Protocol Failure in the Escrowed Encryption Standard

Matt Blaze
AT&T Bell Laboratories
mab@research.att.com

August 20, 1994

Abstract

The Escrowed Encryption Standard (EES) defines a US Government family of cryptographic processors, popularly known as "Clipper" chips, intended to protect unclassified government and private-sector communications and data. A basic feature of key setup between pairs of EES processors involves the exchange of a "Law Enforcement Access Field" (LEAF) that contains an encrypted copy of the current session key. The LEAF is intended to facilitate government access to the cleartext of data encrypted under the system. Several aspects of the design of the EES, which employs a classified cipher algorithm and tamper-resistant hardware, attempt to make it infeasible to deploy the system without transmitting the LEAF. We evaluated the publicly released aspects of the EES protocols as well as a prototype version of a PCMCIA-based EES device. This paper outlines various techniques that enable cryptographic communication among EES processors without transmission of the valid LEAF. We identify two classes of techniques. The simplest allow communication only between pairs of "rogue" parties. The second, more complex methods permit rogue applications to take unilateral action to interoperate with legal EES users. We conclude with techniques that could make the fielded EES architecture more robust against these failures.

1 Introduction and Background

In April 1993, the Clinton Administration announced a proposed new federal standard symmetric-key encryption system for the protection of sensitive-but-unclassified government and civilian data [Mar93]. The proposal, called the Escrowed Encryption Standard (EES) [NIST94], includes several unusual features that have been the subject of considerable debate and controversy. The EES cipher algorithm, called "Skipjack", is itself classified, and implementations of the cipher are available to the private sector only within tamper-resistant modules supplied by government-approved vendors. Software implementations of the cipher will not be possible. Although Skipjack, which was designed by the US National Security Agency (NSA), was reviewed by a small panel of civilian experts who were granted access to the algorithm, the cipher cannot be subjected to the degree of civilian scrutiny ordinarily given to new encryption systems.

By far the most controversial aspect of the EES system, however, is *key escrow*. As part of the crypto-synchronization process, EES devices generate and exchange a "Law Enforcement Access Field" (LEAF). This field contains a copy of the current session key and is intended to enable a government eavesdropper to recover the cleartext. The LEAF copy of the session key is encrypted with a device-unique key called the "unit key", assigned at the time the EES device is manufactured. Copies of the unit keys for all EES devices are to be held in "escrow" jointly by two federal agencies that will be charged with releasing the keys to law enforcement under certain conditions.

At present, two EES devices are being produced. The simplest, the Clipper chip (also known as the MYK-78), is essentially a drop-in replacement for a conventional DES [NBS77] chip and relies on key negotiation being handled off the chip. The other EES device, the Capstone chip (MYK-80), adds built-in support for public-key negotiation and digital signatures, with modular arithmetic functions, random number generation, and other such features.

The interface to the Skipjack cipher is similar to that of DES, based on a 64 bit codebook block cipher and supporting FIPS-81 [NBS80] standard modes of operation. Keys are 80 bits in length, as opposed to DES's 56 bits.

The initial application of EES is in stand-alone voice encryption telephone units, such as the AT&T

© 1994. This is a pre-print of a paper to appear at the 2nd ACM Conference on Computer and Communications Security, Fairfax, VA, November 1994.

Model 3600 Telephone Security Device. To facilitate computer applications such as electronic mail and file encryption, a version of the Capstone chip will also be available packaged in a standard PCMCIA card. EES PCMCIA cards can be installed easily in many commercially available laptop computers, and SCSI-based PCMCIA card readers can connect EES cards to most other computers. The government has specified a standard application interface library for communicating with the cards.

Clipper and Capstone chips are, at present, available only for use in approved products that comply with LEAF handling requirements. EES PCMCIA cards, on the other hand, are themselves a stand-alone product, and are to be made generally available "off the shelf" in the United States.

The government has stated that the goal of the EES is to make a strong cipher available for legitimate use without supplying criminals and other adversaries with a tool that can be used against American interests or to hide illegal activities from law enforcement. Thus the system is intended to be difficult to deploy without also sending a valid LEAF and thereby exposing the traffic to the possibility of government monitoring. In this paper, however, we show that it is possible to construct applications that can enjoy use of the Skipjack cipher but that do not admit law enforcement access through the LEAF. For the purposes of this paper, we consider two classes of "rogue" EES applications: those that can communicate only with other rogue systems and those that can successfully interoperate with EES "legal" systems as well. The latter category especially threatens the goals of the EES program, since such rogue applications would be operationally equivalent to their legal counterparts without being subject to government access.

