Fine-Grain Configurability for Secure Communication

Matti A. Hiltunen Sumita Jaiprakash Richard D. Schlichting Carlos A. Ugarte

TR 00-05

Exploiting Fine-Grain Configurability for Secure Communication[1]

Matti A. Hiltunen, Sumita Jaiprakash, Richard D. Schlichting, and Carlos A. Ugarte

TR 00-05

Abstract

Current solutions for providing communication security in network applications allow customization of certain security attributes and techniques, but in limited ways and without the benefit of a single unifying framework. Here, the design of a highly-customizable extensible service called SecComm is described in which attributes such as authenticity, privacy, integrity, and non-repudiation can be customized in arbitrary ways. With SecComm, applications can open secure communication connections in which only those attributes selected from among a wide range of possibilities are enforced, and are enforced using the strength or technique desired. SecComm has been implemented using Cactus, a system for building configurable communication services. In Cactus, different properties and techniques are implemented as software modules called micro-protocols that interact using an event-driven execution paradigm. This non-hierarchical design approach has a high degree of flexibility, yet provides enough structure and control that it is easy to build collections of micro-protocols realizing a large number of diverse properties. This paper gives an overview of the design and implementation of SecComm, and gives initial performance figures for a prototype implementation running on a cluster of Pentiums using the Mach MK 7.3 operating system.

June 16, 2000

Department of Computer Science
The University of Arizona
Tucson, AZ 85721

[1]This work supported in part by the Office of Naval Research under grant N00014-96-0207, the Defense Advanced Research Projects Agency under grant N66001-97-C-8518, and the National Science Foundation under grants CDA-9500991 and ANI-9979438.

1 Introduction

The ability to ensure security attributes such as confidentiality, integrity, and authenticity is an increasingly important requirement for systems that implement network communication. While a common need for many applications, the best security solution for a given situation can vary widely depending on the potential impact of a security violation. For example, the theft of a credit card number might result in the loss of a few thousand dollars, while the theft of critical military information might result in the loss of human life. The tradeoff, of course, is that the cost of providing security—in terms of execution time or some other metric such as network throughput—increases as the guarantees are strengthened. To support the diverse needs of varying applications, then, secure communication services should allow the tradeoff between the level and the cost of the guarantee to be explicitly managed.

The value of customizing communication security in this way has been recognized in recent Internet security protocols such as IPSec [KA98], SSL [FKK96], S-HTTP [RS98], and TLS [DA99]. For example, IPSec, a set of protocols developed by the IETF to support secure packet exchange at the IP layer, provides two security options. The *authentication header* (AH) option does not encrypt the data contents of the packet, but provides optional authenticity, integrity, and replay prevention by adding an AH that contains a cryptographic message digest. The *encapsulating security payload* (ESP) option provides privacy by encrypting the data contents of the packet and optional authenticity, integrity, and replay prevention using a message digest. While such facilities are useful, the degree of customization is usually limited, either to ensure interoperability or to improve performance. Thus, existing services typically do not support options such as encrypting a message with multiple encryption methods, alternating encryption methods, or other methods that result in data expansion (e.g., steganography [JJ98]).

We argue in this paper that it should be possible to customize all security attributes related to communication and to customize them in essentially arbitrary ways. Such customization allows explicit control over the tradeoffs between security and performance for a given attribute, not only by providing multiple algorithmic choices, but also by allowing combinations of algorithms to be used. A communication service supporting the fine-grain and dynamic application of security has other potential benefits as well. For example, security algorithms can be used on a per-message rather than a per-connection basis, *artificial diversity* [CP97] can be realized by customizing different instances of a service in different ways, and services can adapt dynamically to react to changed security requirements or to a change in perceived threats.

To support this argument, we present the design of a highly customizable and extensible secure communication service called SecComm. With SecComm, applications can open secure communication connections in which the security attributes and the strength of guarantees associated with each attribute can be customized at a fine-grain level. In addition, SecComm allows an attribute to be guaranteed using combinations of security algorithms, and it supports extensibility by allowing the addition of new algorithms as separate modules. At another level, our approach can also be viewed as a technique for implementing protocols such as IPSec, SSL, S-HTTP, and TLS in a modular and extensible fashion.

The customization and extensibility attributes of SecComm derive from the use of Cactus as the underlying implementation platform [HSH+99]. Cactus is a framework for constructing highly-configurable network services, where each service attribute or variant is implemented as an independent software module called a *micro-protocol*. A customized version of the service is then constructed by choosing micro-protocols based on the desired properties. In the Cactus model, micro-protocols within a service are composed non-hierarchically and interact primarily using a flexible event mechanism. This indirect approach to communication promotes the independence of micro-protocols, yet is flexible enough that even complex

interactions are possible. While other researchers have also proposed systems that allow for modular composition of security properties [OOSS94, RBH+98, NK98], none of these approaches has the advantages of fine-grain composability and non-hierarchical composition available in SecComm. In short, our work pushes the envelope on configurability and flexibility much further than any other work in this area.

This paper has several goals. The first is to argue that fine-grain configurability and extensibility are valuable characteristics for realizing security attributes in future communication services. The second is to demonstrate that the Cactus approach provides a good platform for constructing configurable secure communication services in an efficient and secure manner. The third is to present the implementation of SecComm service based on Cactus.

2 Configurable Communication Security

2.1 Overview

Our system model consists of a collection of machines connected by a local- or wide-area communication network. Application level processes communicate by using a communication subsystem that typically consists of IP, some transport level protocol such as TCP or UDP, and potentially some middleware level protocols such as IIOP [OMG98]. The SecComm protocol can be inserted in any layer above IP in the communication subsystem as illustrated in figure 1. (The internal structure of SecComm is explained further in section 3.) SecComm is generally independent of the choice of the lower level communication protocol, but the guarantees provided by the lower level may affect the set of viable micro-protocols. For example, some security micro-protocols require that the underlying protocol provides reliable ordered delivery, which constrains the use of these particular micro-protocols to the case where SecComm is used on top of TCP or some other transport protocol with similar guarantees.

