$ as a query.

The engine will either fail (if no instance of $q\alpha$ is provable), or it will find all variable bindings β such that $q\alpha\beta$ is provable from T and p, that is,

$$T, p \vdash q\alpha\beta.$$

If we let $\sigma = \alpha\beta$, we see that we have found a substitution σ of parameters into values such that

$$T, p \vdash q\sigma$$

The combination of these two steps is *abstraction binding.*

Recall that $A = X \cup C$, and let $A(\phi)$ be the set of atoms occurring in formula ϕ. Also let $A(T) = \bigcup\{A(\phi) | \phi \text{ occurs in } T\}$. Abstraction binding has the following property:

Proposition 1. *Let T be a constant-free theory, consisting of rules such that $A(T) = \emptyset$, and let p and q be formulas. Then abstraction-binding finds the set of all substitutions σ on $A(q) \setminus A(p)$ into $A(p)$ such that $T, p \vdash q\sigma$.*

If q is a conjunction, we can achieve the same result by satisfying its conjuncts sequentially, extending σ as needed. (One way to do this is to introduce a new predicate name for q, and add a rule for it with the conjuncts of q serving as the body.)

For convenience, we define the abstraction-binding AB relation as follows:

Definition 1 (Abstraction Binding). AB (T, p, q, σ) iff σ is a substitution on $A(q) \setminus A(p)$ into $A(p)$ such that $T, p \vdash q\sigma$.

AB relations are preserved by precondition substitutions.

Proposition 2 (AB Substitution). *If* AB (T, p, q, σ) *and τ is a substitution on $A(p)$, then* AB $(T, p\tau, q, \sigma\tau)$.

This conclusion is valid because T is constant-free, so the inference cannot depend on particular choices for the parameters of p. The parameters of p are like Skolem constants, since they do not occur in T. An inference that holds with them can be generalized to a universal statement about variables in their places, which are given values by τ.

3 Selection: Call by Contract

In this section we begin by looking at a static specification of a successful service call. In a protocol with a service call, the service role is represented by an idealization of the service, in which only the initial and final nodes are present, as in Figure 1. A bundle containing a service call instantiates the caller role r_c

with some ground substitution σ_c, and the idealized service r_s with a ground substitution σ_s, which is defined only on the service parameters named in the contract.

A request from a caller has a precondition formula p_c and a postcondition formula q_c. Each service has a precondition p_s and a postcondition q_s. With the substitutions and the condition formulas just defined, Figure 1 turns into Figure 2.

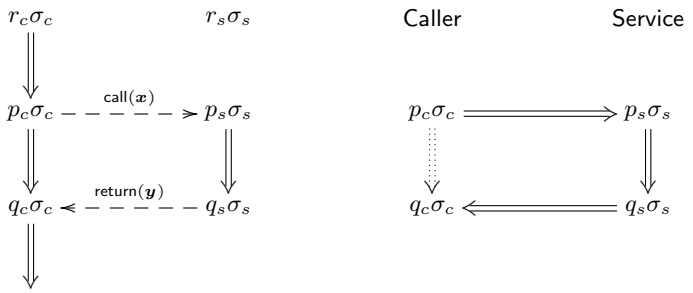


Fig. 2. General service diagram (a) and implications (b)

In Fig. 2(a), the input and output conditions do not have to match exactly, but we want them to satisfy the implications summarized in the diagram (b).

For any service, we assume that it satisfies its contract, expressed as a precondition, postcondition pair (p_s, q_s). The contract assumes that the service has been called with an assignment of values to input parameters, which are the parameters in p_s. If p_s holds with these values, the contract promises to find values for the output parameters (the parameters in q_s that are not input parameters) to satisfy q_s.

Notation. For a precondition, postcondition pair (for either a service or a caller), it is convenient to introduce symbols for its input parameters $I = A(p)$, its output parameters $O = A(q) \setminus A(p)$, and all of its parameters $P = I \cup O$. Subscripts s or c will be applied as needed in context.

Definition 2. A *service contract* is a pair of node formulas (p_s, q_s) such that $P_s \subseteq X$ and, for all σ on I_s into Z,

$$p_s\sigma \Rightarrow (\exists\tau)\ q_s\sigma\tau$$

where τ is on O_s into Z.

