Analysis of the Ciphire System's Resistance to Insider Attacks

Technical Report 05-01
January 10, 2005

Bruce Schneier

Counterpane Systems
101 East Minnehaha Parkway
Minneapolis, MN 55419, USA
tel: +1 (612) 823-1098
schneier@counterpane.com

Contents

1 Executive Summary

We analyzed the Ciphire e-mail encryption system, as documented in [1] and [2], and as further explained in [5]. We focused our analysis on investigating ways that a Ciphire insider or set of Ciphire insiders might attack the users of the Ciphire System. We assume that such an insider would have access to the internal Ciphire infrastructure, Ciphire private cryptographic keys, and so on.

The driving question of our analysis was: can the users of the Ciphire System feel confident that Ciphire insiders cannot or will not compromise the confidentiality and authenticity of their (the users') communications? This question has two underlying components. First (1), is it possible for a Ciphire insider to compromise the privacy or authenticity of a Ciphire user's communication? Second (2), if it is possible for a Ciphire insider to compromise the security of a user's communication, will it be possible for the Ciphire user to, post facto, detect the fact that an attack has taken place, and determine whether the attack was mounted by a Ciphire insider? For the latter, the hope is that the chance of detection and incrimination will deter an insider attack.

Our results. At a high level, we are very pleased with the care and thought clearly put into the design of the Ciphire System. We also commend the designers of the Ciphire System for being concerned enough about the security of its users to try to design the Ciphire System to resist not only attacks from outsiders, but also attacks originating from those within the Ciphire infrastructure.

To address our question (1) from above, in our analysis we find that a Ciphire insider will be able to compromise the privacy or authenticity of a Ciphire user's communications. To address our question (2), for some of the insider attacks that we uncover, it may be possible for a powerful-enough Ciphire insider to prevent the detection of an attack. Still, doing so could be very challenging, if not infeasible, for a real attacker. Thus, in practice we can probably assume that legitimate Ciphire users will eventually detect the presence of these attacks, although not until after having had the security of their communications compromised. Unfortunately, for some of our other attacks, even if detection were possible, it would be impossible for the user to convincingly point the blame for the attack at a Ciphire insider; these attacks could appear to have been mounted by an outsider, although these attacks might be easier to mount from within. For this latter class of attacks, there will be little "threat of discovery" to deter the Ciphire insider from mounting them in practice.

Our three main classes of attacks are summarized in the following three headings.

Trojan horses. We understand that Ciphire plans to publicly release the source code for their Ciphire client application this year. That will be a major (and necessary) first step toward convincing users that there are no Trojan horses or backdoors in the Ciphire client that an insider could use to compromise the privacy or authenticity of a user's communications. Still, although releasing the source code to the Ciphire client is a *necessary* step, it is not *sufficient*. In Section 3, we consider how a Ciphire insider might embed the hooks for malicious code in the Ciphire client in ways that are both very hard to find and that appear to be accidental coding mistakes or bugs instead of intentionally placed backdoors. Further, although the insider would clearly have an advantage in mounting this attack (since the insider might have maliciously inserted the "bug"), an outsider could also mount an attack that makes use of the "bug."

Since our attacks in Section 3 would not incriminate the insider, and could appear to be the result of an honest programming mistake on the part of the Ciphire authors, we view this class of attacks as a realistic way that an insider might try to compromise the security of Ciphire users' communications. Even if the "bugs" were later uncovered and patched, the Ciphire insider would likely have had amply opportunity to make use of those "bugs." Additionally, the insider could insert similar "bugs" in newer generations of the Ciphire client.

Man-in-the-middle attacks. If security is a chain, the security of an entire system is only as strong as its weakest link. In the case of the security of the Ciphire System against insider attacks, we believe that one of the weakest links is in the software implementation of the Ciphire protocol, as described in the above heading.

Nevertheless, even if we ignore our concerns about the software implementation and focus solely on the cryptographic aspects of the Ciphire System, we find that an insider can still mount man-in-the-middle attacks against Ciphire users without triggering any detection mechanisms built into the Ciphire clients themselves (Section 4). The attacks in Section 4 bypass the Ciphire Fingerprint subsystem by carefully controlling the flow of information between different users of the Ciphire System.

Although the man-in-the-middle attacks we consider may be difficult to mount in practice, they do prove that the Ciphire System is not immune to man-in-the-middle attacks from Ciphire insiders. Two questions remain. First, although we claim that Ciphire could avoid triggering any detection

mechanism built into the Ciphire clients, how difficult will it be for the insider to maintain this deception? Second, could the attacks be detected using out-of-band means? Although it is impossible to precisely quantify the cost of an attack, in general we believe that the answer to the first question is that it will be unrealistic to assume that a real-world adversary will be able to maintain the deceit indefinitely. With regard to the second question, our attacks in Section 4 can be detected using out-of-band mechanisms.

Exploiting Ciphire's ease-of-use. The Ciphire e-mail system is designed to be very transparent and easy to use: the Ciphire Mail client, which sits between a user's mail client and the user's mail server, will transparently intercept outgoing messages, automatically look for the existence of a certificate for the e-mail address in the `To` line of the message, and then encrypt the message if a valid certificate is found.

The adversary could use tricks to convince the sender to send e-mail to the wrong address. For example, if an encrypted e-mail from Alice to Bob did not originally contain a `Reply-To` field, the adversary could insert a bogus `Reply-To` field in the header of the encrypted e-mail. If the adversary is able to do this, the adversary could trick Bob into sending his reply e-mail to an address for which the insider knows the corresponding private keys. Since the encryption happens transparently to Bob, and is based solely on the recipient's e-mail address, Bob's reply to Alice will be encrypted under a key created by the adversary and inserted in the Ciphire data structures by an insider.

We consider these classes of attacks in Sections 6 and 7. As with our Trojan horse attacks, these attacks could be mounted by both outsiders and insiders, although in some cases the outsider may find it more difficult to register certificates for the e-mail address to which Bob will eventually send e-mail. Unlike our man-in-the-middle attacks, these attack might not incriminate the Ciphire insider. Therefore, we believe that the attacks in Sections 6 and 7 will be another preferred way for Ciphire insiders to compromise the security of the Ciphire System.

Summary. Protecting against attacks from insiders is an incredibly daunting, if not impossible, task. Given such constraints, we believe that the Ciphire System still performs remarkably well. Our concerns with Trojan horses in Section 3 would be applicable regardless of what underlying cryptographic mechanisms Ciphire decided to employ. In practice, eventually our man-in-the-middle attacks in Section 4 will likely be detected by Ciphire users; this is in some sense the best that we can hope for, since preventing such attacks in the first place may be impossible. Our attacks in

Sections 6 and 7 are byproducts of Ciphire's ease-of-use, and thus cannot be fully addressed without compromising on Ciphire's ease-of-use design goal.

Addendum. We analyzed the version of the Ciphire System described in the documents [1] and [2]. From recent communications with Ciphire, we understand that Ciphire Labs is planning to address or has addressed some of the issues that we raise. This review does not reflect those changes.

2 The Ciphire System

In this section we summarize the design of the Ciphire System and the Ciphire Fingerprint System. We describe how the Ciphire System is supposed to behave in the absence of an adversary. Since an adversary will try to subvert the intended operation of the Ciphire System, we consider the (potential) effects of an adversary in subsequent sections.

2.1 Cast of Characters

As is standard in much cryptographic literature, and to facilitate our discussions, in our discussions we shall use a cast of three main characters:

Alice: Alice is a user of the Ciphire System. Alice's (primary) e-mail address is `alice@company.com`.

Bob: Bob is another user of the Ciphire System. Bob's (primary) e-mail address is `bob@company.com`.

The adversary: In our analysis we assume that the adversary has "insider" privileges.