The method used to insert SecComm into the communication subsystem depends on the particular implementation platform. Systems such as the *x*-kernel [HP91], CORDS [TMR96], and Scout [MMO+95] allow explicit construction of protocol graphs. In such systems, SecComm is simply inserted into the protocol graph before compilation. On other systems, SecComm is either inserted into the existing kernel communication subsystem using methods such as loadable modules, or is built on top of TCP or UDP sockets in user space. The method of integration does not affect the internal design of SecComm.

A secure communication connection is established by opening a *session* through the SecComm service. Each session has two sets of customized security attributes that are specified at open time, one for messages traversing the session from the application to the network and the other for messages traversing the session in the opposite direction. This feature allows, for instance, the security guarantees for request messages from a client to a server to be different than those for the reply messages. SecComm is also independent of the communication paradigm used by the application, i.e., it can be used for symmetric group communication as well as for asymmetric client/server interactions. Finally, the SecComm service does not impose a single form of key management on applications. The keys can be established on a higher level and simply passed to the SecComm session or, alternatively, the SecComm service can establish the keys itself. If the latter is chosen, options include protocols such as Diffie-Hellman [DH76] or the use of external Certification Authorities (CA) and Key Distribution Centers (KDC) such as Kerberos [SNS88, NT94].

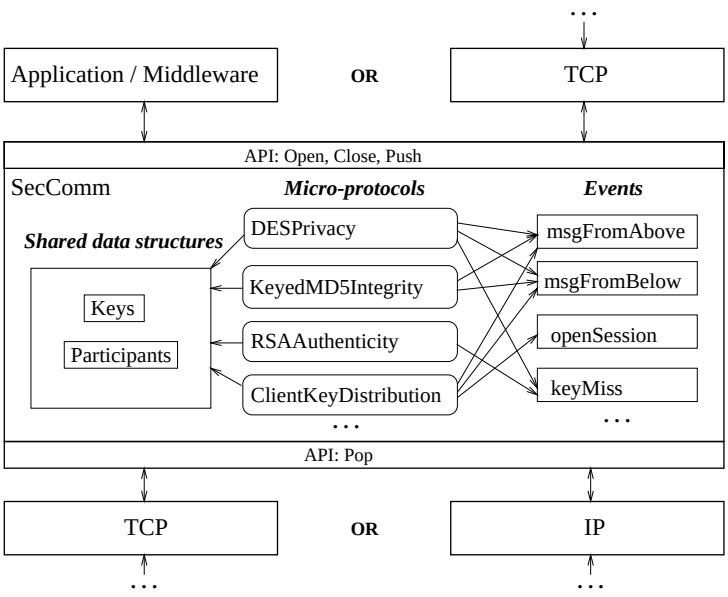

Figure 1: System protocol stack.

2.2 Security properties

As a first step towards exploiting fine-grain configurability, security properties and their variants are identified. Well-known abstract security properties include:

- **Authenticity**. Ensures that a receiver can be certain of the identity of the message sender. Can be implemented using public key cryptography [Hel78], any shared secret, or a trusted intermediary such as Kerberos [SNS88, NT94].
- **Privacy**. Ensures that only the intended receiver of a message is able to interpret the contents. Can be implemented using any shared secret, public key cryptography, or combinations of methods.
- **Integrity**. Ensures that the receiver of a message can be certain that the message contents have not been modified during transit. Some authenticity and privacy methods also provide integrity as a side effect if the message format has enough redundancy to detect violations. Additional redundancy can be provided using message digest algorithms such as MD5 [Riv92]. Integrity can be provided without privacy, but at a minimum, the message digest itself must be protected.
- **Non-repudiation**. Ensures that a receiver can be assured that the sender cannot later deny having sent the message. Relies on authenticity provided by public key cryptography and requires that the receiver store the encrypted message as proof.

We can identify other security properties that are focused on prevention of specific security attacks. These properties include:

- **Replay prevention**. Prevents an intruder from gaining an advantage by retransmitting old messages. Can be implemented using timestamps, sequence numbers, or other such nonces in messages. Typically used in conjunction with authenticity, privacy, or integrity since otherwise it would be trivial for an intruder to generate a new message that appears to be valid.

- **Known plain text attack prevention**. Prevents an intruder from utilizing known plain text based attacks by including additional random information ("salt") at the beginning of a message.

In addition to these security properties, key management is completely customizable.

2.3 Cactus implementation platform

A service in Cactus is implemented as a *composite protocol*, with each semantic variant of a security attribute or other functional component within the composite protocol implemented as a *micro-protocol* (figure 1). A micro-protocol is, in turn, structured as a collection of *event handlers*, which are procedure-like segments of code that are executed when a specified *event* occurs. Events are used to signify state changes of interest, such as "message arrival from the network". When such an event occurs, all event handlers bound to that event are executed. Events can be raised explicitly by micro-protocols or by the composite protocol.

The primary event-handling operations are:

- **bind**(*event, handler, order, static_args*). Specifies that *handler* is to be executed when *event* occurs. *order* is a numeric value specifying the relative order in which *handler* should be executed relative to other handlers bound to the same event. When the handler is executed, the arguments *static_args* are passed as part of the handler arguments.

- **raise**(*event, dynamic_args, mode, delay*). Causes *event* to be raised after *delay* time units. If *delay* is 0, the event is raised immediately. The occurrence of an event causes handlers bound to the event to be executed with *dynamic_args* (and *static_args* passed in the **bind** operation) as arguments. Execution can either block the invoker until the handlers have completed execution (*mode* = SYNC) or allow the caller to continue (*mode* = ASYNC).

Other operations are available for unbinding handlers from events, creating and deleting events, halting event execution, and canceling a delayed event. Execution of handlers is atomic with respect to concurrency, that is, a handler is executed to completion before execution of any other handler is started unless the handler voluntarily yields execution by either raising another event synchronously or by invoking a blocking semaphore operation. In the case of a synchronous raise, the handlers bound to the raised event are executed before control returns to the handler that invoked the raise operation. In addition to the flexible event mechanism, Cactus supports shared data that can be accessed by all micro-protocols configured into a composite protocol.