Definition 3. A service contract (p_s, q_s) is *independently selectable* for caller request (p_c, q_c) if, for any input substitution $\sigma_e : I_c \to Z$, there exist substitutions $\sigma_c : P_c \to Z$ extending σ_e and $\sigma_s : P_s \to Z$ such that

1. $T, p_c\sigma_c \vdash p_s\sigma_s$,
2. $p_s\sigma_s \Rightarrow q_s\sigma_s$, and
3. $T, q_s\sigma_s \vdash q_c\sigma_c$.

The qualifier "independently" reflects the condition that the same service is selectable regardless of which input substitution is chosen. Non-independent selectability would mean that a service is selectable for some inputs but not others. With a constant-free selector theory, we shall see that independent selectability is not only possible, but it can be checked as soon as the service contract and caller protocol are known.

3.1 From the Static to the Algorithmic View

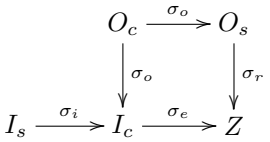
The Selection Theorem shows how selectability can be established via two abstraction-binding steps.

Theorem 1 (Selection). *Let T be a constant-free theory, and suppose that p_c, q_c, p_s, q_s are node formulas such that P_c is disjoint from P_s. Assume that*

1. $\sigma_e : I_c \to Z$,
2. (p_s, q_s) *is a service contract,*
3. $\mathsf{AB}\ (T, p_c, p_s, \sigma_i)$, *and*
4. $\mathsf{AB}\ (T, q_s\sigma_i, q_c, \sigma_o)$.

Then (p_s, q_s) is independently selectable for (p_c, q_c). Furthermore, there exists a substitution $\sigma_r : O_s \to Z$ such that $T, p_c\sigma_e \vdash q_c\sigma_o\sigma_e\sigma_r$.

Proof. We will find $\sigma_r : O_s \to Z$ from which we construct $\sigma_c = \sigma_o\sigma_e\sigma_r$ and $\sigma_s = \sigma_i\sigma_e\sigma_r$. The diagram may help in tracing the substitutions.



We apply the contract once and AB Substitution twice. Hypothesis 3 implies that $\sigma_i : I_s \to I_c$, and hypothesis 4 implies that $\sigma_o : O_c \to P_s$.

Applying hypothesis 3 with $\sigma_e : I_c \to Z$, we get $\mathsf{AB}\ (T, p_c\sigma_e, p_s, \sigma_i\sigma_e)$ by AB substitution. This means that $T, p_c\sigma_e \vdash p_s\sigma_i\sigma_e$. By construction, the first condition $T, p_c\sigma_c \vdash p_s\sigma_s$ is satisfied (since p_c and p_s have no outputs). Applying the contract in hypothesis 2, we produce $\sigma_r : O_s \to Z$ such that $q_s\sigma_i\sigma_e\sigma_r$ holds. This satisfies the second condition $p_s\sigma_s \Rightarrow q_s\sigma_s$ by construction.

By AB substitution on hypothesis 4 with $\sigma_e\sigma_r$, we find that $T, q_s\sigma_i\sigma_e\sigma_r \vdash q_c\sigma_o\sigma_e\sigma_e\sigma_r$. This gives us $T, q_s\sigma_s \vdash q_c\sigma_c$. □

The Selection Theorem gives us more than just independent selectability, since it shows that the caller and service substitutions can be factored through input and output mappings σ_i and σ_o. This turns out to be important later when we consider confidentiality constraints. The following definition names this result.

Definition 4. A service contract (p_s, q_s) is *uniformly selectable* for caller request (p_c, q_c) if, for any input substitution $\sigma_e : I_c \to Z$, there exist substitutions $\sigma_i : I_s \to I_c$ and $\sigma_o : O_c \to P_s$ such that independent selectability is satisfied by $\sigma_c = \sigma_o\sigma_e\sigma_r$ and $\sigma_s = \sigma_i\sigma_e\sigma_r$ for some $\sigma_r : O_s \to Z$.

Corollary 1. *Under the conditions of the Selection Theorem,* (p_s, q_s) *is uniformly selectable for* (p_c, q_c).

