Enhancing Survivability of Security Services using Redundancy*

Matti A. Hiltunen Richard D. Schlichting
AT&T Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07932
hiltunen/rick@research.att.com

Carlos A. Ugarte
Department of Computer Science
The University of Arizona
Tucson, AZ 85721
cau@cs.arizona.edu

Abstract

Traditional distributed system services that provide guarantees related to confidentiality, integrity, and authenticity enhance security, but are not survivable since each attribute is implemented by a single method. This paper advocates the use of redundancy to increase survivability by using multiple methods to implement each security attribute and doing so in ways that can vary unpredictably. As a concrete example, the design and implementation of a highly configurable secure communication service called SecComm are presented. The service has been implemented using Cactus, a system for building highly configurable protocols and services for distributed systems. Initial performance results for a prototype implementation on Linux are also given.

1 Introduction

Security services that provide attributes such as confidentiality, integrity, and authenticity typically implement each attribute using a single method. For example, in a secure communication service, confidentiality may be provided by DES and integrity by keyed MD5. Although such an approach may be secure in the traditional sense, it is not survivable—once a method is compromised, all security guarantees on the connection related to that attribute are gone. Each method is, in essence, a single point of vulnerability very much analogous to a single point of system failure when considering fault-tolerance attributes. This problem is the same for many other aspects of security, including authentication and access control.

This paper advocates the use of a standard fault-tolerance technique—redundancy—to increase the survivability of security services. For example, using this approach for secure communication, message integrity can be implemented by calculating redundant independent signatures, while confidentiality can be implemented by encrypting the message with a combination of algorithms with keys established using different methods. As a result, even if an intruder manages to find one key or break one algorithm, the security guarantees may remain intact. The task of the intruder can be complicated further by using secret combinations of methods or by dynamically altering the set of methods during the lifetime of the connection. By using multiple methods and doing so in ways that can vary unpredictably, the space of possibilities that must be considered by an attacker and the effort expended to compromise the attribute expands combinatorially. The approach also allows the tradeoff between the cost of the survivability and the protection to be managed explicitly and dynamically in response to changing threat scenarios.

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 supports extensibility by allowing the addition of new algorithms as separate modules. These attributes derive from the use of Cactus, a framework for constructing highly-configurable network services, as the underlying implementation platform [11].

This paper focuses on two key requirements for using redundancy to improve survivability: the development of appropriate techniques and the availability of suitable system support. We begin by discussing some specific redundancy techniques for both communication security and other security services, and then turn to the issue of system support. As an example of a system that has the necessary characteristics, we describe Cactus. This is followed by presentation of SecComm. Independent of survivability, SecComm is novel in its own right as a highly configurable and flexible secure communication service.

*This work supported in part by DARPA under grant N66001-97-C-8518 and NSF under grants ANI-9979438 and CDA-9500991.

2 Techniques

Implementing privacy and other attributes. The basic idea behind using redundancy to improve survivability is simple—with multiple methods enforcing a given attribute, the attribute should remain valid if at least one of the methods remains uncompromised. As with fault tolerance, however, the effectiveness of the approach depends on the details of how it is used. As an example of this approach, we focus on using redundant cryptographic methods to ensure communication privacy.

In this context, a number of specific techniques can be devised based on redundant encryption. The simplest, of course, is to apply the different methods successively on the same data. For example, a message might first be encrypted using DES then IDEA. However, there are many other approaches, including:

- Alternating the order in which methods are applied, e.g., apply DES before IDEA for some messages and IDEA before DES for others.
- Applying different methods to different parts of the data, e.g., encrypt different parts of one message or different messages in a stream using different methods.

An important factor influencing the effectiveness of these approaches is the *independence* of the methods used, where two methods A and B are independent if compromising A provides no information that makes it easier to compromise B, and vice versa. A simple example of non-independence is when two encryption methods use the same key, since if one method is broken or the key stolen, privacy is completely compromised. To maximize independence in this type of situation, the keys should be established using different key distribution methods, for example, one using Diffie-Hellman and the other using Kerberos. As a result, the system may even be able to tolerate an attack where one or more of the distribution methods are compromised.