Finally, the system supports a Cactus message abstraction designed to facilitate development of configurable services. The main features provided by Cactus messages are named message attributes and a coordination mechanism that only allows a message be sent out of the composite protocol when agreed by all micro-protocols. The message attributes are a generalization of traditional message headers and they have scopes corresponding to a single composite protocol (*local*), all the protocols on a single machine (*stack*), and the peer protocols at the sender and receiver (*peer*). A customizable pack routine concatenates peer attributes to the message body for network transmission, or for operations such as encryption and compression. A corresponding unpack routine extracts the peer attributes from a message at the receiver.

Several prototype implementations of Cactus have been constructed, including one written in C that runs on Mach version MK 7.3 from OpenGroup [Rey95], another written in C++ that runs on Solaris and Linux, and a third written in Java that runs on multiple platforms. An initial prototype of SecComm has been implemented on the MK version of Cactus on a cluster of Pentiums. Other prototype services that have

been successfully implemented using Cactus or the predecessor Coyote system [BHSC98] include group RPC [HS95], membership [HS98], and a real-time channel abstraction [HSH+99].

3 SecComm Design

3.1 Application programming interface

The SecComm service allows a higher level service or application to open secure connections and then send and receive messages through these connections. The specific operations exported by SecComm are the following:

- **Open**(participants,role,properties). Opens a session for a new communication connection, where *participants* is an array identifying the communicating principals, *role* identifies the role of this participant in opening the connection (active or passive), and *properties* is a specification of the desired security properties of the session.
- **Push**(msg). Passes a message from a higher level protocol or application to a SecComm session to be transmitted with the appropriate security attributes to the participants.
- **Pop**(msg). Passes a message from a lower level protocol to a SecComm session to be decrypted, checked, and potentially delivered to a higher level protocol. When the SecComm protocol passes a message to the higher level and authentication is required, it adds a stack attribute AUTH_SENDER that is the ID of the authenticated sender.
- **Close()**: Closes a SecComm communication session.

We assume that the participants of the communication connection negotiate the properties for the connection on a higher level. Once negotiated, properties are specified in the open operation as two ordered lists of micro-protocols and their arguments, the first for messages going downward through the composite protocol and the second for messages going upward. Thus, for example, the following specifies that messages going downward are processed first by DESPrivacy and then by RSAAuthority, while messages going upwards are processed by the same micro-protocols but in the reverse order:

{DESPrivacy(DESkey), RSAAuthenticity(RSAkey); RSAAuthenticity(RSAkey), DESPrivacy(DESkey)}

This relatively low level approach to specifying properties is an interim strategy. Our eventual goal is to develop an approach in which properties are given as formal specifications that are then translated automatically into collections of micro-protocols and arguments.

3.2 Shared data structures and events

The main use of shared data in SecComm is to store keys. In particular, each SecComm session contains a shared table *Keys* that stores all the keys currently used in this session. This table is initialized using the predefined keys passed in the Open operation, with other keys potentially added during execution by key distribution micro-protocols.

Our prototype implementation of SecComm uses the cryptographic package Cryptlib [Gut98] to provide basic cryptographic functionality. Any cryptolibrary with the necessary functions could be used, however.

The design of SecComm uses a number of events for communication between micro-protocols and to initiate execution when messages arrive. The SecComm composite protocol uses the following events to indicate message arrivals from above and below:

- msgFromAbove(msg). Indicates that *msg* has arrived from a higher level protocol or application.
- dataMsgFromBelow(msg). Indicates that a data *msg* has arrived from a lower level protocol or OS.
- keyMsgFromBelow(msg). Indicates that a *msg* associated with key distribution has arrived from a lower level protocol or OS.

These events are raised within the push and pop operations provided as part of the composite protocol's runtime system. The pop operation has been customized using facilities provided by the Cactus framework to distinguish between data and key distribution messages.

Other events are used for communication between the micro-protocols that secure data communication and those that implement key distribution and security monitoring:

- keyMiss(index,length,check). Indicates that the key *index* in the Keys table is required. The key should be of size *length* and satisfy validity test *check*. The validity test can be used to eliminate weak keys.
- securityAlert(type,msg). Indicates that a potential security violation of *type* related to *msg* has been detected.

3.3 Micro-protocol structure

The abstract security attributes described in section 2.2, as well as key distribution, are implemented by one or more micro-protocols. When a number of micro-protocols implement variations of the same abstract property, we collectively refer to them as a *class* of micro-protocols. For example, the class of privacy micro-protocols includes DESPrivacy, RSAPrivacy, and IDEAPrivacy micro-protocols that use DES, RSA, and IDEA algorithms, respectively. Figure 2 illustrates the main micro-protocol classes and typical event interactions between them.

The design of the SecComm service allows any combination of security micro-protocols to be used together in both static and dynamic ways. Naturally, there may be some configuration constraints between micro-protocols that restrict which combinations are feasible. These constraints and other composability issues are discussed in section 4.3.

The SecComm service consists of two major types of micro-protocols: basic security micro-protocols that perform simple security transformations such as encryption or integrity checks, and meta-security micro-protocols that build more complex security protocols using the basic security micro-protocols as building blocks. An example of a simple security micro-protocol would be DESPrivacy, which provides privacy of data exchange using the DES algorithm. An example of a meta-security protocol would be MultiSecurity, which uses multiple basic security micro-protocols to provide stronger guarantees. Each type is now described in turn.

3.4 Basic security micro-protocols

The basic security micro-protocols are simple, as illustrated in figure 3. These micro-protocols typically consist of two event handlers and an initialization section. One of the event handlers is used for the data

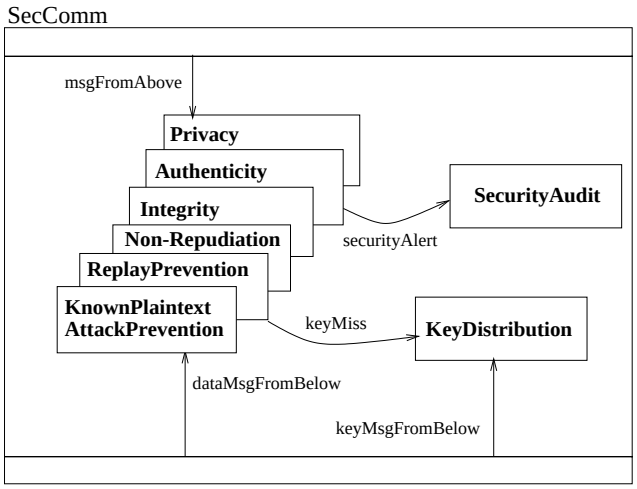

Figure 2: Micro-protocol classes and their interactions.

passing down through the SecComm protocol and the other one is used for data passing up through the protocol. The initialization section of the micro-protocol is executed when a new SecComm connection is opened, i.e., when a session is created.