The *contract selection algorithm* implied by the Selection Theorem is to consider each available service contract (p_s, q_s). To test acceptability of a contract, apply abstraction binding to find σ_i, and if successful, apply abstraction binding again to find σ_o. If successful again, the service is selectable. Backtracking or other search methods can explore multiple candidates for σ_o and σ_i. If none work, other contracts are tested.

Note that all of this can be done without knowing σ_e. At run time, the service is invoked with the input substitution $\sigma_i\sigma_e$. When the service completes, caller outputs are bound with $\sigma_o\sigma_e\sigma_r$.

When multiple (σ_i, σ_o) solutions exist, application preferences or constraints can be used to guide the search order or otherwise choose among alternative call mappings. The contract selection algorithm as stated does not yet consider security constraints, which are discussed in the next section.

4 Separate Verification of Protocol Services

Our objective is to separate the verification as well as the design and specification of services from that of calling protocols. We need to investigate the conditions under which this is possible. In this section we provide some preliminary issues and answers, though more research is advised to obtain broader results.

In the context of the strand space approach, we are concerned with authentication goals, as expressed in protocol rely formulas, and confidentiality goals, either for their own sake, or to support authentication proofs.

Some issues regarding information flow are discussed first, then the general question of protocol separation is considered.

4.1 Information Flow Through Services

The implementation of a service could affect the soundness of the protocol, if it violates confidentiality assumptions. We need to know that the computation implementing a service does not cause information flows from a secret input parameter to an unprotected output parameter.

For example, suppose that a protocol role has a uniquely originated, fresh, random nonce `k` that we want to keep secret. And suppose there is a service with contract `(true, copy(a,b))`, where `copy(a,b)` does what it says, namely to bind `b` to the value of `a`. If this service is called to obtain a postcondition `copy(k,m)`, and then the caller sends `m` as a message, the nonce `k` has obviously been compromised.

If `copy` were just a predicate used as a guarantee, this problem would surface routinely. The behavior of `copy` would have to be defined in the local theory. The theory would include a rule like `A = B :- copy(A,B)`, and any decent security analysis would notice the consequences.

However, a `copy` service might be selected for a caller whose local theory did not even possess an equality predicate. The caller might only have requested a weaker postcondition like `same_length(k,m).`

To use services safely, then, we need to provide a way for the caller to specify confidentiality constraints, and for services to advertise their confidentiality promises. We also need a way to verify such promises, and to establish that the constraints are sufficient to preserve the security properties of the caller.

Protocol analysis based on computational models can determine whether services computed algorithmically leak significant partial information, even when they do not actually lead to an equality relation. This kind of analysis is discussed, for example, in [1]. In this paper, we just point out that such techniques exist, and our discussion on verifying confidentiality will focus on protocols and subprotocols.

4.2 NDAs

As mentioned in Section 1.4, a confidentiality requirement or contract is expressed as an NDA (non-disclosure agreement). An NDA associates a predicate on message terms with each parameter of a role. The NDA $\nu(x)$ of a parameter x expresses a constraint on the way x may be released. The predicate formula is not necessarily restricted to the language of node formulas.

In this subsection, we leave open the semantics of $\nu(x)$. The way in which an NDA is used will be presented, at first, just formally, and then in the next subsection we provide a particular interpretation that justifies some security conclusions.

Definition 5. A *secure service request* is a 4-tuple $(p_c, q_c, \sigma_e, \nu)$ of a caller precondition, a caller postcondition, an input value substitution, and an NDA defined on the caller parameters P_c. A *secure service contract* is a triple (p_s, q_s, ν) where ν is defined on the service parameters P_s.

The contract (p_s, q_s, ν) is *securely selectable* for $(p_c, q_c, \sigma_e, \nu)$ if it is uniformly selectable with parameter mappings σ_i, σ_o and

(1) if $x \in I_s$ then $\nu(x)\sigma_i \Rightarrow \nu(x\sigma_i)\sigma_o$ and
(2) if $y \in O_c$ then $\nu(y\sigma_o)\sigma_i \Rightarrow \nu(y)\sigma_o$.

Condition (1) says that the constraint on a service input is stricter than the constraint on the caller input to which it maps. Condition (2) says that the constraint on a service output is stricter than the constraint on any caller parameter mapped to it.