The insider could be a Ciphire employee (or set of employees) with access to Ciphire's infrastructure, private keys, development environment, and so on. The adversary could also be an individual or group of people who were originally outsiders but who have broken into and compromised Ciphire's internal infrastructure.

2.2 Overview of the Ciphire System

The Ciphire System is an e-mail encryption and signature system designed for "ease of use." There are two general installation modes for the Ciphire System: it can be installed and used by individual users, or it can be installed by corporations to act as an "encryption gateway." When installed by an individual user, the Ciphire client sits between the user's mail client and the user's mail server and automatically intercepts and, if possible and depending on the user's settings, encrypts and signs the user's outgoing messages. The Ciphire client also decrypts and verifies the signatures on incoming encrypted messages before delivering them to the user's mail client.

Each individual user of the Ciphire system has a set of private and public keys, and each user has a certificate signed by a Ciphire Certificate Authority. The certificate is stored in the Ciphire Certificate Directory and is made available to other users of the Ciphire System. When the Ciphire System is installed by a corporation as an "encryption gateway," the high-level architecture is not significantly different: the corporation would have a set of public keys signed by the Ciphire Certificate Authority and included in the Ciphire Certificate Directory. When an e-mail is sent to a user within a corporation, it will be encrypted to the corporation's public keys. Thus, for the purposes of this review, when gateway encryption is used, one can generally think of the corporation as a single Ciphire entity. If individual users within the corporation have their own Ciphire certificates, however, then e-mails to those users will be additionally encrypted to those users' public keys.

The aforementioned "ease of use" comes from the fact that, after installing the Ciphire Mail client, the process for encrypting and decrypting messages is practically transparent to the user. When Alice wishes to send an e-mail to Bob, she writes the message as normal, using her normal e-mail client. She then sends the message as she would send any other message; e.g., by clicking on the "send" button; she does not need to click any button like "encrypt and sign." The Ciphire Mail client will intercept the outgoing message and look at the destination e-mail address (`bob@company.com`). Assuming that Alice has not communicated with Bob recently, the Ciphire client will communicate with the Ciphire Certificate Directory (or a proxy) and try to obtain Bob's latest certificate. If Alice's Ciphire client is able to find and verify Bob's certificate, it will encrypt Alice's message to Bob, and then send that encrypted message to Alice's mail server. If Alice's Ciphire client is unable to find and verify Bob's certificate, then depending on the settings that Alice pre-specified for the Ciphire client, the Ciphire client will send the message to Bob unencrypted, return an error message to Alice's mail client, or prompt Alice, asking her whether she would like the e-mail

sent unencrypted or not. In subsequent sections, we consider what "verify" in the previous sentences really means, including how "verify" depends on certain properties of the Ciphire Fingerprint subsystem.

The decryption process is analogous. For signature verification, the process for finding and verifying the signer's public key is also automated and transparent.

2.3 Enrolling in the Ciphire System

For individual users, Ciphire certificates bind a set of public keys to an e-mail address, such as `alice@company.com`. For corporations using the Ciphire System as an "encryption gateway," Ciphire certificates bind a set of public keys to a domain name or host name, such as `@company.com`.

For an individual user, Alice, to obtain a certificate and enroll in the Ciphire System, Alice's Ciphire client will first generate an RSA public and private key pair (for encryption and signatures), an ElGamal public and private key pair (for encryption), and a DSA public and private key pair (for signatures). Alice's client will then send a Certification Signing Request to the Ciphire Certification Authority. To verify that the public keys do indeed belong to a person who can receive e-mail at `alice@company.com`, the Ciphire Certification Authority sends a challenge back to `alice@company.com`. Using her private keys, Alice will reply to the challenge. Upon receiving Alice's reply, the Certification Authority will sign Alice's certificate. Alice signs her certificate before the Certification Authority does, and Alice's signature is included in the data that the Ciphire Certification Authority signs. Each certificate has a unique Certificate Identity (CID), assigned by the Ciphire Certification Authority. The e-mail address stored in the certificate is referred to as the certificate's Address Identity (AID).

After being signed by the Ciphire Certification Authority, Alice's certificate is given to the Ciphire Certificate Directory. When another user's client software wishes to obtain Alice's certificate, it will do so by accessing the Ciphire Certificate Directory, or a proxy sever.

2.4 Revoking Certificates

As designed, the Ciphire System only allows one certificate to be associated with a given e-mail address (or domain name in the case of gateway encryption) at any given time. This section describes the method for revoking a certificate and, if desired, obtaining a new one.

2.4.1 Standard Revocation

To revoke Alice's current certificate, Alice's Ciphire client will create a revocation request, signed using Alice's current private keys, and sends that request to the Ciphire Certification Authority. The Certification Authority will then issue a revocation certificate that is identical to a regular (non-revocation) certificate except that the "certificate type" field will indicate that the certificate is a revocation certificate. The Certificate Identifier of the revoked certificate will be included as part of the data signed in the new revocation certificate.

If Alice chooses to create a new set of public and private keys at the same time, then the revocation certificate for the old set of keys will include a forward reference to the Certificate Identifier of Alice's new certificate. Furthermore, part of the data in the revocation certificate will be signed by Alice's new public keys, and those signatures will be included in the data signed by the Certification Authority as part of the revocation certificate. In the Ciphire documentation, this is referred to as certificate chaining.

2.4.2 Emergency Revocation

When Alice creates a new set of public and private keys and obtains a certificate, she will also provide the Ciphire emergency revocation system with a signed revocation request, encrypted under a passphrase of Alice's choice. The signed revocation request is encrypted by Alice's client software before being uploaded to the Ciphire emergency revocation system.

If Alice loses her private keys and cannot revoke her current certificate using the standard method, she must invoke the Ciphire emergency revocation system. She does this by entering her e-mail address and her passphrase on a Ciphire website. Using Alice's e-mail address, this website will look up Alice's current certificate and encrypted revocation request, decrypt the encrypted revocation request using Alice's passphrase, and then issue a revocation certificate as in Section 2.4.1.

2.5 Fingerprint Lists

The purpose of the Ciphire Fingerprint System is to help mitigate attacks against users of the Ciphire System from an attacker who has compromised the Ciphire infrastructure or Ciphire private keys; e.g., the attacker might be a Ciphire employee or set of Ciphire employees. The approach taken by the Ciphire Fingerprint List System is to compute and regularly distribute functions of the hashes of the certificates that the Ciphire Certification Authority issues. The hope is that an attacker with access to the Ciphire

internal infrastructure should not be able to issue rogue certificates or modify existing certificates without triggering detection mechanisms built into users' Ciphire client software.

In more detail, let $\mathsf{Cert}_1, \mathsf{Cert}_2, \ldots, \mathsf{Cert}_n$ denote the certificates or revocation certificates issued during a given time interval; e.g., one hour. Let $\mathsf{AID}_1, \mathsf{AID}_2, \ldots, \mathsf{AID}_n$ denote the respective Address Identifiers (e-mail addresses or domain names) in the certificates, and let $\mathsf{CID}_1, \mathsf{CID}_2, \ldots, \mathsf{CID}_n$ denote the respective Certificate Identifiers. The *fingerprint* for the i-th such certificate is the concatenation of $H(\mathsf{AID}_i)$, $H(\mathsf{CID}_i)$, $H(\mathsf{Cert}_i)$, and M_i, where H is a hash function and M_i is two bytes of metadata indicating the type of the i-th certificate. For example, M_i might indicate that the i-th certificate is a revocation certificate.

At the end of each time interval, the fingerprints for the time interval are combined into a hash tree. The leaves of the tree, or the *Branch FPLs* (fingerprint lists) contain the fingerprints for the certificates. The Branch FPL that a given certificate is assigned to depends on the hash for the Address Identifier, $H(\mathsf{AID}_i)$. Hashes of the Branch FPLs are stored in *Section FPLs*, and hashes of the Section FPLs are stored in a *Master FPL*. In the current Ciphire System, each Master FPL has two children Section FPLs, and each Section FPL has two children Branch FPLs. Each Branch, Section, and Master FPL contains header and footer information indicating the time interval for the FPL, a hash over the complete FPL of the previous and current intervals, and other metadata information.