A basic security micro-protocol typically takes 4 or 5 arguments. In this parameter list, *downEvent* and *upEvent* are events that signify message arrival from an upper- and lower-level protocol, respectively. The two handlers in the micro-protocol are bound to these events to initiate execution at the appropriate time. The *dorder* and *uorder* parameters are the relative orders in which this particular security micro-protocol is to be applied to messages flowing down and up, respectively. Finally, the *key* is an index in the Keys data structure. The *key* argument is omitted from basic security micro-protocols that do not use keys, such as replay prevention.

Note that if the key used by the security micro-protocol has yet not been established, it raises the event `keyMiss` that is handled by the key distribution micro-protocols (see section 3.6). This event is raised synchronously and thus, the handler is blocked until the associated event handlers have completed execution. This allows the key distribution micro-protocols to block the appropriate handler until the key has been established.

The design uses event pointers as arguments rather than fixed event names to allow multiple types of configurations, an approach that demonstrates the inherent flexibility provided by an event-based execution model. As the most simple case, assume that a SecComm configuration uses only basic security micro-protocols. The configuration can then be initialized to use the *msgFromAbove* and *dataMsgFromBelow* events directly as follows:

```
BasicSecurity(msgFromAbove,1,dataMsgFromBelow,1,0)
```

The same mechanism can also be used to establish different security guarantees for different communication directions, such as might be used in client/server communication. For example, to provide DES-based privacy only for communication from server to client, the following can be used, where *nullEvents* are events that are never raised:

```
micro-protocol BasicSecurity(downEvent,dorder,upEvent,uorder,key) {

    handler ProcessDownMsg(msg){
        if Keys[myKey] == NULL raise(keyMiss,myKey,SYNC);
        add attributes, pack, encrypt, etc.;
    }
    handler ProcessUpMsg(msg){
        if Keys[myKey] == NULL raise(keyMiss,myKey,SYNC);
        decrypt, unpack, check attributes, etc.;
    }
    initial { myKey = key;
        bind(downEvent,ProcessDownMsg,dorder); bind(upEvent,ProcessUpMsg,uorder);
    }
}
```

Figure 3: Generic basic security micro-protocol.

At client:

```
nullEvent = new Event();
DESPrivacy(nullEvent,1,dataMsgFromBelow,1,0);
```

At server:

```
nullEvent = new Event();
DESPrivacy(msgFromAbove,1,nullEvent,1,0);
```

As a more complex example, the same mechanism can be used to easily create a variant of triple DES, as follows:

```
DESPrivacy(msgFromAbove,1,dataMsgFromBelow,1,0);
DESPrivacy(dataMsgFromBelow,2,msgFromAbove,2,1);
DESPrivacy(msgFromAbove,3,dataMsgFromBelow,3,0);
```

This variant exploits the fact that our DESPrivacy micro-protocol encrypts messages associated with *downEvent* and decrypts messages associated with *upEvent* to realize the appropriate triple DES semantics. Note, however, that this variation is not identical to standard 3DES since the whole message is encrypted completely by one method at a time, whereas 3DES encrypts each block of a message with each of the three encryption methods before the next block is processed. This problem could be eliminated if each of the micro-protocols received a block of data at a time rather than the whole message, but varying block sizes required by different encryption algorithms would make this solution more complicated. In our simple design, the standard 3DES could be implemented as one micro-protocol.

The above example illustrates the importance of taking the relative execution order of micro-protocols into account for handling events *msgFromAbove* and *dataMsgFromBelow*. Specifically, the order in which the cryptographic micro-protocols are applied at the receiver must be the reverse of the order used at the sender since cryptographic methods are typically not commutative. The need to realize ordering constraints between micro-protocols or micro-protocol classes is common in other situations as well. For example, replay prevention must be executed before data integrity so that any nonces added will be protected from modification by the message digest. Other often subtle constraints–e.g., that message integrity should be done before privacy—are identified in [AN96, AN95].

```
micro-protocol MetaSecurity(downEvent,dorder,upEvent,uorder, downBasicEvents,upBasicEvents) {

    handler ProcessDownMsg(msg){
        in some order raise(downBasicEvents[i],msg,SYNC);
    }
    handler ProcessUpMsg(msg){
        in some order raise(upBasicEvents[i],msg,SYNC);
    }
    initial {
        bind(downEvent,ProcessDownMsg,dorder); bind(upEvent,ProcessUpMsg,uorder);
    }
}
```

Figure 4: Generic meta-security micro-protocol.

3.5 Meta-security micro-protocols

Meta-security micro-protocols construct more complex security protocols out of the basic security protocols. For example, a meta-security micro-protocol may apply multiple or alternating basic security micro-protocols to a message. The outline of a typical meta-security micro-protocol is presented in figure 4. In this design, the micro-protocol is passed vectors of down and up events that correspond to the events to which handlers in the basic micro-protocols have been bound as arguments *downBasicEvents* and *upBasicEvents*.

The concept of meta-security micro-protocols can be applied to strengthen any security property for which using multiple or alternating methods provides enhanced guarantees. Privacy, authenticity, and message integrity among others fall under this category. The SecComm design does not prevent the same idea from being used for other properties such as replay prevention and non-repudiation, but the benefit for such properties is more questionable. Finally, note that the ease with which such meta-security micro-protocols can be constructed is again a direct result of flexibility provided by the Cactus model and something that would be more difficult in systems that support only hierarchical composition.