The parameter mappings applied to the ν sets are necessary to make them comparable. This is easy to understand if one considers the way terms are

mapped to values. With uniform selection, a caller term t gets the value $t\sigma_c = t\sigma_o\sigma_e\sigma_r$, and a service term t' gets the value $t'\sigma_s = t'\sigma_i\sigma_e\sigma_r$. So if $t\sigma_o = t'\sigma_i$ then t and t' are mapped to the same value.

4.3 Example: Binary NDA

Suppose that $\nu(x)$ is either `true`, meaning that x is releasable without restriction, or `false`, meaning that x may not be released at all. There is a trivial service contract (`nonce(x), nonce(x)`), and a selector rule `copy(A,A) :- nonce(A).` The secure service request is (`nonce(a), copy(a,b)`, σ_e, ν) where σ_e is arbitrary and $\nu(\mathtt{a}) = \nu(\mathtt{b}) = \mathtt{false}$. Note that $I_c = \{\mathtt{a}\}, O_c = \{\mathtt{b}\}, I_s = \{\mathtt{x}\}$, and $O_s = \emptyset$. What should the NDA of the service be?

First, observe that the service is uniformly selectable with $\mathtt{x}\sigma_i = \mathtt{a}$ and $\mathtt{b}\sigma_o = \mathtt{x}$.

Applying the parameter mappings, the secure selectability conditions become

(1) $\nu(\mathtt{a})\sigma_i \Rightarrow \nu(\mathtt{a})\sigma_o$ and
(2) $\nu(\mathtt{x})\sigma_i \Rightarrow \nu(\mathtt{b})\sigma_o$.

Now apply the ν values for the caller. Condition (1) is just `false = false`. Condition (2) is $\nu(\mathtt{x})\sigma_i = \mathtt{false}$. Hence we need $\nu(\mathtt{x}) = \mathtt{false}$. The service is being required to keep its input confidential.

There is a way to enforce binary NDA checking for free through the basic functional selection mechanism. For each caller input x such that $\nu(x)$ is true, add to the initial theory the assertion `public`(x). And if $\nu(x)$ is false, add the assertion `hidden`(x). Now, for both caller and service, all `public` assertions are added as conjuncts to preconditions, and all `hidden` assertions are added as conjuncts to postconditions. Thus, if a service treats an input as public, that will automatically be justified, as part of the selectability check, by checking that the caller input it maps to is public; and if a caller believes that an output is hidden, that will be justified by checking that the service treats the parameter it maps to as hidden.

4.4 Services Implemented as Protocols

Many useful services are implemented as protocols that exchange messages with a third party. For example, a service might check certificate revocation by contacting a validation authority. The "third" party might also be a party already participating in the main protocol, if the service performs some standard negotiation. Figure 3 illustrates a protocol service that exchanges messages m, m' with an instance of a role r_t. The dotted transitions indicate that the service might go through several nodes before returning to the caller.

Ordinarily, a protocol soundness proof for a protocol that invokes a subprotocol would consider the protocol and subprotocol together, verifying the composition as a single larger protocol. However, we wish to take advantage of prior work that identifies conditions under which protocol components may be verified in isolation, preserving their security properties when they are combined.

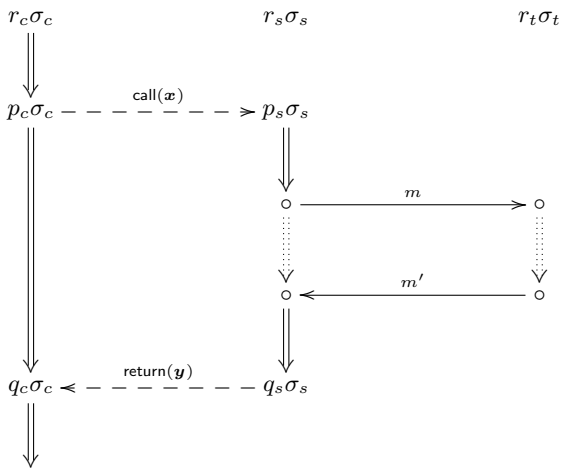


Fig. 3. Protocol Service Call

Protocols may be combined in two senses. One is by refinement, which is close to the idea of a service: a protocol needs some function accomplished, and the objective is to prove its correctness and security while treating the function as a black box. Then the function is later expanded into a subprotocol, which is verified separately. One thread of research that has taken this approach with sufficient formality and attention to security in a Dolev-Yao modeling context is represented by Datta, et al. [6] Their composition methodology, however, requires one to identify invariants used in the proofs of different protocol components, and check that each component satisfies the other's invariant as well as its own. This approach does not, in this general form, facilitate our objective for independent design and verification of services and the protocols using them.