A *Cross FPL* contains hashes of the Master FPLs for different time intervals. Specifically, each entry in the Cross FPL contains at least information indicating the end time of the corresponding interval and a Cross FPL Hash, which is calculated over all previous Cross FPL entries (hashes and timestamps) and the full contents of the corresponding Master FPL. The Cross FPL is signed by the Ciphire Fingerprint Authority, and is made available to users of the Ciphire system.

There are several different configurations for Ciphire clients with respect to how they download portions of the Ciphire Fingerprint data structures. In the default mode, a Ciphire client will keep a fresh copy of the Cross FPL by downloading it, or the updates, from a Ciphire server. As for the other portions of the Ciphire Fingerprint data structure, in the default mode a client will only download the portions needed to verify a certificate in question.

As part of a cross-client verification step, when a Ciphire client sends an encrypted e-mail message to another user, the latest hash in the Cross FPL is included in the e-mail. The receiving side checks that the received hash

is identical to the copy that the receiver obtained directly from a Ciphire server.

2.6 Fingerprint Trees

By stepping through the entire Fingerprint List data structure, a user of the Ciphire system could theoretically use the Fingerprint Lists from Section 2.5 to determine whether an active certificate for a given Address Identity or Certificate Identity exists. Alice might wish to do this if the Ciphire Certificate Directory claims that Bob does not have a certificate, but Alice believes that Bob does.

Because stepping through the entire Fingerprint List data structure would be too computationally expensive in practice, in addition to the Fingerprint List data structure, the Ciphire system will also maintain two Fingerprint Tree data structures, one indexed off of the certificates' Address Identities, and one indexed off of the certificates' Certificate Identities. (Fingerprint Trees are not implemented in the current version of the Ciphire system.)

For both the Address Identities and the Certificate Identities in Ciphire certificates, a hash tree is recreated after each pre-specified interval of time (e.g., one day). Each such tree is referred to as an *Interval FPT* (fingerprint tree). These trees, by definition, only reflect active certificates, which means that recreation consists of adding new certificates to the previous time interval's tree, and removing revoked certificates.

Each Interval FPT consists of a single *Master Bucket* (the root) and multiple children *Hash Buckets*. The Master Bucket contains the hash of each of its child Hash Buckets. For the Fingerprint Trees keyed off of certificates' Address Identities, each Hash Bucket contains the hash of Address Identities in currently active certificates. For the Fingerprint Trees keyed off of certificates' Certificate Identities, each Hash Bucket contains the hash of Certificate Identities in currently active certificates. The Hash Bucket that a given Address Identity or Certificate Identity is assigned to depends on the output of the hash function. For example, if the system is defined to have eight Hash Buckets per time interval, then the first three bits of the hash of the Address or Certificate Identities will be used to determine the appropriate Hash Bucket. In addition to the above-described contents, the Master and each Hash Bucket contains a carry-over hash calculated over the complete contents of the previous interval's corresponding bucket and the current interval, as well as metadata fields indicating, for example, the end time of the interval.

The Interval FPTs are chained together via a *Cross Tree List*, much like the Cross FPL from Section 2.5. Here, however, the Master Buckets for the Address and Certificate Identities take the place of the Master FPL.

2.7 Verifying Certificates

When Bob wishes to encrypt a message to Alice, he will first attempt to obtain Alice's certificate from the Ciphire Certificate Directory or a proxy. He will look up her certificate based on her e-mail address. If he is able to find such a certificate, he will first verify the self signature and the issuer signature on the certificate, as well as the certificate chain. If all the above checks pass, Bob will then verify the presence of Alice's certificate in the Fingerprint List by downloading the appropriate portions of the Fingerprint List data structure to verify the hash contained in the latest Cross List entry. If an error occurs when checking the Fingerprint List, the error is brought to the user's attention.

If the Ciphire Certificate Directory does not return a certificate for Alice, Bob will look up Alice's e-mail address in the Fingerprint Tree data structure. If Bob finds a certificate for Alice in the Fingerprint Tree, an appropriate error message is brought to the user's attention, indicating that the Ciphire Certificate Directory failed to return a certificate for Alice when a certificate for Alice really exists.

If Bob wishes to verify Alice's signature on an e-mail, the process is similar, except that Bob looks up Alice's certificate based on the Certificate Identifier included in the e-mail, not on Alice's e-mail address.

3 Compromising the Software

In an e-mail on December 23, 2004 [4], Ciphire wrote: "What we now wanted is how an attacker or us as the operators of the infrastructure would efficiently attack and compromise the infrastructure, given that the crypto is all correctly implemented and that we do not compromise the source code (which we will publish in 05)."

First, we commend Ciphire for planning to publicly release the source code for their system in 2005. As we write in [8], publicly releasing the design and source code for a security system is often a necessary step toward ensuring the security of and public confidence in a system. Unfortunately, however, although releasing the source code to a security system is often a *necessary*

condition, it is seldom a *sufficient* condition. This observation is supported simply by the large number of security bugs found in open source software systems [6], including cryptographic applications like OpenSSH [3] that were written by competent and highly security-conscious developers.

Therefore, to fully address Ciphire's question as to what types of attacks they, as the designers of the Ciphire System and as the operators, could mount, we *must* consider not just the cryptographic protocols used in the Ciphire System, but the software implementation as well. If it turns out that, regardless of the cryptographic strength of the Ciphire protocols or Fingerprint System, a Ciphire insider could still mount an attack against Ciphire users by exploiting some aspect of the software implementation, then it will be unclear how much "additional security" the Fingerprint System provides.

Unfortunately, because of the difficulty in auditing software systems for implementation-level security holes, we believe that it would be reasonable to assume that a Ciphire insider could sneak a security hole into the implementation of Ciphire clients. Moreover, we believe that, in practice, this might be the easiest and least risky way for a Ciphire insider to attack the security of the Ciphire System.

3.1 Hiding Security Holes in Software "Bugs"

While an insider might try to insert an explicit Trojan horse directly into the Ciphire Mail client — e.g., malicious code that forwards unencrypted copies of encrypted e-mails to the attacker — an important observation we make is that intentionally inserted Trojan holes do not need to be so blatant.

Intentionally inserted Trojan security holes can be made completely indistinguishable from accidentally occurring security-critical bugs. This observation is based on the fact that the insider does not need to introduce a sophisticated Trojan horse in the Ciphire Mail client to be successful. For example, the insider could still do serious damage if he or she (1) made the Ciphire Mail client vulnerable to a remotely exploitable buffer overflow (or related) attack and (2) could use the buffer overflow (or related) attack to inject malicious code into a running client.

Because a buffer overflow vulnerability could just as easily be caused by normal programmer error instead of malicious intent, the insider who inserted the buffer overflow vulnerability into the Ciphire Mail client may be able to successfully plead innocence. Furthermore, even if the hole is later patched, the insider would still have had a window of opportunity to compromise the security of users' Ciphire clients. And even if one hole is patched, there

is no guarantee that other holes do not exist, or that there might not be (intentionally placed) holes in the patch itself or in later versions of the software.

3.1.1 The Attack

We elaborate on our preferred method for installing a hard-to-detect Trojan horse into the Ciphire Mail client. We do so to support our conclusion that it will be difficult, if not impossible, to fully protect Ciphire users from attacks by a Ciphire employee or other insider.

Under our preferred method, the Ciphire insider will begin by inserting a remotely exploitable buffer overflow vulnerability, format string vulnerability, or other related vulnerability into the Ciphire Mail client software. Alternatively, the insider could begin by noticing the existence of such a vulnerability in a development version of the Ciphire Mail client, but decide not to fix the vulnerability nor bring it to the attention of anyone else.