3.6 Key distribution micro-protocols

If the keys used by the secret key cryptographic methods are not agreed upon *a priori*, they must be established after the communication session is opened. Among the potential options for key distribution are:

- **Asymmetric**. One communicating principal (e.g., a client or a server) creates a session key and distributes it to the other principals.
- **Symmetric**. A session key is created using the Diffie-Hellman algorithm.
- **External**. Some external security principal creates the session key and distributes it to communicating principals (e.g., Kerberos, certification authority).

Key distribution has security risks analogous to data communication, but with greater potential impact since the compromised key will likely be used for a period of time. Thus, the same techniques used for data security can also often be applied for key distribution security. In our design, key distribution micro-protocols are meta-security micro-protocols and thus, can easily utilize basic security micro-protocols when appropriate. Naturally, not all techniques developed for data security apply for key distribution, and in such cases, the key distribution micro-protocols must directly implement the necessary techniques. For example,

```
micro-protocol KeyDistribution(myKeys,downBasicEvents,upBasicEvents) {

    handler ProcessKeyMiss(index){
        if index ∈ myKeys {
            create a key distribution msg;
            msg.addAttr(KEY,index,PEER);
            process msg using downBasicEvents;
            send msg; P(semaphore); }
    }
    handler ProcessKeyMsg(msg){
        if msg.getAttr(KEY) ∈ myKeys {
            process key distribution msg using upBasicEvents;
            Keys[msg.getAttr(Key)] = new key; V(semaphore);}
    }
    initial { bind(keyMiss,ProcessKeyMiss); bind(keyMsgFromBelow,ProcessKeyMsg); }
}
```

Figure 5: Generic key distribution micro-protocol.

if Kerberos is used for key distribution, Kerberos-specific methods are used for privacy, authenticity, integrity, and replay prevention, while Kerberos message formats are used for the key distribution process. A KerberosKeyDistribution micro-protocol would directly implement the required techniques to interact with a Kerberos server.

Figure 5 outlines a generic key distribution micro-protocol. It takes as arguments an array of key indices (*myKeys*) that it needs to establish and set of event pointers used to secure key distribution messages. The array *myKeys* includes the identities of the principals when necessary. For example, if the key distribution micro-protocol must obtain the RSA public keys of security principals, it must know the identity of the principal whose key is required. Note that the event handler for the *keyMiss* event blocks on a semaphore to ensure that the security micro-protocol that raised the event will be blocked until the key is ready to be used. Note also the operations addAttr and getAttr, which insert and extract named message attributes, respectively.

4 Implementation and Performance

A prototype of SecComm has been implemented using the C version of Cactus on a cluster of 133 MHz Pentium PCs running OpenGroup/RI MK 7.3 and CORDS. As noted above, the internal structure of SecComm is largely independent of the specific version of Cactus used and the execution platform.

This section provides details on some of the micro-protocols available in SecComm, as well as initial performance numbers. The goal of the micro-protocol descriptions is to be representative rather than inclusive.

4.1 Micro-protocols

Privacy. SecComm includes numerous basic privacy micro-protocols, ranging from those based on standardized cryptographic methods such as DESPrivacy and RSAPrivacy, to others based on non-standard methods. The latter include OneTimePadPrivacy, which encrypts a stream of messages by xoring it with a secret file that is shared by the sender and receiver. Other simple privacy micro-protocols include XOR-

Privacy that xors each block of the data with a secret key. Such method does not provide a high level of privacy, but may be enough the deter a casual observer. Moreover, combinations of fast trivial methods used in conjunction with a standard method such as DESPrivacy may enhance privacy considerably by making it harder to use specialized equipment designed to break DES, for example.

Authenticity, integrity, and non-repudiation. Authenticity, integrity, and non-repudiation are discussed together since non-repudiation depends on authenticity and integrity can be considered a subset of authenticity. Authenticity and integrity can be achieved either by encrypting the entire message or by using a message digest generated by a cryptographic message digest or hash function such as MD5 [Riv92] or SHA [SHA95]. If a standard message digest function is used, the digest itself must be protected by either encrypting it or by calculating the digest over the data and a secret key (e.g., keyed MD5 [MS95a] and SHA [MS95b]).

SecComm includes two basic message digest micro-protocols, MD5Integrity and SHAIntegrity, and their keyed counterparts, KeyedMD5Integrity and KeyedSHAIntegrity. Each of these micro-protocols creates a message digest as a peer attribute with tag DIGEST at the sender, and checks it at the receiver. If the non-keyed integrity micro-protocols are used, they must be executed before the corresponding cryptographic protocol so that the message digest is protected.

Two authenticity micro-protocols based on public keys are also included. RSAAuthenticity encrypts the entire message with the sender's private key, while RSADigestAuthenticity encrypts only the message digest. Note that the integrity micro-protocols may be used with RSAAuthenticity. In this case, the entire message including the message digest will be encrypted for authenticity.

Unlike methods based on public keys, with shared secrets, one micro-protocol can be used to provide both privacy and authenticity by encrypting the entire message. Similar to RSADigestAuthenticity, however, we can develop variants of the privacy protocols that only encrypt the message digest. Since these micro-protocols do not provide privacy, we consider them exclusively authenticity micro-protocols. Thus, the DES based micro-protocol is called DESDigestAuthenticity.

Non-repudiation is based on public key authentication, with the receiver storing the message encrypted using the sender's private key as a proof of having received the message. The NonRepudiation micro-protocol simply stores the encrypted message in a file with a timestamp indicating when it was received. To make it easier to check the authenticity of the stored message at a later time, a SecComm session should be configured to use public key authentication as the first cryptographic method at the sender and thus, the last cryptographic method at the receiver. Thus, if NonRepudiation is configured to execute just before message authentication at the receiver, authenticity can be verified at some later time knowing only the sender's public key at the time the message was sent.

Attack prevention. Several SecComm micro-protocols address replay attacks and known plain text attacks. To prevent accidental or malicious message replay, TimeReplayPrevention adds a timestamp to the message at the sender, and then checks that value at the receiver to determine if it is too old. If the underlying communication protocol guarantees FIFO delivery, then the SeqReplayPrevention micro-protocol can be used. It attaches a sequence number to each message and then verifies that the number in a new message is larger than the largest sequence number seen so far. Both would typically be used in conjunction with integrity or privacy methods to prevent undetected modification of the message.