Protocols are also combined, in a sense, when they are run independently in the same attacker environment, either concurrently or sequentially. Many papers have noted that it is possible for protocols to interfere with one another, invalidating security properties proved in isolation, when the same principal (using the same long-term private keys) participates in more than one protocol. Papers on this topic, with varying degrees of formality, go back at least to [11] and include [4] and [9]. Universal composability [3] is a strong property in the computational arena that can also lead, for some protocols, to preservation of security in a multiprotocol environment.

How could we use independent composition to verify services? The idea is to treat the caller request as a pair of guarantees, rather than as a guarantee for the precondition and a rely for the postcondition. It can then be verified separately. The service is a separate protocol anyway, its only dependence to the caller being the choice of input values. However, some additional conditions

will be necessary to prevent interference that would undermine security. Those conditions can be reflected in the choice of NDA and its semantics.

To give an example of a particular result along these lines, we apply the protocol independence theorem from [9]. The definitions are reviewed here, though [9] should be consulted for more details.

4.5 Review of Protocol Independence

The results concern a strand space Σ consisting of regular strands Σ_1 of a *primary* protocol (think of this as the caller), regular strands of another *secondary* protocol (from services), and penetrator strands. The following definitions are needed.

Definition 6. A strand space Σ is *full* if every atomic value a that originates on any secondary strand in Σ also originates on a penetrator strand in Σ.

An atom a is *private* if it originates uniquely only in Σ_1.

A term t *occurs in* a term t', written $t \sqsubseteq t'$, if it is a subterm, but not in the key part of an encryption. A term occurs in a node if it occurs in its message. A *component* of a non-pair term is the whole term, and a pair t, t' has components t and t'. A "new component" of a node is a term that has not occurred as a subterm of a prior node (by $\Rightarrow$) in the same strand.

Σ has *disjoint outbound encryption* if and only if the following holds. Let $n_1 \in \Sigma_1$ be a positive node, let $n_2 \in \Sigma_2$ be a negative node, and let a be a private atom occurring in some encrypted term $\{h\}_K$, where $\{h\}_K \sqsubset n_1$ and $\{h\}_K \sqsubset n_2$. Then there is no positive n_2' such that $n_2 \Rightarrow^+ n_2'$ and a occurs in a new component of n_2'.

Σ has *disjoint inbound encryption* if, for any negative node $n_1 \in \Sigma_1$ and positive node $n_2 \in \Sigma_2$ in which an encrypted term $\{h\}_k$ occurs, $\{h\}_k$ does not occur in any new component of n_2.

Two bundles over Σ are *equivalent* if they have the same Σ_1 nodes. Σ_1 is *independent* of Σ_2 if every bundle over Σ is equivalent to one with no Σ_2 nodes.

Theorem 2 (Protocol Independence, Prop. 7.2 in [9]). *If Σ is full and has both disjoint inbound and disjoint outbound encryption, then Σ_1 is independent of Σ_2.*

The significance of protocol independence is that any attack on an authentication property that is possible in the Σ multiprotocol strand space can, by equivalence, be reproduced without Σ_2 strands, and hence should have been excluded by a verification of the primary protocol in isolation.

4.6 Application of Protocol Independence

The idea behind disjoint inbound and outbound encryption is that a secondary protocol should not alter or repackage any encrypted term that either originated in the primary protocol or could be received by it. It was observed in [9] (and

the idea was also suggested in other papers) that a protocol identifier could be included in every encrypted term that was produced by a particular protocol, so that a party that received a term encrypted by a confidential key would be able to check that it was produced in its own protocol. All roles of the same protocol would use the same identifier, but a different (secondary or service) protocol would use a different identifier. Use of protocol identifiers in this way would guarantee disjoint inbound and outbound encryption.

To apply the protocol independence theorem, we also need the "full" property. The "full" condition requires that a confidential atom originating in the primary protocol may not also originate in the secondary protocol. This is a problem when the atom is passed as an input parameter to a service, since parameters that are not received in a message appear to originate in the service. However, we can use an NDA to fix this.