To exploit the vulnerability, the attacker might pass a specially crafted "encrypted e-mail" or attachment to a user's Ciphire Mail client for decryption. Or the attacker might activate the exploit over some other channel, like the one between the Ciphire Mail client and the Ciphire Certificate Directory or its proxies.

Once the adversary is able to inject malicious code onto a user's computer, the adversary will have complete liberty in determining what to do with that user's data. For example, in addition to being able to forward unencrypted copies of encrypted e-mails to the adversary, the adversary might be able to install a keyboard logger and learn even more information. Or the adversary could learn the user's private decryption and signing keys. Or the adversary could force the Ciphire Mail client to encrypt messages using symmetric encryption keys with low entropy, i.e., with keys that the adversary could easily guess.

We briefly remark that the attacks described here could be mounted by anyone, once the source code for the Ciphire Mail client becomes publicly available or someone reverse engineers enough of the client from a binary to be able to find a remotely exploitable bug. But insiders would still have an easier task because they understand the details of the system better and because they were the ones to insert the remotely exploitable bugs into the software in the first place.

3.1.2 Will Opening the Source Solve the Problem?

The discussion above raises the following question: If Ciphire were to publish the source code for their e-mail client for public review, wouldn't any buffer overflow or similar security problems be discovered by the public?

Unfortunately, because of the subtlety in buffer overflow and related bugs, it may not be reasonable to assume that the public will discover the presence of the attacker-inserted security bug, at least not in a timely fashion. Furthermore, even if the attacker-inserted security bug is found, the vulnerable version may have still been used by some individuals, and therefore the security of those users' e-mails is in question. Additionally, even after the bugs are found and patched, there is little reason to believe that the patch won't have its own security problems and that there are not other remotely exploitable security bugs in other portions of the system (Section 3.2 and [6]).

We acknowledge that there have recently been important advances in the field of automated analysis of software for security problems, like [10]. Still, these recent tools are not guaranteed to discover all vulnerabilities in software applications. This means that although members of the public might run these tools against the Ciphire software, these tools may not uncover bugs inserted by the adversary.

3.1.3 Plausible Deniability

Suppose that a software application is vulnerable to a hard-to-find buffer overflow. Assuming that an outsider auditor actually discovers the existence of the buffer overflow, it would likely still be impossible for the auditor to determine whether the bug was the result of an unintentional coding mistake, or whether it was maliciously inserted by an adversary. This means that if Ciphire or an employee intentionally inserted a buffer overflow or a similar problem into the Ciphire client, Ciphire or the employee could deny any malicious intent. Such is not the case for some of the other attacks we consider in this review (e.g., the attacks in Section 4 where the adversary creates and signs, using Ciphire's private keys, two different version of the Fingerprint data structures). The plausible deniability of the attacks in this section means that an attacker bent on compromising the security of the Ciphire system could compromise the source code for the Ciphire client with relatively little fear of being caught.

3.2 Injecting Malicious Code via Software Updates

From [1] we infer that Ciphire plans to have the ability to automatically update the Ciphire client software running on users' machines. The automatic software update capability will provide another vector for an attacker to insert malicious code into the Ciphire clients, though we remark that users can turn off automatic updates.

To elaborate, we know from their e-mail on December 23, 2004 [4], that Ciphire plans to publicly release the source code for their Ciphire client application. While individual users or the general public may review the security of the publicly released source code, these reviewers may not have the time or ability to review each individual update to the Ciphire client application for vulnerabilities, and even if they do, there will likely be some latency between when the new version of the software is released and when the attacks, if any, are discovered.

3.3 On the Feasibility of Ciphire's Goal

Figuring out how to create software immune to buffer overflow and other remotely exploitable implementation-level attacks is currently an active area of research, especially when the author of the software is malicious and intentionally tries to embed (remotely exploitable) bugs into the software [9]. Indeed, we expect that one of the easiest ways for a Ciphire insider to attack the Ciphire system would be to embed a remotely exploitable but hard-to-detect bug in the Ciphire client. Given this discussion, it may not be possible for Ciphire to convince users that an adversary or Ciphire insider would not be able to use aspects of the Ciphire client software to compromise users' privacy.

3.4 Assumption for Subsequent Sections

For the remainder of this review we shall assume that there are no bugs or other problems in the implementation of the cryptographic or other portions of the Ciphire System. In practice, we might assume that each Ciphire user wrote their own Ciphire client, based on the Ciphire specification, and that if there are remotely-exploitable bugs in the software implementation, Ciphire does not know what they are and therefore will have a hard time exploiting them.

4 Attacking the Clients' "Views"

The design of Ciphire's Fingerprint Lists (FPLs) and Fingerprint Trees (FPTs) is based on sound cryptographic principles. Indeed, the use of hash chains and hash trees prevents certain classes of attacks (Section 4.2). Unfortunately, however, for the hash chains to be the most effective, we would require that all parties be given *the same* Cross FPL and Cross Tree List entries.

Unfortunately, in the Ciphire System, it does not seem possible to *guarantee* that all parties will receive the same copy of these data structures. Specifically, although the Ciphire Mail clients are supposed to automatically download the latest Cross FPL and Cross Tree List entries (under the default mode), since we are considering attacks from within the Ciphire infrastructure, there is no reason to believe that all users will receive the same copies of the Cross FPL and Cross Tree List entries.

In an attempt to address this problem, the Ciphire clients include copies of their latest Cross FPL and Cross Tree List entries, and possible other information, in the encrypted e-mails that they exchange with other Ciphire users. In this way, it is hoped that if Alice and Bob are given different Cross FPL and Cross Tree List entries from the Ciphire servers, they will be able to detect such a situation once they begin to communicate. First, even if this were true, it is not a perfect solution since the presence of an attack might not be detected until after one user already reveals confidential information to the adversary. Second and more importantly, it is not necessarily true that Alice and Bob will be able to detect the fact that they were given different Cross FPL and Cross Tree List entries. In this section we consider how an adversary might "undetectably" exploit the fact that different users' copies of the Cross Lists are not guaranteed to be identical. ("Undetectably" is in quotes since here we are referring to the fact that the attacks will be undetectable to the detection mechanisms built in to the Ciphire clients; the presence of these attacks could be determined using out-of-band methods that do not make use of the Ciphire clients themselves; we return to this in Section 4.1.7.)

4.1 Man-in-the-Middle Attacks

When considering "man-in-the-middle" attacks against a system, it is customary to give the adversary complete control over the delivery of messages between the parties in the system [7]. For example, if Bob sends a message or ciphertext X to Alice, the adversary could look at the contents of X, delay the delivery of X, or not deliver X at all. The adversary could also

create its own messages and send them to Alice as if they were originally sent by Bob. The adversary is thus the man (or entity) in the "middle."

In the conventional model, and in the model we use, we limit the power of the adversary by assuming that it does not have access to the users' secret cryptographic keys and that the adversary is restricted to reasonable computational resources (e.g., it cannot exhaustively search users' private keys). In the model we are considering, however, the adversary does have access to Ciphire's infrastructure and secret keys.

4.1.1 A Basic Man-in-the-Middle Attack

We begin by considering the following simple scenario. Assume that Alice registers as a new user of the Ciphire System and obtains a certificate Cert_A for her public key. But assume that the adversary is in control of the Ciphire infrastructure and wants to read e-mails sent to Alice, and that the adversary has the capability of monitoring Alice's network communications. Although in our model we also give the adversary the ability to modify the communications on Alice's network, the adversary mounting the attack in this section will not need to do so.

Beginning when Alice registers, one thing that the adversary could do is to start maintaining *two* copies of the Fingerprint List and Fingerprint Tree data structures, one that it gives to Alice and one that it gives to all other users. The version that it gives Alice will contain Alice's certificate. This means that if Alice were to check her copy of the Fingerprint data structures for the presence of her certificate, she would find it, and Alice would not have any reason to believe that the server is behaving maliciously. Now suppose that the version of the Fingerprint data structures that the adversary gives other users does *not* contain a copy of Alice's certificate.