Known plain text attacks can be made more difficult by inserting a random sequence of data at the

beginning of each message. The KPTAPrevention micro-protocol does this by inserting a message attribute with a random contents to the message at the sender. The default packing routine of Cactus automatically packs the message attributes before the message body, but if a different order is desired, a custom packing routine can be developed.

All micro-protocols that detect a potential security problem with a message raise event *securityAlert* with the message and cause for the alert as arguments. The alert events will be handled by security audit micro-protocols. Multiple options are possible here, ranging from logging the event or notifying users to taking active steps to increase the security of a connection when an intrusion is suspected.

Key distribution and meta-security micro-protocols. The SecComm service includes key distribution micro-protocols based on the Diffie-Hellman algorithms, as well as asymmetric key distribution algorithms where either a client or a server creates a session key. Naturally, the latter schemes must use other predefined keys to secure the key exchange. Other key distribution micro-protocols based on certification authorities and Kerberos key distribution centers are planned.

The collection of meta-security micro-protocols includes MultiSec, which applies multiple basic security protocols to a message sequentially, and AltSec, which applies alternating basic security protocols to a message. Randomized versions of these micro-protocols apply the basic micro-protocols in random order, but must include information identifying the order so that the receiver can correctly decode the message. Even more complicated security protocols can be constructed by combining multiple meta-security micro-protocols.

4.2 Performance.

The current prototype implements a subset of the micro-protocols presented in this paper, including privacy micro-protocols based on DES, RSA, IDEA, Blowfish [Sch94b], and XOR, integrity micro-protocols based on MD5 and SHA, an authentication micro-protocol based on DSA, a time-stamp based replay prevention micro-protocol, and a non-repudiation micro-protocol. Other micro-protocols are currently being added.

We have conducted a number of preliminary experiments using different subsets of micro-protocols on the MK 7.3 Pentium cluster mentioned above. Table 1 gives examples of roundtrip times in milliseconds for passing 100-byte messages using different configurations. The average roundtrip times are determined by measuring the time for 500 roundtrips and then dividing the result by 500. All SecComm configurations were on IP, and the system was lightly loaded during testing. As baselines, an average roundtrip time directly on IP was 3.30 ms. The cost over IP column indicates the roundtrip time overhead of the configuration compared to roundtrip on IP. The cost over basic SecComm indicates the overhead of including the chosen micro-protocol(s) versus a minimal SecComm with only a trivial micro-protocol that simply passes the messages through untouched.

In these tests, DESPrivacy uses a 56-bit key running in CFB mode, BlowfishPrivacy uses a 448-bit key running in CFB mode, XORPrivacy uses a 64-bit "key," and IDEAPrivacy uses a 128-bit key running in CFB mode. The NonRepudiation tested ensures that messages are written to disk before the message is delivered to the next level. Other non-repudiation variants that allow delayed write to disk are naturally less expensive. (Note: results from more experiments will be included in the final paper.)

The cost over basic SecComm gives the most realistic indication of the cost of combining multiple micro-protocols. This column shows that in general, the cost of combining multiple micro-protocols is less than or roughly equal to the sum of the corresponding micro-protocol costs. For example, the cost for

Configuration	Roundtrip time	Cost over IP	Cost over basic SecComm
basic SecComm	4.06	0.76	n/a
XORPrivacy	4.27	0.97	0.21
DESPrivacy	7.24	3.94	3.18
XOR, DES	7.43	4.13	3.37
BlowfishPrivacy	5.72	2.42	1.66
DES, Blowfish	8.84	5.54	4.78
XOR, DES, Blowfish	9.24	5.94	5.18
IDEAPrivacy	8.30	5.00	4.24
XOR, DES, Blowfish, IDEA	13.23	9.93	9.17
MD5Integrity	6.04	2.74	1.98
SHAIntegrity	6.59	3.29	2.53
MD5, SHA	7.60	4.30	3.54
DES, MD5	8.69	5.39	4.63
NonRepudiation	58.58	55.28	54.52

Table 1: Roundtrip times (in ms)

combining DES and Blowfish is 4.78 ms, which is less than the sum of the corresponding costs of the single micro-protocols. The cases where the cost is slightly greater than the sum of individual micro-protocols can be attributed to measurement error since the interactions between different micro-protocols are very minimal in these cases.

4.3 Configuration constraints and composibility

A number of factors must be considered when micro-protocols are combined into a custom instance of the SecComm service. In particular, there are both algorithmic or property-based constraints that are independent of a particular implementation, and implementation constraints that are specific to our Cactus-based prototype. Algorithmic constraints are those that result from the inherent nature of properties being enforced or the algorithms used. For example, authenticity micro-protocols based on message digests require that an integrity micro-protocol create the digest. Similarly, the non-repudiation micro-protocol requires the use of an authenticity micro-protocol based on public keys. Finally, all micro-protocols that use a key require either that the key is provided when the session is created or that a key distribution micro-protocol is included.

Other algorithmic constraints affect the order in which various security algorithms are applied. For example, all attack prevention micro-protocols should execute before privacy, integrity, or authenticity micro-protocols at the sender to ensure that the mechanism used for attack prevention is protected from modification. Similarly, non-repudiation micro-protocols should be executed immediately before authentication at the receiver so that only the sender's private key is required to later prove the message was sent by the sender. Other ordering constraints have been identified elsewhere [AN96, AN95]. A related issue not addressed here but considered elsewhere is the actual effectiveness of multiple encryption and custom security solutions [Bla99, MH81, Rit99, Sch94a, Sch99].

Implementation constraints are those that result from the specific design of the SecComm micro-protocols. Compared with systems that support linear or hierarchical composition models, the non-hierarchical model

supported by Cactus introduces minimal implementation constraints on configurability. That is, with Cactus, it is generally possible to implement independent service properties so that this independence is maintained in the micro-protocol realization [Hil98]. When extra constraints do get imposed, it is usually because making an extra assumption about which other micro-protocols are present significantly simplifies the implementation.