If a parameter a is uniquely originating in the calling protocol, and it is provided as an input to a service, we can list the encrypted terms in which it occurs. Let $\nu(a)$ be a predicate that tests membership in that list. A service with a compatible NDA for a parameter mapped to a recognizes a smaller list of terms that will be mapped to the same ground terms. The semantics of the NDA for service is the same: it specifies the set of "safe" terms in which a confidential parameter may occur. This means that the confidential parameter will be transmitted only in terms that can be traced back to terms originating in the caller. The full property is then satisfied for those parameters.

The "full" condition also prevents a confidential value from being generated by the service, since a confidential return parameter cannot be recognized as "private". This limits the applicability of the protocol independence result.

Note that, if we use protocol identifiers, use NDAs to limit private atom contexts, and refrain from generating confidential values in services to be returned to the caller, the conditions for protocol independence apply between different services as well as between the caller and its services.

To summarize:

Corollary 2. *Under the following conditions, a calling protocol and its services preserve authentication properties when verified independently:*

1. *each protocol (and service) includes a different protocol identifier in encrypted terms constructed by that protocol;*
2. *each service respects its promised NDA, in that each parameter x may occur only in components of sent messages satisfying $\nu(x)$;*
3. *no service originates a confidential atom returned to the caller.*

5 Conclusion

Call by contract is a way to specify and use interchangeable services in secure protocols, so that protocols and services can be independently designed and verified. The interface to a service is specified with a precondition and postcondition as in an Eiffel contract. However, to facilitate independent design, the calling

protocol requests a service with its own precondition and postcondition. The calling protocol need not know the name of the service or its parameter list.

A selection algorithm is given to test whether a candidate service is uniformly selectable. Uniform selection implies the existence of parameter mappings between the caller and service such that the preconditions and postconditions are sufficient, independently of the particular input parameter values. The selection algorithm is based on a technique called abstraction binding, employing a Datalog engine.

To facilitate independent security verification of the calling protocol and its services, contracts and requests also provide an NDA. Informally, NDAs are confidentiality constraints on parameters. Formally, NDAs of caller and service are compared in a straightforward way for "secure" selectability. The semantics of NDAs, and any proof that secure selectability enables independent verification, are not fixed, and can be interpreted differently in different protocol modeling contexts. We have given an example of NDA semantics that yields some limited independence, but stronger results, and different interpretations and results in other contexts, should be possible in the future.

We have an experimental prototype implementation of the service algorithm to support CPPL, our protocol compiler that makes use of a Datalog runtime engine. This prototype is still under development.

References

1. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *IEEE Symposium on Security and Privacy*, pages 171–182. IEEE Computer Society, 2005.
2. M. Bartoletti, P. Degano, and G-L. Ferrari. Enforcing secure service composition. In *18th IEEE Computer Security Foundations Workshop*, pages 211–223. IEEE Computer Society, 2005.
3. R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proc. IEEE FOCS*, pages 136–145, 2001.
4. R. Canetti, C. Meadows, and P. Syverson. Evironmental requirements for authentication protocols. In *Symposium on Requirements Engineering for Information Security*, March 2001.
5. Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions of Knowledge and Data Engineering*, 1(1), 1989.
6. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security. *Journal of Computer Security*, 2005.
7. Trusted Computing Group. Trusted platform module main specification, version 1.2, 2006.
8. Joshua Guttman, Jonathan Herzog, John Ramsdell, and Brian Sniffen. Programming cryptographic protocols. In *Symposium on Trusted Global Computing*, volume 3705 of *Lecture Notes in Computer Science*. Springer, April 2005.
9. Joshua Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Computer Security Foundations Workshop*. IEEE Computer Society, 2000.

10. Joshua Guttman, F. Javier Thayer, Jay Carlson, Jonathan Herzog, John Ramsdell, and Brian Sniffen. Trust management in strand spaces: A rely-guarantee method. In *European Symposium on Programming (ESOP)*, 2004.
11. N. Heintze and J.D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22(1):16–30, 1996.
12. Ninghui Li, Joan Feigenbaum, and Benjamin Grosof. A logic-based knowledge representation for authorization with delegation. In *12th Computer Security Foundations Workshop*, pages 162–174. IEEE Computer Society, 1999.
13. F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

A Strand Spaces: A Brief Summary

The purpose of this summary is primarily to review the vocabulary used in this paper. It should be read to understand the latter part of Section 4.