If Bob or any other user decides to send an e-mail to Alice, he will look up Alice's certificate in the Ciphire Certificate Directory, and not find it. He will then look up Alice's e-mail address in his copy of the Fingerprint Tree, and again not find it. He will therefore send the message to Alice unencrypted (or decide not to send the message at all). In this way, the Ciphire insider could learn the contents of the e-mails that other users send to Alice. Note that the cross-client exchange of entries in the Cross FPL and Cross Tree List does not protect against the attack described here since the message that Bob sends Alice is unencrypted. If Bob decides to sign his message to Alice, and if the signature is also computed over the latest Cross FPL and Cross Tree List entries, the adversary could simply remove the signature and Cross entries before delivering the message to Alice; this is unnecessary with the current design, however, since Cross FPL and Cross

Tree List entries are currently not included in unencrypted but signed e-mails.

4.1.2 Improvements to the Basic Attack

Although simple and only of limited applicability, we consider the attack in Section 4.1.1 to be strong evidence that the Ciphire Fingerprint System does not achieve its design goal of protecting against man-in-the-middle attacks. Still, one might consider the attack above to be unrealistic for several reasons. For example, if Bob thinks that his message to Alice will not be encrypted, Bob may decide not to send the message at all, in which case the adversary will learn nothing. Moreover, if Bob communicates out-of-band to Alice the fact that Alice does not have a public key (according to Bob's view of the Fingerprint data structures), then Alice and Bob may learn that an attack is taking place; much better would be to have Bob encrypt a message to Alice under a public key to which the insider knows the corresponding private key. In this section, we begin addressing these deficiencies with the attack in Section 4.1.1, and continue in Section 4.1.3 with additional improvements.

First, we provide more details about how messages are encrypted under the Ciphire System. Let M be the plaintext message. According to Section 5.4 of [2], if the message is to be signed, Bob first signs M before encrypting it. It is unclear from the description in [2] whether Bob's copies of his latest Cross FPL and Cross Tree List entries are included in the signed data or not, but we assume that they are. When encrypting the message and the signature, the Ciphire client first creates a symmetric encryption key. It then encrypts the message using that symmetric encryption key, and then signs the symmetric encryption key with Bob's private key and encrypts the symmetric encryption key under Alice's public key (Sections 5.3 and 5.5 of [2]). Bob computes a signature on the symmetric encryption key even if he does not sign the message itself. From [5] we know that Alice's Certificate Identity and public key index are signed along with the symmetric encryption key.

Returning to our improvements to the attack in Section 4.1.1, instead of giving Bob Fingerprint data structures that do not contain any certificate for Alice, the Ciphire insider could give Bob Fingerprint data structures that contain a certificate for Alice, Cert'_A. Here, however $\mathsf{Cert}'_A \neq \mathsf{Cert}_A$ and, in particular, the Ciphire insider knows the private keys corresponding to the public keys in Cert'_A. Now, when Bob sends an encrypted message to Alice, Bob will encrypt the message under the public keys in Cert'_A. The Ciphire insider will remove the encryption under the public keys in Cert'_A,

and re-encrypt the message under Alice's real public keys. During the re-encryption processes, the Ciphire insider will replace the Cross FPL and Cross Tree List entries included by Bob with copies that will be consistent with the data structures that it gave Alice.

If Bob chose to sign the message, the signature will most likely be computed over Bob's Cross FPL and Cross Tree List entries, as well as over the message itself. In this case, since the Ciphire insider will not be able to change the Cross List entries without voiding Bob's signature, the Ciphire insider will simply remove the signature from the message and forward the re-encrypted but unsigned message to Alice.

Recall that, regardless of whether Bob signs the message, Bob will still sign the symmetric key used during the encryption process, Alice's Certificate Identity, and the index of the public key used in the encryption. Note that the Ciphire insider does not need to change this symmetric key when re-encrypting the message for Alice. Also note that if the Ciphire insider sets the Certificate Identity in Cert'_A to be equal to the Certificate Identity in Cert_A, the insider will not need to change Certificate Identity that Bob signed along with the symmetric key. This means that the Ciphire insider will not need to forge a signature on the symmetric encryption key and Alice's Certificate Identity, and can therefore simply include the original signature in the re-encrypted e-mail that it delivers to Alice. (If Ciphire protocol also mandated that Bob sign the fingerprint of Alice's certificate along with the symmetric encryption key, rather than the Certificate Identity and public key index, then the Ciphire insider would have to forge a new signature in order to prevent Alice's Ciphire Mail client from issuing a warning to Alice.)

4.1.3 Continued Improvements

Although the attack in Section 4.1.2 is an improvement over the attack in Section 4.1.1, and although the attack does provide even stronger evidence for our claim that the Ciphire System is vulnerable to man-in-the-middle attacks, the attack in Section 4.1.2 does suffer from at least one practical problem: If Alice ever replies to an e-mail from Bob, without further chicanery on the part of the adversary, the structure of Alice's reply will tip Bob off to the fact that a man-in-the-middle attack has taken place. Instead of delivering Alice's replies to Bob, the adversary could simply drop all of Alice's replies, but that would likely also alert Bob to the presence of the adversary.

There are several reasons why the adversary cannot just deliver Alice's reply e-mails directly to Bob. First, Alice's reply e-mail will contain copies of

Alice's latest Cross FPL and Cross Tree List entries, which will be different from Bob's. Second, even if Alice does not sign her reply message to Bob, the symmetric key used in the encryption process will be signed by Alice's real private key, not the private key corresponding to the certificate Cert'_A given to the other users of the Ciphire System. Because of this situation, if the adversary were just to implement the attack described in Section 4.1.2, Bob might become wise to the fact that a man-in-the-middle attack was taking place because either (1) he stopped receiving encrypted e-mails from Alice or (2) the e-mails he received from Alice have different Cross FPL and Cross Tree List entries and "invalid" signatures (with respect to the public keys in Cert'_A) on the symmetric encryption keys. We address these concerns here.

For simplicity, in this subsection let us assume that there are only two users of the Ciphire system, Alice and Bob; we will add additional users in Section 4.1.4. In this attack, the Ciphire insider will present Alice with Fingerprint List and Tree data structures that contains Alice's real certificate, Cert_A, and a fake certificate for Bob, Cert'_B. Similarly, the Ciphire insider will present Bob with Fingerprint List and Tree data structures that contain Bob's real certificate, Cert_B, and a fake certificate for Alice, Cert'_A. As before, we assume that the insider knows the private keys corresponding to the public keys in the fake certificates Cert'_A and Cert'_B.

Now suppose that Bob wishes to send a message M to Alice. Bob's Ciphire client will look up Alice's certificate, find Cert'_A, and then (possibly) sign the message to Alice using Bob's private keys, and then encrypt the message using the public keys in Cert'_A. The adversary will intercept this e-mail and decrypt it using the private keys corresponding to the public keys in Cert'_A. The adversary will then take the plaintext message, re-sign it and the symmetric encryption key using the private keys corresponding to the fake certificate Cert'_B, encrypt the message under Alice's real public keys in Cert_A, and send the resulting ciphertext to Alice. During the re-signing process, the adversary will replace the latest Cross FPL and Cross Tree List entries that Bob sent with entries that correspond to Alice's latest entries.

Because of the symmetry of the data structures (Alice is given Cert_A and Cert'_B, while Bob is given Cert'_A and Cert_B), the Ciphire insider can also mount this man-in-the-middle attack when Alice sends encrypted replies to Bob. Thus, Alice and Bob have complete liberty to use the Ciphire System as normal, but the adversary will be able to read all of their communications without triggering any detection mechanism built into the Ciphire clients.