1.1 LEAF Structure and Protocols

The LEAF is a 128 bit structure containing enough information for law enforcement recovery of the session key with the cooperation of the two agencies holding the unit key database. The structure contains a 32 bit unique unit identifier (the serial number of the chip that generated the LEAF), the current 80 bit session key (encrypted with the device's unit key) and a 16 bit LEAF checksum. The entire structure is encrypted with a fixed "family key" to produce the final LEAF message. All cryptographic operations employ symmetric (secret) key techniques. The family key is shared by all interoperable EES devices. The family key, the encryption modes used to encrypt the unit key and the LEAF message, and the details of the checksum are all secret. Externally, the LEAF is an opaque 128 bit package. See Figure 1.

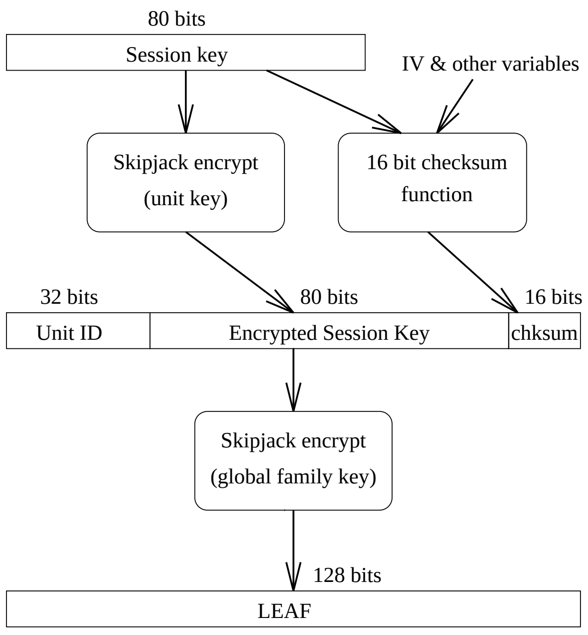

Figure 1: LEAF Structure

To decrypt EES traffic, a law enforcement agency first must intercept the LEAF and the traffic itself using conventional data wiretapping technology. The LEAF is decrypted with the family key, revealing the chip serial number, the unit key-encrypted session key and the LEAF checksum. The chip serial number is provided, with appropriate authorization, to the two escrow agencies, which each return half of the unit key for the given serial number. The two half-unit keys can be combined (by bitwise exclusive-or) to produce the unit key, which the law enforcement agency can then use to decrypt the session key. This session key can then be used to decrypt the actual traffic.

The wiretapping system thus relies on the availability of the LEAF along with the encrypted traffic. To force applications to send the LEAF on the same channel as the traffic, EES devices will not decrypt data until they have received a valid LEAF for the current session key. Presumably, EES devices perform various integrity checks on received LEAFs prior to accepting them.

To provide a convenient application interface for LEAF management, EES devices generate and load LEAFs along with the FIPS-81 initialization vectors (IVs). The devices provide "generate IV" and "load IV" functions that operate on 192 bit fields containing an unencrypted 64 bit IV concatenated with the 128 bit encrypted LEAF. The load IV operation fails if the associated LEAF does not pass an integrity check.

1.2 Experimental Observations

Most details of the LEAF creation method, encryption modes, and data structures, beyond those mentioned above, are classified and are therefore unknown to us. In particular, the EES standard does not specify the exact mechanism that enforces the transmission of the correct LEAF. However, we were able to perform a number of simple experiments on our prototype devices to confirm and expand our knowledge of LEAF internals. All experiments were performed at the protocol level through the standard interface and did not involve cryptanalysis or direct hardware "reverse engineering." We summarize our observations below.

- LEAF integrity is verified entirely via redundancy in the checksum field. In general, attempts to load an incorrect LEAF fail. This must be due entirely to the checksum field and not through direct verification of the unit ID or encrypted session key; the receiving chip cannot confirm the correctness of the unit ID or encrypted session key fields since it does not know the unit ID or unit key of the sender. Therefore, the LEAF must be testable by the receiver based only on known information (such as the cleartext session key and IV) included in the checksum computation.

- LEAF checksum computation includes (implicitly or explicitly) the current IV. The LEAF changes whenever a new IV is generated for a given session key. Since the IV is not included directly as one of the LEAF fields, it must influence the checksum. Furthermore, the receiving device refuses to load the wrong IV for a given LEAF.

- LEAF checksum computation includes the cleartext of the current session key. Attempts to load a LEAF (and corresponding IV) from a previous session key fail. It is therefore not possible to "reuse" a LEAF generated from an old session key, even though the LEAF itself appears internally consistent.

- LEAF checksum computation includes other parts of the LEAF. Attempts to load LEAFs with a single bit inverted anywhere in the 128 bit structure fail.