A *strand* is a sequence of nodes. A node has a label $(\pm m, \phi)$ where $\pm m$ is a directed (+ for sent or − for received) message, and the annotation ϕ is a formula. The atoms used in m and ϕ are elements of the disjoint union $A = X \cup Z$ of *parameters* X and *values* Z. The *trace* of a strand is its sequence of node labels. Any sequence of node labels may be referred to as a trace. When applied to a trace, the term "node" refers to a position in the label sequence.

A *bundle* is a finite partially ordered set of nodes, where each node belongs to a partial strand (an initial subsequence of a strand) and the partial ordering combines the strand node sequence ordering (represented with $\Rightarrow$) and a communication ordering $\rightarrow$ that relates each receive node to a prior send node with the same message. All atoms occurring in a bundle must be from Z, though some authors make use of *semibundles* in which parameters may occur, and some receive nodes do not have predecessors in the communication ordering.

A parameter is *bound* at a node if it occurs in the node label of that node, or of a prior node, or (for any node) if it is an input.

A protocol is a set of roles. A *role* is a 4-tuple $r = (R, I, U, T)$ where R is a trace in which all atoms are parameters, I is a subset of parameters designated as inputs, U is a subset of parameters designated as *uniquely originating*, and T is the *initial theory*. T consists of general inference rules (see Section 2) and facts expressible as formulas over input parameters.

There are *penetrator* roles that define the attacker model. The penetrator roles do not belong to any particular protocol. A penetrator role is one of a few specific types representing the ability of the attacker to construct and decompose messages. For example, $(-k, \top), (-\{x\}k, \top), (+x, \top)$ is a penetrator role that performs decryption, where $\top$ is the constant *true* formula.

A strand is a *ground* strand if its atoms (the atoms occurring in its node labels) are all values. A *(strand) substitution* is an idempotent map of a subset of X into A. Strand substitutions are extended to messages, formulas, node labels, and traces in the usual way. A strand is an *instance* of a role r under substitution σ if its trace is $r\sigma$.

It is a *ground instance* if the range of σ is included in Z. A *regular* strand is an instance of a protocol role. A *penetrator* strand is an instance of a penetrator role.

A bundle is *consistent* with a protocol with roles $\{r_i\}$ if its nodes are the disjoint union of nodesets of ground partial strands, such that each strand is either a regular strand of some role r_i or a penetrator strand. Furthermore, we require that (1) each positive node formula is provable from the (mapped) initial theory together with the formulas on prior negative nodes of the strand, and (2) the bundle respects unique origination.

Any value of a uniquely originating parameter must be uniquely originating in the bundle in the sense of [13]. A bundle can be consistent with a protocol without necessarily satisfying security properties expressed as rely formulas. Protocol verification establishes correctness of rely formulas in any bundle consistent with the protocol.

B Implementation Notes

Notes on our implementation of the selector algorithm follow. The implementation follows the algorithm in Section 3 except that it combines the two abstraction-binding steps into a single Datalog query. This is facilitated by adding rules expressing the contract.

Let $S(\phi)$ be a sequence of the parameters in formula ϕ. Each parameter in $S(\phi)$ occurs once, and the parameters are ordered by their first appearance in the formula. Let $V(\boldsymbol{X})$ be a sequence of distinct variables of the same length as $\boldsymbol{X}$. ($V(\boldsymbol{X})$ can be the $\boldsymbol{X}$-length initial subsequence of a long fixed variable sequence.) Let $r :: \boldsymbol{X}$ be the literal constructed from predicate symbol r and the sequence of terms $\boldsymbol{X}$. Let $p[\boldsymbol{X}]$ be the formula in which each parameter in $\boldsymbol{X}$ is replaced in p by its corresponding variable in $V(\boldsymbol{X})$. Thus, $p[\boldsymbol{X}]$ is an abstraction of p over a specific set of parameters. The concatenation of sequences $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ is $\boldsymbol{X}_1 \cdot \boldsymbol{X}_2$. The subsequence of $\boldsymbol{X}_1$ elements that are not in $\boldsymbol{X}_2$ is $\boldsymbol{X}_1 \setminus \boldsymbol{X}_2$.