4.1.4 The Attack With More Users

In Section 4.1.3, we assumed for simplicity that Alice and Bob were the only two users of the Ciphire System. But now suppose that there are additional users of the Ciphire System, such as Charlie, Diane, and Ethan.

Let us assume that the adversary is interested in reading all of the e-mails to and from Alice, but that the adversary is not interested in reading the e-mails of the other Ciphire users. Then we can modify the attack in Section 4.1.3 as follows. The Ciphire insider will include the fake certificate for Alice, Cert'_A, in the Fingerprint data structures that it gives to all of the users of the Ciphire System, except Alice. In this way Bob, Charlie, Diane, Ethan, and others, will all have the same Cross FPL and Cross Tree Lists, and all these other users will be able to communicate with each other as normal; most importantly, the adversary would not need to be able to mount a man-in-the-middle attack on or modify any of the communications between the other parties in order to change the values of the Cross FPL and Cross Tree List entries included in the e-mails. As for the Fingerprint data structures that the adversary gives to Alice, the Ciphire insider will create fake certificates for all of the other users, and include these fake certificates in the Fingerprint data structures that it gives to Alice.

Since we are assuming that the adversary is attacking Alice and is capable of mounting man-in-the-middle attacks, the adversary will be able to intercept and modify all of the e-mails that Alice sends and receives. Thus, when any other user sends a (possibly signed and) encrypted message to Alice, the attacker will be able to strip off the encryption, read the message, modify the included Cross FPL and Cross Tree List entries, re-sign the message if necessary, re-encrypt the message under Alice's real public key, and re-sign the symmetric encryption key using the fake certificate for the sender. The adversary can similarly read the contents of and adjust the Cross FPL and Cross Tree List entries and the signature on the symmetric encryption key in the encrypted e-mails from Alice to the other users of the Ciphire System.

4.1.5 Starting These Attacks After Alice Enrolls

The attacks in Sections 4.1.1 through 4.1.4 work when the adversary or Ciphire insiders knows that they wishes to mount a man-in-the-middle attack against a user when that user first enrolls in the Ciphire System. This allows the adversary to give Cert'_A to all of the other users of the Ciphire System without having to revoke any previous certificate for Alice. And this allows the adversary to include fake certificates for the other users in the Fingerprint data structures that it gives to Alice without having to revoke any of the other users' certificates.

Now suppose that the Ciphire insider decides that he or she wishes to mount a man-in-the-middle attack against Alice, but Alice has already enrolled in the Ciphire System and her real certificate, Cert_A, is already reflected in the Cross FPL and Cross Tree List downloaded by other users of the Ciphire System. Also assume that Cert_A is not set to expire for quite some time.

If no other user has already communicated with Alice, then no user will actually have a copy of Cert_A, and Alice would not have a copy of the certificates for any other user. In this case, the insider could just issue fake certificates for Alice and the other users as in Section 4.1.4. Since there does not appear to be a mechanism within the Ciphire System for Ciphire clients to look up *all* of the certificates ever issued for a given Address Identifier, the new fake certificates will override the existence of the real certificates, and the adversary will be able to read Alice's communications, as described in Section 4.1.4.

The attack becomes more complicated if Bob already has a copy of Alice's real certificate Cert_A, and if Cert_A has not already expired. If this were the case, and because Bob's client application is supposed to check the self-signature on the revocation certificate for Cert_A before accepting the fake new certificate Cert'_A, then the adversary would have to create a valid revocation certificate for Cert_A. Recall that the self-signature on a revocation certificate is a signature generated by the private keys corresponding to the public keys in the certificate being revoked. See Section 8 for how an insider might attempt to forge a valid revocation certificate for Cert_A, perhaps by exhaustively searching the decryption key for the emergency certificate revocation messages stored on a Ciphire server.

In the first step of the attack, the adversary would revoke Alice's certificate, create a new certificate Cert'_A, and include it in the fingerprint data structures that it gives to all of the users of the Ciphire System besides Alice.

Now suppose that Bob looks up Alice's certificate in the Ciphire Certificate Directory, or a proxy. Here we assume that the Ciphire insider also has insider access to the proxy server, which may not be the case if the proxy server is run by an independent third party. When Bob does this, the adversary will learn that Bob is planning on sending an encrypted e-mail to Alice. If the adversary does not expect Alice to ever reply to Bob's e-mail, the adversary could just give Bob Cert'_A, and then decrypt and re-encrypt Bob's ciphertext as in Section 4.1.2.

If the adversary anticipates that Alice will reply to Bob's e-mail, and if Alice already has a copy of Bob's real certificate Cert_B, then the adversary will need to create a revocation certificate for Cert_B and a new bogus certificate

Cert'_B for Bob, as in Section 4.1.3. If the adversary is able to do this, it will give Alice new Fingerprint data structures with the revocation certificate for Cert_B and the new certificate Cert'_B. If the adversary is unable to create a revocation certificate for Bob, perhaps because the adversary cannot exhaustively search the key for Bob's emergency revocation certificate, it might be risky for the adversary to mount a man-in-the-middle attack on the communications from Bob to Alice since the adversary would not be able to mount a man-in-the-middle attack on Alice's reply. In such a situation, the adversary could just have the Ciphire Certificate Directory return Cert_A to Bob instead of Cert'_A, and Bob would encrypt his message under Alice's real public key.

In the above paragraph, we have the Ciphire insider returning either Cert_A or Cert'_A to Bob, depending on whether the adversary can create a valid revocation certificate for Cert_B. There are several reasons why we can do this. First, Bob's Fingerprint List data structure will contain both Cert_A and Cert'_A, so when Bob tries to verify the certificate he receives from the Ciphire Certificate Directory, the certificate verification procedure will succeed. During the verification procedure, Bob's client application will not access the Ciphire Fingerprint Tree data structure. Second, from the documentation we received, there does not currently seem to be a built-in mechanism for Bob to verify that Alice has only one active certificate, though such a mechanism could be created using the Fingerprint Trees. It is also unclear whether there exists a mechanism for Bob to look for the existence of a revocation certificate for Cert_A, especially if the Ciphire insider does not wish for Bob to be able to find that revocation certificate. Third, even if Alice's client has mechanisms to detect extraneous revocation certificates or regular certificates for *her* e-mail address in her copy of the Fingerprint List data structure, Alice's copy of the Fingerprint data structures will contain neither the revocation certificate for Cert_A nor the new certificate Cert'_A.

4.1.6 Attacking Authenticity

Although the discussions above concentrate on the methods that a Ciphire insider might use to compromise the privacy of two Ciphire users' communications, we remark here that our attacks could also be used to compromise the authenticity of those user's communications. For example, before the adversary re-signs a message from Bob to Alice with the private keys corresponding to the certificate Cert'_B, the adversary could change the contents of the message. By doing this, the Ciphire insider could convince Alice that Bob wrote something in his e-mail when in fact Bob did not.

4.1.7 Detecting These Attacks

In the preceding sections, we observed that the attacks would not be detectable by mechanisms built into the Ciphire clients. Still, the attacks could be detectable using some out-of-band communications mechanisms. For example, if Alice and Bob communicate the fingerprints of their certificates to each other over another channel, such as the telephone, they will quickly learn that they were given the wrong certificates, and therefore that a man-in-the-middle attack might take or have taken place. Unfortunately, requiring such out-of-band communications greatly reduces the transparency and usability of the current Ciphire System.

The above attacks, once mounted, also require the adversary to continuously mount the man-in-the-middle attack afterwards. Otherwise, Alice will get messages encrypted to the wrong public key, Alice will send messages to others encrypted under the wrong public key, or the hashes in the Fingerprint Lists will not verify. If the adversary ever stops mounting the man-in-the-middle attack, Alice or the party that she is communicating with will learn that a man-in-the-middle attack has taken place in the past.

4.1.8 On the Severity of These Attacks

The question now arises: how serious are the man-in-the-middle attacks that we uncover above?