Note that this type of independence is very much analogous to the fault-tolerance concept of independent failure modes for redundant hardware or software components. Components are independent in this sense when the failure of one component does not affect the correct execution of any other component.

While the independence of encryption methods is difficult to argue rigorously, the risk of methods not being independent is likely to be minimized if the methods are substantially different or if they encrypt data in different size blocks. It is also possible to develop combinations that attempt to maximize independence by not simply encrypting the same data multiple times, but by finding other ways to combine different encryption methods. For example, suppose that m is a cleartext message and E_1 and E_2 are different encryption methods. A ciphertext message cm could be constructed as $cm = \{E_1(\mathrm{m} \otimes \mathrm{r}),\ E_2(\mathrm{r})\}$, where $\otimes$ is the exclusive-or operation and r is a random bit sequence the same length as the message. Given this method, breaking only E_1 or E_2 does not produce any useful information, which means that the attacker has to break both simultaneously to know if the system has been compromised. As a result, the effort required is multiplicative.

Determining independence of methods is easier for other security attributes such as message integrity. Let m be the message to be protected and $d_1(m), d_2(m), ...$ be different cryptographic message digests of m. Since the message digest algorithms operate on the message independently, an attacker would need to compromise each integrity algorithm separately. In this case, the increase in the breaking effort is additive since the attacker knows when each method has been broken.

Finally, it is also possible to exploit analogues of fault-tolerance techniques that operate on a sequence of messages rather than on individual messages. For example, using techniques similar to forward error correction (FEC), message modification or modifications of the message stream (i.e., insertions and deletions) could be detected. Such techniques can naturally be used together with message-based methods to increase survivability further.

Other security services. Similar ideas can be applied to other types of security services in distributed systems as well. For example, redundancy can be used to increase the survivability of certification agencies and the PKI. If multiple certificates are required from multiple independent certification agencies or a user's public key is verified with a number of public key servers, the chance that an intruder can cause extensive damage by compromising one agency is reduced.

In areas of security where the existing operating system already provides mechanisms—e.g., authentication and file access control—overall system survivability can sometimes be improved by introducing redundant independent methods to enforce the desired security guarantees or to detect violations. For example, access control can be augmented with encryption, in which case a user can only read a file if allowed by the access control system and with the necessary key.

Similarly, an intrusion detection system (IDS) can be viewed as a redundant component that monitors user behavior to detect illegal activities that are acceptable to the standard operating system security mechanisms. If the IDS further employs redundant detection modules that implement different techniques, it stands an even greater chance of accurately detecting an intrusion. The agreement between the different modules can be tuned to obtain the desired tradeoff—allowing a single module to trigger an alarm results in discovering more intrusions, while requiring more modules to agree reduces the number of false positives.

Redundancy mechanisms can also be developed to address specific security problems. For example, the Cactus framework discussed below has been used to develop a distributed system monitoring tool that was extended with a module designed to deal with an intruder modifying the web pages of an organization. This module monitors a subtree of the directory structure by comparing a checksum of each file against the previous checksum of the same file. If a file is modified, the system administrator is then notified. This approach allows unauthorized web page modifications to be detected quickly using redundant checks.

3 System Support

The realization of services that use redundancy-based survivability techniques such as those described above can be simplified using an appropriate software customization framework. Here, we discuss one such system called Cactus, together with SecComm, a highly-customizable secure communication service that illustrates the potential of using redundancy to enhance survivability.

Cactus. Cactus is a system for constructing configurable network protocols and services where each service property or functional component is implemented as a separate software module called a *micro-protocol* [11]. A customized instance of a service is then created by choosing a collection of micro-protocols based on the properties to be enforced, and configuring them together with the Cactus runtime system to form a *composite protocol* that implements the service on each machine. A micro-protocol is structured as a collection of event handlers that are executed when a specified event occurs. Events can be raised explicitly by micro-protocols or by the runtime.