In the current SecComm prototype, the only additional implementation constraint is that each integrity and replay prevention micro-protocol can be used at most once in a given configuration. Thus, for example, two instances of MD5Integrity cannot be used together, while MD5Integrity and SHAIntegrity can be. This restriction results from the use of fixed message attribute names for each micro-protocol, which could be avoided by dynamically assigning attribute names at startup time.

5 Related work

Related work can be divided into customizable security services based on configuration frameworks and customizability in secure communication standards. A number of configuration frameworks have been used to construct modular or configurable secure communication services including the *x*-kernel [OOSS94], Ensemble [RBH+98], and a Java implementation of the conduit model [NK98]. Each of these models allows a communication subsystem to be constructed as a stack or directed graph of protocol objects, which allows different security protocols be configured in arbitrary ways with respect to one another and other communication protocols. The *x*-kernel framework was used to develop a modular implementation of Kerberos as well as a suite of seven cryptographic protocols, including MD5, SHA, DES, RSA, DSS, and Diffie-Hellman [OOSS94]. The Ensemble work augments normal group communication services with a *Signing Router* module that provides integrity using keyed MD5, an *Encrypt* module that provides privacy using RC4, and *Exchange* and *Rekey* modules that establish an agreed group key using PGP for authentication [RBH+98]. Finally, the conduit model is used for a modular implementations of the IPSec and ISAKMP [MSST97] protocols [NK98].

The event-based model used in Cactus has a number of advantages compared with these hierarchical models, or others such as System V Streams [Rit84] or Scout. One advantage is that it is more flexible. While some combinations of security micro-protocols must be executed in a linear order for semantic reasons, our meta-security micro-protocols are just one example of how this flexibility can be used advantageously. The flexibility also provides a natural way of handling exceptional events such as key misses, dynamic key changes, or detected security violations that do not correspond to the normal linear message flow.

Another advantage of Cactus is that it alleviates property composition problems that are inherent in hierarchical models. When composing a collection of modules hierarchically into a subsystem, a property P enforced by some module M is only guaranteed for the entire subsystem if all other modules executed after M preserve P. Implementing modules to preserve other properties and keeping track of such preservation relations can be difficult. Furthermore, with hierarchical composition, it is difficult to ensure the entire collection of properties for new messages that may be generated by modules within the subsystem. In Cactus, these problems are lessened since different micro-protocols cooperate within the composite protocol to implement a service. Thus, a message that arrives at a composite protocol—or is generated within the composite protocol—can only leave when the properties enforced by all the micro-protocols are satisfied. The Cactus model allows such rich interactions, while providing mechanisms that maximize independence.

The Cactus model also provides an efficient composition framework since it avoids the per layer over-

head such as message demultiplexing associated with hierarchical models. Finally, Cactus does not force all messages to flow through all modules. In particular, it is easy to raise specialized events based on message type or the current system state, which are then fielded by only the required subset of micro-protocols.

As noted in the Introduction, customizability is supported in several recent communication security standards. For IPSec [KA98], the security option, AH or ESP, and the specific cryptographic algorithms and keys used for a connection are specified in a *security association* (SA). A SA may be created manually or negotiated using key management protocols such as IKE [HC98]. Multiple security methods may be specified for a connection by giving multiple SAs, called a *SA bundle* [KA98]. TLS [DA99], as well as its predecessor SSL [FKK96], offers two security layers. The lower level protocol, the TLS Record Protocol, provides privacy (e.g., DES or RC4), integrity (e.g., keyed SHA or MD5), and optional message compression. A higher level TLS Handshake Protocol authenticates the communicating partners and negotiates cryptographic algorithms and keys. The handshake protocol typically uses X.509 certificates to authenticate the server and potentially the client. A number of key exchange options are supported. Similar types of customization is provided in other Internet protocol proposals, including Privacy-Enhanced Mail (PEM) [Lin93] and the Secure HyperText Transfer Protocol (S-HTTP) [RS98]. The Secure Electronic Marketplace for Europe (SEMPER) proposal provides optional non-repudiation and anonymity for financial transactions in addition to privacy, authenticity, and integrity [Sem99].

IPSec, TLS, and SecComm have a similar goal of customizable secure communication, but with different constraints on the solution and different approaches to achieving this goal. For example, TLS is designed for communication between a client and server that do not have prior agreement on security configuration. It also does not directly support multiple encryptions or multiple message digests, nor does it provide non-repudiation. In addition, both IPSec and TLS are optimized for the case where a connection is used to send a large number of messages rather than a few messages, which means that good performance is "hard-wired" to be a high priority rather than something that can be customized in the context of the performance/security tradeoff. Moreover, although IPSec is relatively flexible, the approach presented in this paper offers a simpler design—for example, separate AH and ESP options and SA bundles are not needed—with unlimited flexibility in combining security techniques. Similarly, it appears that our design easily surpasses the flexibility of TLS.

Finally, note that IPSec and TLS are protocol specifications and therefore do not specify they must be implemented internally. Our approach could be used to configure, in essence, an instance of SecComm that is IPSec or TLS compliant. Of course, the message packing routines would have to be customized to generate IPSec and TLS compliant message formats, but this can be done using the customization facilities provided by the Cactus runtime system. Although IPSec and TLS can naturally be implemented in a modular manner without using Cactus, the benefits of the approach become more prominent when arbitrary combinations of methods are desired, when key distribution must be customizable, and, in particular, if the security mechanisms must change adaptively at runtime.

6 Conclusions

The ability to customize security attributes at a fine-grain level allows users to pick the most appropriate point along the security spectrum based on the characteristics of their particular application and security environment. SecComm is a security service designed to support this type of customization for the communication needs of networked applications. While similar in spirit to existing protocols such as IPSec and

TLS, SecComm goes beyond these to support more attributes and more variants, all within a flexible and extensible implementation framework based on micro-protocols and events. The design also decouples to a large extent the security aspects and the communication aspects of the problem. This allows, for example, SecComm to be used with multiple transport protocols and at multiple locations in the protocol hierarchy with little or no change. The current version of SecComm only provides the mechanisms for configurability, the eventual goal is to add a configuration support tool that can enforce the various composability constraints.