From a theoretical perspective, the attacks are quite damaging because they prove that a Ciphire insider could mount a man-in-the-middle attack against a Ciphire user, and thereby read (or modify) all of that user's communications. Further, the attack will be undetectable to any detection mechanism built into the Ciphire clients.

In the real world, however, we recognize that it would likely be unreasonable to assume that, once the adversary begins a man-in-the-middle attack, that the adversary will be able to continuously mount the man-in-the-middle attack on all of the user's subsequent communications. Thus, we could expect the attacked user to eventually realize that an attack has taken place, and to raise the appropriate alarms.

The threat of eventual detection will likely be enough of a deterrent for the Ciphire insider to not try to mount the attacks in the section, but to instead mount an attack from, say, Section 3 or Section 7.

4.2 Attempted Improvements: Attacking the Fingerprint Data Structures Directly

It would be tempting to try to modify the Fingerprint data structures so that the copy Alice receives does not include Cert'_A, the copy Bob and others receive includes Cert'_A, and both Alice's copy and the other copies have the same Cross FPL entries. If the adversary were able to do this, the adversary's task in a man-in-the-middle attack would be simplified because the adversary would not have to reconcile Alice's and Bob's differing views of the latest Cross FPL and Cross Tree List entries. (Here we are assuming that Alice will check her copy of the Fingerprint List data structures for rouge certificates with her e-mail address, which happens under a certain configuration for Alice's Ciphire Mail client. Otherwise, the adversary could just include Cert'_A in the Fingerprint List data structures given to everyone, including Alice; see Section 5 for related discussion.)

However, by judicious use of hash trees and hash chains in the Ciphire Fingerprint System, the above goal does not seem to be possible. Rather, if the adversary were able to create two Fingerprint data structures with different contents but the same Cross List entries, it seems that an adversary would have found a collision for the underlying hash function. Since we consider finding collisions against SHA-256 to be unrealistic, it is reasonable to conclude that the adversary will not be able to create such data structures.

4.3 Assumption for Subsequent Sections

In addition to the assumption made in Section 3.4, for the remainder of this review we shall assume that all users have access to the same Fingerprint List and Fingerprint Tree data structures.

5 Multiple Active Certificates for a Single Address Identity

In Section 4.2 we observe that the structure of the Ciphire Fingerprint List data structure should prevent an adversary from creating two fingerprint data structures with the same Cross List entries but with different contents (one containing Cert'_A, the other not).

In this section, we ask the question: What could a Ciphire insider accomplish if he or she inserted the fake certificate Cert'_A into the Fingerprint List given to all parties, including Alice? Under the default download mode (page 12 of [1]), Alice would not detect the presence of Cert'_A since her client would only download the portions of the Fingerprint List data structure necessary to verify the certificate that the client is dealing with. This means that Alice would likely not notice the introduction of Cert'_A.

Having done this, the adversary could mount a man-in-the-middle attack by giving Cert'_A to Bob, decrypting Bob's message to Alice using the private keys for Cert'_A, and then re-encrypting the message under the public keys in Cert_A. Since even if Bob does not sign his message to Alice, he will still sign the symmetric key used in the encryption and the Certificate Identity of the certificate that he used for Alice (Cert'_A), the adversary must also insert a fake certificate Cert'_B for Bob into the Fingerprint List data structures, and give Cert'_B to Alice as Bob's certificate. The adversary would then re-sign the symmetric encryption key and the Certificate Identity for Cert_A with the private keys corresponding to Cert'_B.

If Alice or Bob checked for a certificate for Alice in the Fingerprint Tree data structure, then under the above construction, they would find two active certificates for Alice. Similarly for the certificates for Bob. Thus, instead of having Alice herself check all the entries in her branch of the fingerprint list for a fake certificate Cert'_A, one way to protect against the attack above would be to have all users check for the uniqueness of their and their peers' certificates in the Fingerprint Tree data structure. (Although these checks were not described in the documents that we reviewed, e.g., the verification scenarios on page 18 of [1], we understand that Ciphire now plans to include such checks in their Ciphire Mail client.)

6 Creating Certificates for Users Not Enrolled in the System

On page 27 of [2], Housley and Ferguson remark "The Ciphire CA may just create a certificate for an arbitrary e-mail address if no active certificate exists for that address. In combination with a man-in-the-middle attack, this bogus certificate could trick a user into sending data he would have never without encryption. The threat is minor. Most importantly, the owner of the e-mail address will find out about the bogus certificate if he tries to create a legitimate certificate."

We agree with Housley and Ferguson that the threat exists. However, without further context about who the users in question are, and the situation, we would be loath to claim without qualification that the risk is minor. Indeed, one could envision situations where this attack could be serious.

There are at least three properties of the attack described here that make it both practical and potentially a serious risk. First, it would be unreasonable to assume that all users will eventually try to obtain a legitimate certificate, which means that the condition "if he tries to create a legitimate certificate" in the last sentence of the quote may not always mean much in practice. Second, even if the legitimate user subsequently attempts to obtain a legitimate certificate, the fact remains that the adversary could have read e-mails already encrypted to the user under the adversary's chosen public keys; i.e., the adversary would already have succeeded in compromising the privacy of the user. Third, the attack described above could be mounted by an outsider instead of by a Ciphire employee, although the setup might not be as simple. For example, Alice's systems administrator might request and obtain a certificate for Alice. This means that if a Ciphire employee were to mount the attack described above, and if Alice did indeed later attempt to obtain a legitimate certificate for herself, Ciphire could claim that the attack was not mounted by a Ciphire insider. Thus, Alice may not realize that her security was compromised from within Ciphire itself.

The discussion above raises the following question: How does the Ciphire System handle the situation where the real owner of an e-mail address cannot get a certificate because someone else has already (maliciously) obtained a certificate for that address, or because that address once belonged to someone else who does not wish to revoke his or her Ciphire keys (e.g., the CFO of a company quits, but does not wish to revoke his or her keys for the e-mail address `cfo@company.com`). Although the documents [1, 2] do not discuss any appropriate mechanism, from [5] we know that Ciphire does have an Abuse Revocation mechanism for extreme situations where a self-signed revocation certificate cannot be created. We discuss the security implications of the Abuse Revocation mechanism in Section 8.

7 Creating Certificates for Related E-mail Addresses

Related to the attack in Section 6, we could consider an attack in which a Ciphire insider creates certificates for *variants* of a legitimate user's e-mail address. For example, assume that Alice's e-mail address is `alice@company.`

com. Now suppose that the Ciphire employee registers certificates for the e-mail addresses alice@compny.com ("company" intentionally misspelled), alcie@company.com ("Alice" intentionally misspelled), and alice@company.org (domain name intentionally ".org" instead of ".com"). Then if Bob were to try to send an encrypted message to Alice but accidentally mistyped her e-mail address, the e-mail would be encrypted to a public key that the adversary controlled. This attack may be very effective for an adversary because of the transparency and ease-of-use of the Ciphire System — the fact that the Ciphire Mail client automatically requests certificates for e-mail addresses means that Bob may never realize that he mistyped Alice's e-mail address.

Also note that Alice may have multiple valid e-mail addresses at her company; e.g., alice@company.com, ali@company.com, Alice.Adams@company.com, Adams.Alice@company.com, Alice_Adams@company.com, and Alice.B.Adams@company.com might all reach Alice. Furthermore, Alice's company might have multiple domain names or hostnames, and Alice might also be reachable at alice@companytwo.com and alice@companythree.com. Unless Alice obtains certificates for all these e-mail addresses, a Ciphire insider could easily obtain certificates for these e-mail addresses. Since these are all legitimate e-mail addresses for Alice, someone might encrypt a message to Alice under the public key corresponding to one of these e-mail addresses for which the adversary knows the private keys. In the current version of the Ciphire Mail client, one way an adversary might trick the sender into doing this is by adding an appropriate Reply-To address to e-mails from Alice, assuming that the encrypted e-mails from Alice do not already contain a Reply-To address. The adversary could then easily read the contents of those private e-mails. (We remark that this latter trick, of adding bogus Reply-To addresses to encrypted e-mails from Alice, will not work if Alice already includes a Reply-To in her encrypted e-mails. This is because Alice's real Reply-To will be encrypted and MACed along with the body of her message.)