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. Handler execution is atomic with respect to concurrency, i.e., 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 issued the raise. 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 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 [21] and Red Hat Linux release 6.2, another written in C++ that runs on Solaris and Linux, and a third written in Java that runs on multiple platforms.

SecComm overview. SecComm is a highly configurable secure communication service with the inherent flexibility needed to realize redundancy-based survivability techniques. The system model for SecComm 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.

SecComm 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 below.) 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

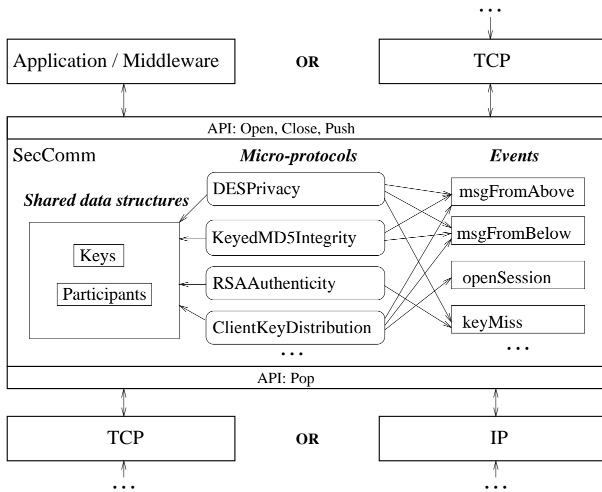

Figure 1. System protocol stack.

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 [12], CORDS [25], and Scout [18] 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 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.

Security properties. As a first step towards exploiting redundancy, security properties and their variants are identified, each of which can be implemented using different methods. 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, any shared secret, or a trusted intermediary such as Kerberos.
- **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. 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.

4 SecComm Design

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 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 RSAAuthenticity, 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.

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 Cryptlib cryptographic package [10] 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.

Micro-protocol structure. The abstract security attributes described in section 3, 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 the 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. The ability to use multiple micro-protocols within a given class at the same time, for example, is one way in which redundancy can be used to support a survivable service. Naturally, there may be some configuration constraints between micro-protocols that restrict which combinations are feasible.

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,

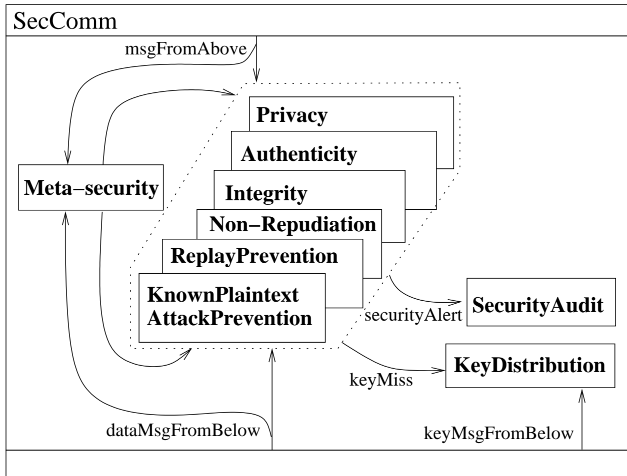

Figure 2. MP classes and interactions.

```
micro-protocol BasicSecurity(dEvnt,dOrd,uEvnt,uOrd,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(dEvnt,ProcessDownMsg,dOrd);
    bind(uEvnt,ProcessUpMsg,uOrd);
  }
}
```

Figure 3. Generic basic security MP.

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.

Basic security micro-protocols. The basic security micro-protocols are simple, typically consisting of two event handlers and an initialization section. One of the event handlers is used for the data 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 (figure 3) typically takes 4 or 5 arguments. In this parameter list, *dEvnt* and *uEvnt* 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 *dOrd* and *uOrd* parameters are the relative orders in which this particular security micro-protocol is to be applied to messages flowing down and up, respectively. Finally, *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. 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)

As a complex example involving redundant application of encryption, 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 *dEvnt* and decrypts messages associated with *uEvnt* 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. A version that uses different algorithms in sequence could be done similarly.