Once the prototype implementation is completed, we intend to experiment with the service in the context of various applications, including a configurable distributed file system that is also being built using Cactus. In addition, we will explore altering security attributes and techniques within the service dynamically using the adaptive facilities provided by the system. Our ultimate goal is to use this fine-grain configurability and fast adaptation ability as the basis for an *inherently survivable system architecture* that can automatically react to threats in the execution environment [HSUW00].

Acknowledgments

Gary Wong implemented the Cactus framework used for the SecComm implementation. He also provided excellent comments and suggestions that improved the paper.

References

[AN95] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Proceedings of Crypto'95*, pages 236–247, 1995.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, Jan 1996.

[BHSC98] N. Bhatti, M. Hiltunen, R. Schlichting, and W. Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, Nov 1998.

[Bla99] G. Blakley. Twenty years of cryptography in the open literature. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 106–107, May 1999.

[CP97] C. Cowan and C. Pu. Immunix: Survivability through specialization. In *Proceedings of the 1997 Information Survivability Workshop*, Feb 1997.

[DA99] T. Dierks and C. Allen. The TLS protocol, version 1.0. Request for Comments (Standards Track) RFC 2246, Certicom, Jan 1999.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[FKK96] A. Freier, P. Karlton, and P. Kocher. The SSL protocol, version 3.0. Internet-draft, Netscape Communications, Nov 1996.

[Gut98] P. Gutmann. Cryptlib. http://www.cs.auckland.ac.nz/∼pgut001/cryptlib/, 1998.

[HC98] D. Harkins and D. Carrel. The internet key exchange (IKE). Request for Comments (Standards Track) RFC 2409, Cisco Systems, Nov 1998.

[Hel78] M. Hellman. An overview of public-key cryptography. *IEEE Transactions on Communications*, 16(6):24–32, Nov 1978.

[Hil98] M. Hiltunen. Configuration management for highly-customizable software. *IEE Proceedings: Software*, 145(5):180–188, Oct 1998.

[HP91] N. Hutchinson and L. Peterson. The *x*-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan 1991.

[HS95] M. Hiltunen and R. Schlichting. Constructing a configurable group RPC service. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 288–295, Vancouver, BC, Canada, May 1995.

[HS98] M. Hiltunen and R. Schlichting. A configurable membership service. *IEEE Transactions on Computers*, 47(5):573–586, May 1998.

[HSH+99] M. Hiltunen, R. Schlichting, X. Han, M. Cardozo, and R. Das. Real-time dependable channels: Customizing QoS attributes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):600–612, Jun 1999.

[HSUW00] M. Hiltunen, R. Schlichting, C. Ugarte, and G. Wong. Survivability through customization and adaptability: The Cactus approach. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 294–307, Hilton Head, SC, Jan 2000.

[JJ98] N. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. *IEEE Computer*, 31(2):26i–34, Feb 1998.

[KA98] S. Kent and R. Atkinson. Security architecture for the internet protocol. Request for Comments (Standards Track) RFC 2401, BBN Corp, Home Network, Nov 1998.

[Lin93] J. Linn. Privacy enhancement for internet electronic mail: Part I: Message encryption and authentication procedures. Request for Comments RFC 1421, Feb 1993.

[MH81] R. Merkle and M. Hellman. On the security of multiple encryption. *Communications of the ACM*, 24(7):465–467, Jul 1981.

[MMO+95] A. Montz, D. Mosberger, S. O'Malley, L. Peterson, and T. Proebsting. Scout: a communications-oriented operating system. In *Proceedings of the Hot OS*, May 1995.

[MS95a] P. Metzger and W. Simpson. IP authentication using keyed MD5. Request for Comments RFC 1828, Piermont, Daydreamer, Aug 1995.

[MS95b] P. Metzger and W. Simpson. IP authentication using keyed SHA. Request for Comments RFC 1852, Piermont, Daydreamer, Sep 1995.

[MSST97] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol (ISAKMP). Internet-draft, Internet Engineering Task Force, Jul 1997.

[NK98] P. Nikander and A. Karila. A Java Beans component architecture for cryptographic protocols. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, Jan 1998.

[NT94] B. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, Sep 1994.

[OMG98] Object Management Group. *The Common Object Request Broker: Architecture and Specification (Revision 2.2)*, Feb 1998.

[OOSS94] H. Orman, S. O'Malley, R. Schroeppel, and D. Schwartz. Paving the road to network security or the value of small cobblestones. In *Proceedings of the 1994 Internet Society Symposium on Network and Distributed System Security*, Feb 1994.

[RBH+98] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev. The architecture and performance of security protocols in the Ensemble group communication system. Technical Report TR98-1703, Department of Computer Science, Cornell University, Dec 1998.

[Rey95] F. Reynolds. The OSF real-time micro-kernel. Technical report, OSF Research Institute, 1995.

[Rit84] D. M. Ritchie. A stream input-output system. *AT&T Bell Laboratories Technical Journal*, 63(8):311–324, Oct 1984.

[Rit99] T. Ritter. Cryptography: Is staying with the herd really best. *IEEE Computer*, 32(8):94–95, Aug 1999.

[Riv92] R. Rivest. The MD5 message-digest algorithm. Request for Comments RFC 1321, MIT and RSA Data Security, Inc., Apr 1992.

[RS98] E. Rescorla and A. Schiffman. The secure hypertext transfer protocol. Internet-draft, Terisa Systems, Inc., Jun 1998.

[Sch94a] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., New York, 1994.

[Sch94b] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Lecture Notes in Computer Science*, 809:191–204, 1994.

[Sch99] B. Schneier. Cryptography: The importance of not being different. *IEEE Computer*, 32(3):108–112, Mar 1999.

[Sem99] Semper Consortium. Advanced services, architecture and design. http://www.semper.org/info/#D10, Mar 1999.

[SHA95] *Secure Hash Standard*. National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., Apr 1995.

[SNS88] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings*, pages 191–202, Dallas, TX, Winter 1988.

[TMR96] F. Travostino, E. Menze, and F. Reynolds. Paths: Programming with system resources in support of real-time distributed applications. In *Proceedings of the IEEE Workshop on Object-Oriented Real-Time Dependable Systems*, Feb 1996.