As with the attack in Section 6, this attack could also be mounted by an outsider, though not in all cases as easily. Nevertheless, knowing that he or she will not necessarily be implicated as the party responsible for registering the bogus certificates, the Ciphire insider might feel more comfortable mounting these attacks than some of the other attacks in this review.

8 Creating Revocation Requests

In some cases a Ciphire insider may wish to issue revocation certificates for a user unbeknown to or against that user's wishes. For example, see the attack in Section 4.1.5. As with the attack in Section 4.1.5, we assume that the insider has some mechanism to prevent the user from learning that his or her certificate has been revoked, as would be necessary if the user's client is configured to download the complete contents of his or her branch of the Fingerprint Lists (which is not the default mode).

One approach the Ciphire insider might take is the following. Note that although the normal procedure for revoking a certificate is interactive and assumes that the user is currently in possession of his or her private key, the Ciphire System does provide a mechanism for users to revoke their certificate if they have lost access to their private key. This uses the Ciphire emergency revocation system, as described in Section 2.4.2.

The critical observation is that many of the emergency revocation requests may be encrypted under easily guessable passphrases. When this is the case, the Ciphire insider with access to the database storing the encrypted emergency revocation requests could issue emergency revocation messages for the user in question. From [5] we know that future versions of the Ciphire System will allow a user to choose not to store his or her encrypted emergency revocation certificate on the Ciphire server, but to instead store it on his or her own computer. Although this will prevent the attack described earlier in this paragraph, it would seem reasonable to assume that users that choose this option will also be the same users that would choose hard-to-guess passphrases. Therefore, the users with easily guessable passphrases might still be susceptible to the attack described above.

Another approach for issuing revocation certificates for a user against that user's will might be to try to trick the user into signing a revocation certificate instead of an e-mail message; e.g., the adversary might construct a revocation certificate, and then send it to the user in the hopes that the user would forward the certificate to another user and, in the process, sign the certificate. Fortunately, from [5] it seems that this attack will not work. Namely, it appears that revocation certificates and encrypted e-mails have different structures, and therefore a signature on an e-mail or its attachments cannot be mistaken as a signature on a certificate. If this were not the case, we would suggest making sure that the certificate and e-mail message spaces are disjoint, e.g., by prepending an appropriate tag to the certificates and e-mails before signing them.

There may be cases when certificates for e-mail addresses need to be revoked, but the standard revocation procedure, which involves using self-signed certificates, cannot be invoked. For example, consider the case when an attacker creates a certificate for Charlie's e-mail address before Charlie has a chance to. When Charlie later attempts to create a certificate for himself, he will be unable to do so because a certificate already exists for his e-mail address. Since we can expect the attacker not to willingly participate in the standard revocation procedure, the Ciphire System needs a way to revoke the attacker-created certificate for Charlie's e-mail address without having a valid self-signed certificate revocation request. From [5] we know that the Ciphire has an "Abuse Revocation" mechanism designed for such a situation.

Conceivably, an insider could use the Abuse Revocation mechanism to maliciously revoke the certificates of normal users. Doing so may not, however, be an attractive option to the adversary since, when Bob's Ciphire Mail client finds that Alice's certificate has been revoked using the Abuse Revocation mechanism, a suitable warning will be displayed to Bob's computer. It Bob were to communicate this fact to Alice using some out-of-bands mechanism, both he and Alice would learn that an attack has taken place.

An outsider could pretend to be Alice and try to trick Ciphire into using the Abuse Revocation mechanism to revoke a legitimate certificate for Alice. Since in this case Ciphire is not intentionally trying to attack Alice, Ciphire will give Alice access to the fingerprint data structures that contain the revocation certificate. If Alice's Ciphire Mail client is configured to download the entire contents of her branch of the Fingerprint List, then Alice will detect this attack.

9 Damaging Ciphire's Reputation

A stated purpose of the Ciphire Fingerprint System is to make the Ciphire System more auditable to outside parties, including Ciphire users. For example, the hope is that if a Ciphire insider or someone else with access to the Ciphire secret keys were to do something mischievous such as create fake certificates, the users of the Ciphire System would be able to recognize such an attack by checking the contents of the Ciphire Fingerprint data structures.

We discussed the efficacy of the Ciphire Fingerprint System at achieving the above goal in other sections of this analysis. In this section, we ask if

the Ciphire Fingerprint System could actually have negative side effects on the security of the Ciphire System, or on Ciphire's business operations.

For example, by making the Ciphire System more auditable, it could be the case that an outsider, without access to Ciphire's private keys, could still trigger the Ciphire System's audit warnings. For example, an outsider could try to make all Ciphire Mail clients output a warning message that would *imply* that the Ciphire private keys have been leaked and that there may be security problems with the Ciphire System. If an outsider could mount such an attack, he could make many Ciphire users suspicious of the Ciphire System, and thereby drive away potential customers. (Certainly an insider with access to the Ciphire private keys will be able to trigger the audit warnings on the Ciphire clients and spoil Ciphire's reputation, so we do not consider attacks by the insider here.)

Fortunately, and as one might expect, because an outsider does not have access to the Ciphire private keys, an outsider should not be able to create a valid certificate (signed by the Ciphire private keys) that is not included in the Ciphire Fingerprint List. Therefore, assuming no attacks from within Ciphire, if the Ciphire Certificate Directory or a proxy returns a signed certificate to a user, that user will be able to find that certificate in the Ciphire Fingerprint List, and that user's Ciphire client will not display a warning to the user. Furthermore, if we consider the operators of the Ciphire Certificate Directory proxy servers to be outsiders, then these outsiders will not be able to pretend that certificates for certain e-mail addresses do not exist in the Ciphire Certificate Directory. If the proxies could do this, they could cause problems since the Fingerprint Tree data structure would reveal the existence of certificates for those e-mail addresses. The proxies cannot deny the existence of certificates because all responses from the Ciphire Certificate Directory to the proxies are signed and contain the complete contents of the original lookup, including the Address or Certificate Identity of the lookup, and a timestamp. Thus, if a proxy informs a user that no certificate exists for a given e-mail address, the user would expect a signed message from the Ciphire Certificate Directory indicating that this is true. An outsider could also try to have bogus revocation certificates issued for users, but the approaches discussed in Section 8 either require insider access or, in the case of an outsider trying to have an Abuse Revocation certificate issued for another user, are not scalable.

Bibliography

[1] Ciphire Labs. Ciphire fingerprint system technical description, December 2004. `fingerprint-system-2004-12-02.pdf`.

[2] Russ Housley and Niels Ferguson. Security design review of the Ciphire system, July 2004. Available at `http://www.ciphire.com/cm/technology/reviews.html`.

[3] ISS X-Force. Internet Security Systems security advisory: OpenSSH remote challenge vulnerability, June 2002. Available online at `http://bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=2058%4`.

[4] Errikos Pitsos, December 2004. Personal e-mail.

[5] Errikos Pitsos, January 2005. Personal e-mail.

[6] Eric Rescorla. Is finding security holes a good idea? In *Workshop on Economics and Information Security*, May 2004.

[7] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 1996.

[8] Bruce Schneier. Open source and security. *Crypto-Gram Newsletter*, September 1999. Available online at `http://www.schneier.com/crypto-gram-9909.html#OpenSourceandSecurity`.

[9] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763, August 1984.

[10] David Wagner, Jeffrey S. Foster, Eric A. Brewer, , and Alexander Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In *Network and Distributed System Security Symposium*, 2000.