Meta-security micro-protocols. In this design, meta-security micro-protocols construct more complex security protocols out of the basic security protocols, and are a key feature enabling redundancy for survivability. For example, a meta-security micro-protocol may apply multiple or alternating basic security micro-protocols to a message. The

```
micro-protocol MetaSecurity(dEvnt,dOrd,uEvnt,uOrd,
                             dBasicEvnts,uBasicEvnts) {

  handler ProcessDownMsg(msg){
    in some order raise(dBasicEvnts[i],msg,SYNC);
  }
  handler ProcessUpMsg(msg){
    in some order raise(uBasicEvnts[i],msg,SYNC);
  }
  initial {
    bind(dEvnt,ProcessDownMsg,dOrd);
    bind(uEvnt,ProcessUpMsg,uOrd);
  }
}
```

Figure 4. Generic meta-security MP.

basic structure of a meta-security micro-protocol is shown in figure 4.

Examples of the use of meta-security micro-protocols include:

- MultiSecurity. Applies multiple basic security protocols to a message in sequence.
- AltSecurity. Applies one security micro-protocol to each message, with the method chosen successively from a specified list. If the sequence of methods is deterministic, or agreed by the sender and receiver, no additional information is required provided that the underlying communication is reliable and maintains FIFO ordering.
- RandomAltSecurity. Similar to AltSecurity but uses a randomly chosen method for each message. Each message must carry an identifier than can be used by the receiver to determine which method to use to decrypt the message.

A meta-security micro-protocol can also be configured to use other meta-security micro-protocols. For example, we can create a configuration that applies alternating different multiple encryption methods to each message.

A final example meta-security micro-protocol is ExpansionSecurity, which breaks the message body into two parts by using bit masks so that the original message is equivalent to part one xor part two. It then applies one security micro-protocol to part one and another to part two. This makes it very difficult for the intruder to break the security unless they can break both at the same time.

The concept of meta-security micro-protocols can be applied to increase the survivability of any security property for which using multiple or alternating methods provides enhanced guarantees. Privacy, authenticity, and message integrity among others fall in 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.

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 redundancy techniques used for data security can also often be applied for key distribution security. Multiple key distribution micro-protocols can also be used to obtain keys redundantly.

Redundancy and key distribution can mix in different ways. For example, relying on redundant trusted arbitrators to obtain a key in an external scheme can avoid some of the problems that occur if a single arbitrator is used and compromised. Moreover, if the multiple arbitrators are thought to be vulnerable to the same attack, different algorithms can be used. This is another instance when the tradeoff between survivability and cost can be tuned.

In the above, the multiple methods are used collaboratively to obtain the same key. The scheme can also be used to collect different keys, however. The simplest scenario has each key assigned to a separate basic security micro-protocol. A more complex configuration would allocate multiple keys to the same micro-protocol, which could use alternate keys on a message by message basis.

5 Implementation and Performance

A prototype of SecComm has been implemented using the C version of Cactus on two different clusters. One is a cluster of 133 MHz Pentium PCs running OpenGroup/RI Mach MK 7.3 and CORDS connected by a 10 Mb Ethernet, and the other is a cluster of 600 Mhz Pentium PCs running Red Hat Linux release 6.2 connected by a 1 Gb

Ethernet. This section provides some initial performance numbers from the Linux cluster and discusses issues related to configuring collections of micro-protocols into a custom instance of the SecComm service.

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, 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, a non-repudiation micro-protocol, and two meta-security micro-protocols. Other basic security and meta-security micro-protocols are currently being added.