Note that, in general, a substitution σ on a set X can be represented with respect to an ordering $\boldsymbol{X}$ by a result vector $\boldsymbol{Y}$ such that $\sigma(x) = \boldsymbol{Y}(\boldsymbol{X}^{-1}(x))$. We write $\sigma = \boldsymbol{Y}/\boldsymbol{X}$. If substitutions $\sigma_i = \boldsymbol{Y}_i/\boldsymbol{X}_i$ have disjoint domains, $\sigma_1 \cup \sigma_2 = \boldsymbol{Y}_1 \cdot \boldsymbol{Y}_2/\boldsymbol{X}_1 \cdot \boldsymbol{X}_2$. Also, $(\sigma \circ \tau)(x) = \sigma(\tau(x))$ and $(\sigma \circ \boldsymbol{X})(i) = \sigma(\boldsymbol{X}(i))$.

A contract in the implementation is a 5-tuple $(p_s, q_s, \boldsymbol{I}_s, \boldsymbol{O}_s, s)$, where s is the name of the procedure that implements the service. The service contract of s is (p_s, q_s), the input parameter sequence of s is $\boldsymbol{I}_s$, and the output parameter sequence of s is $\boldsymbol{O}_s$.

A caller provides a triple $(p_c, q_c, \boldsymbol{Z}_c)$, where the caller's pre- and postcondition are (p_c, q_c), and the values associated with the parameters in $S(p_c)$ are given by $\boldsymbol{Z}_c$. Thus, $\sigma_e = \boldsymbol{Z}_c/\boldsymbol{I}_c$ for $\boldsymbol{I}_c = S(p_c)$. A call is ill-formed if the length of $\boldsymbol{Z}_c$ differs from the length of $\boldsymbol{I}_c$. Let $\boldsymbol{O}_c = S(q_c) \setminus S(p_c)$.

If the selector algorithm determines that a service s is selectable, and then selects it, it invokes the service with a sequence of input values $\boldsymbol{Z}_s$ and a sequence

of natural numbers $\boldsymbol{N}_s$. The sequence $\boldsymbol{N}_s$ is defined below; it tells the service which values to return, and in which order, corresponding to caller outputs.

The selector algorithm is the following.

1. Rename parameters in the service contract to ensure that service parameters do not occur in caller's formulas.
2. Assert each literal in p_c.
3. Recall that a formula is a conjunction of literals, and let ϕ^i be the i-th literal. For each literal in $q_s[\boldsymbol{I}_s]$, assert $q_s^i[\boldsymbol{I}_s] \,\text{:-}\, p_s[\boldsymbol{I}_s]$. If any clause is not safe, the service s is not selectable. These rules represent the contract.
4. Let r be a fresh predicate symbol. Assert $r{::}V(\boldsymbol{I}_s{\cdot}\boldsymbol{O}_c)\,\text{:-}\,p_s[\boldsymbol{I}_s{\cdot}\boldsymbol{O}_c], q_c[\boldsymbol{I}_s{\cdot}\boldsymbol{O}_c]$. If the clause is not safe, the service s is not selectable.
5. If an instance of $r :: V(\boldsymbol{I}_s \cdot \boldsymbol{O}_c)$ is derivable, the service s is selectable.

Let $r :: \underline{\boldsymbol{I}}_s \cdot \underline{\boldsymbol{O}}_c$ be a derived instance of $r :: V(\boldsymbol{I}_s \cdot \boldsymbol{O}_c)$. The input values $\boldsymbol{Z}_s$ required by the service are obtained as $(\boldsymbol{Z}_c/\boldsymbol{I}_c) \circ (\underline{\boldsymbol{I}}_s/\boldsymbol{I}_s) \circ \boldsymbol{I}_s$. The number sequence $\boldsymbol{N}_s$ given to the service is obtained as $(\underline{\boldsymbol{I}}_s \cdot \boldsymbol{O}_s)^{-1} \circ \underline{\boldsymbol{O}}_c$.

The service then executes and produces its own output values $\boldsymbol{Z}_o$. It then produces the sequence of values for caller outputs as $\boldsymbol{Z}_s \cdot \boldsymbol{Z}_o \circ \boldsymbol{N}_s$.