We have conducted a number of experiments using different subsets of micro-protocols. Table 1 gives roundtrip times in milliseconds for passing 100-byte messages using different configurations. The average roundtrip times were computed over 1000 or more roundtrips. All SecComm configurations use IP, and the system was lightly loaded during testing. As a baseline, an average roundtrip time using IP directly is 1.202 ms. The entry for base SecComm reflects times for a skeleton version of SecComm that does not use any micro-protocols; its additional cost indicates the approximate cost of adding a new *x*-kernel protocol to the stack. The cost over IP column indicates the roundtrip time overhead of the configuration compared to using just IP. Similarly, the cost over base column indicates the overhead of the configuration compared to just the base.

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.

The cost over base column provides the most realistic indication of the cost of combining multiple micro-protocols. For MultiSecurity, these numbers indicate that the cost is roughly equal to the sum of the costs associated with the corresponding micro-protocols. For example, the overhead of using MultiSecurity to combine DES and Blowfish is 0.385 ms, which is actually slightly less than the sum of the costs of DES and Blowfish since the cost of using Cactus mechanisms is amortized over multiple micro-protocols. For AltSecurity, the cost is approximately the same as the average cost of the individual micro-protocols.

Configuration constraints. A number of factors must be considered when micro-protocols are combined into a custom instance of the SecComm service, including those using redundancy to enhance survivability. 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, the non-repudiation micro-protocol requires the use of an authenticity micro-protocol based on public keys. Similarly, 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.

Configuration	RTT	Cost over IP	Cost over Base
IP	1.202	n/a	n/a
Base SecComm	1.245	0.043	n/a
XORPrivacy	1.365	0.163	0.120
DESPrivacy	1.504	0.302	0.259
BlowfishPriv.	1.442	0.240	0.197
IDEAPrivacy	1.511	0.309	0.266
MD5Integrity	1.577	0.375	0.332
SHAIntegrity	1.598	0.396	0.353
Non-repudiation	4.075	2.873	2.830
DES + MD5	1.798	0.596	0.553
MultiSecurity			
DES	1.515	0.313	0.270
XOR + DES	1.599	0.397	0.354
DES + Blowfish	1.630	0.428	0.385
+ XOR	1.694	0.492	0.449
+ IDEA	1.963	0.761	0.718
AltSecurity			
DES	1.497	0.295	0.252
XOR + DES	1.481	0.279	0.236
DES + Blowfish	1.511	0.309	0.266
+ XOR	1.496	0.294	0.251
+ IDEA	1.513	0.311	0.268

Table 1. Roundtrip times (in ms)

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 public key is required to later prove the message was sent by the sender. Other ordering constraints have been identified elsewhere [1, 2]. A related issue not addressed here but considered elsewhere is the actual effectiveness of multiple encryption and custom security solutions [17, 24].

Implementation constraints are those that result from the specific design of the SecComm micro-protocols. Compared with systems that support linear or hierarchical com-

position 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. 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.

6 Related work

Related work includes the general use of redundancy to enhance survivability, as well as research more directly related to secure communication. Redundancy has traditionally been used to improve file system fault tolerance, but different redundancy techniques have also been used to increase security, or both fault tolerance and security. For example, cryptographic methods were used to store data on untrusted file servers [9] and data fragmentation and replication techniques have been used to prevent intruders from accessing, modifying, or destroying information [6, 8, 14]. Recent developments provide similar guarantees, but also ensure anonymity of the information publisher [3, 26]. Although none of this research is cast in terms of survivability, it can also be viewed in that context.

Other approaches do not replicate data, but introduce redundant detection components. Examples include Tripwire [15], which detects changes in files by maintaining checksums and periodically comparing the files against checksum, and StackGuard [4], which detects buffer overflow attacks by storing a secret "canary" word in the stack and checking it upon function call return. Intrusion detection in general augments a system with a component that detects undesired behavior that would be otherwise allowed by the security mechanisms of the given system [5].

Work specifically related to SecComm can be divided into secure communication standards and other configurable secure communication services. Some degree of customization is supported in several recent standards. For example, IPsec allows a choice of security options, including message integrity and privacy using a selected cryptographic method [13]. It is also possible to apply multiple security methods to a given communication connection. TLS (Transport Level Security) [7] offers a choice of privacy (e.g., DES or RC4), integrity (e.g., keyed SHA or MD5), and optional message compression, but does not directly support the use of redundant methods. In general, SecComm offers more flexible options for using redundancy techniques to enhance the survivability of such services.

Configurable secure communication services have been implemented using various configuration frameworks, including the *x*-kernel [20], Ensemble [22] and the framework described in [19]. All these models are similar in the sense that a communication subsystem is constructed as a directed graph of protocol objects. Although this allows arbitrary combinations of security components, the structure is limiting compared to Cactus and would make it difficult to implement some of our more dynamic redundancy techniques. However, Antigone [16] has adopted an approach similar to Cactus in which micro-protocols and composite protocols are used to implement secure group communication with customizable policies, including rekeying and message security. To our knowledge, none of these projects have explored the use of redundancy techniques in security.

7 Conclusions

This paper has discussed the use of redundant techniques as the basis for improving the survivability of security services. While the idea of combining fault tolerance and security is not new, this paper promotes a more general application of redundancy techniques in different areas of security and introduces a convenient implementation platform for such techniques. Our approach can also be viewed as a way of artificially increasing the diversity of the system, which has been advocated elsewhere as a potential approach to improving survivability [23]. In addition, we described SecComm, a security service that allows customization of security attributes at a fine-grain level. 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.

Future work will focus on using the Cactus framework to implement dynamically adaptable security services, where the security mechanisms are changed at runtime in reaction to changed security requirements (e.g., suspected intrusion) or changes in available resources. The Cactus framework makes it easy to activate and deactivate micro-protocols at runtime, and we have designed and implemented coordination mechanisms that allow adaptations across machines and across system layers to occur smoothly without interrupting normal operation. 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.

Acknowledgments

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

References

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

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

[3] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceeding of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, Jul 2000.

[4] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 119–129, Hilton Head, SC, Jan 2000.

[5] D. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, Feb 1987.

[6] Y. Deswarte, J.-C. Fabre, J.-M. Fray, D. Powell, and P.-G. Ranea. Saturne: A distributed computing system which tolerates faults and intrusions. In *Proceedings of the Workshop on Future Trends of Distributed Computing Systems*, pages 329–338, Hong Kong, Sep 1990.

[7] T. Dierks and C. Allen. The TLS protocol, version 1.0. RFC (Standards Track) 2246, Jan 1999.

[8] J.-C. Fabre, Y. Deswarte, and B. Randell. Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach. In *Proceedings of the 1st European Dependable Computing Conference*, pages 21–38, Berlin, Germany, Oct 1994.

[9] J. Fraga and D. Powell. A fault and intrusion-tolerant file system. In *Proceedings of the IFIP 3rd International Conference on Computer Security*, pages 203–218, Dublin, Ireland, 1985.

[10] P. Gutmann. Cryptlib. Department of Computer Science, University of Auckland, 1998.

[11] 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.

[12] 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.

[13] S. Kent and R. Atkinson. Security architecture for the internet protocol. RFC (Standards Track) 2401, Nov 1998.

[14] H. Kiliccote and P. Khosla. Borg: A scalable and secure distributed information system. In *Proceedings of the Information Survivability Workshop 1998*, pages 101–105, Orlando, FL, Oct 1998.

[15] G. Kim and E. Spafford. The design and implementation of tripwire: A file system integrity checker. In *2nd ACM Conference on Computer and Communications Security*, pages 18–29, Fairfax, Virginia, Nov 1994.

[16] P. McDaniel, A. Prakash, and P. Honeyman. Antigone: A flexible framework for secure group communication. In *Proceedings of the 8th USENIX Security Symposium*, pages 99–114, Aug 1999.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] F. Schneider, editor. *Trust in Cyberspace*. Committee on Information Systems Trustworthiness, National Research Council, National Academy Press, Washington, D.C, Sep 1998.

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

[25] 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.

[26] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, Aug 2000.