- LEAF encryption diffuses its input throughout the entire 128 bit structure. The LEAF structure or encryption mode is not exactly as specified in released documents. Generating a new IV for a given session key causes changes across the entire LEAF. Recall that the EES codebook size is 64 bits, and so encryption of the LEAF involves at least two block encryptions. Since the IV affects only the checksum, and the checksum appears at the end of the LEAF in public documents, we can conclude that at least one of the following is true:
  - The LEAF is encrypted with a non-standard mode in which cleartext in "late" blocks affects the early ciphertext.
  - The LEAF is encrypted with a standard forward-chaining or stream mode but the checksum appears in the first cipherblock of the LEAF.
  - The LEAF is encrypted with a standard forward-chaining or stream mode but the current session IV is itself used to initialize it.

- The LEAF checksum is, in fact, 16 bits. A brute-force search of the LEAF space for a valid LEAF requires about 2^{16} operations. See the discussion of interoperable rogue applications below.

2 Non-interoperable Rogue Applications

First, we consider the problem of constructing a set of applications that use Skipjack to communicate among themselves without key escrow. We are free to use any method permitted by the EES processor interface without regard for standard usage. Since such applications may be restricted to communicating with other rogue systems, their general utility is somewhat limited, although they still violate the intent of the EES.

Several approaches can easily circumvent the law enforcement access mechanism, with a range of practicality and tradeoffs.

2.1 LEAF Obscuring

The simplest approach is to take steps to ensure that the eavesdropper cannot recover the LEAF or the encrypted traffic. One option is pre- or post- encryption of the traffic with another cipher. It is not clear, however, what the attacker gains from doing this, since if the second cipher is believed strong there is no need to use Skipjack in the first place, and if it is believed weak it does not protect the traffic from the government anyway.

A refinement on this approach encrypts only the LEAF. A LEAF encryption scheme could be integrated into a key exchange protocol that produces "extra" shared secret bits (such as Diffie-Hellman [DH76]). Since only 80 bits are required for the Skipjack session key, 128 of the other bits could be used as a Vernam cipher against the LEAF. Note that this scheme is not directly implementable with the EES PCMCIA card key exchange protocol, which does not permit external access to the negotiated key bits. However, an additional key exchange could be performed in software on the host processors.

Another option is to negotiate keys (and LEAFs) out-of-band or in advance. While it is not clear that there is any way EES could prevent such an attack, neither is such a scheme very useful in practical applications. Users would never be able to communicate securely without pre-negotiation or the use of a trusted channel. If a trusted channel existed, it could be just as easily used for the traffic itself. For some applications, however, such as bulk file encryption, pre-negotiated keys may be practical.

2.2 LEAF Feedback

Another possible approach is to avoid sending the LEAF altogether. Depending on the cryptographic mode this can be surprisingly simple. Recall that LEAFs are generated and loaded along with the IV. While applications cannot easily force the chip to use an externally-chosen IV, they can easily generate a new one. Upon negotiation of a session key, the receiving side of a rogue application can simply generate a new IV/LEAF and feed it back to itself, the sender never having sent the IV/LEAF at all. This still leaves the problem of IV synchronization. Because IVs cannot be loaded without a LEAF and LEAF checksums appear to be bound to the IV, sender and receiver have no way to communicate a directly loadable IV without also communicating the corresponding LEAF. Most cryptographic modes require the sender and receiver to synchronize on the IV. This is not an insurmountable problem, however. It is possible to compose efficient implementations of each FIPS-81 cryptographic mode in terms of other modes without explicitly loading the sender's IV at the decrypting side. Let us consider "LEAF feedback" schemes for each commonly used cipher mode.

2.2.1 Electronic Code Book (ECB)

In ECB mode, there is no IV (or, more properly, the IV does not affect the cipher in any way). The primitive block cipher is used directly, without chaining or feedback from other blocks. LEAF feedback is therefore simple – each side generates an IV/LEAF immediately after the key is negotiated and loaded, and uses ECB mode for communication. The fact that sender and receiver generated different IVs does not matter.

ECB mode is itself vulnerable to a number of well-known attacks and is not considered suitable for general use.

2.2.2 Cipher Block Chaining (CBC)

In CBC mode encryption, the cleartext of each block is first exclusive-ORd (XOR) with the ciphertext of the previous block and then encrypted with the block cipher function. The first block is XORd against the IV. Decryption reverses the process, applying the XOR function after the ciphertext has been decrypted with the block cipher function. CBC mode is "self-synchronizing" in that decryption can recover from missing or damaged ciphertext blocks. Since successful decryption of a block depends only on receiving the previous block's ciphertext, loss of the IV only affects the first block. LEAF feedback with a new IV, therefore, corrupts the first block but this can be easily compensated for by prefixing one "dummy" block to the message, to be discarded by the receiver.

2.2.3 Cipher Feed Back (CFB)

CFB mode uses the result of successive encryption through a shift register (initialized with the IV) as a keystream generator. The block encrypt fuction is used to generate the stream, which is XOR mixed with the datastream for both encryption and decryption. The shift register input is "fed" with the ciphertext stream from previous blocks. The stream depends entirely on the key and the previous blocks of ciphertext. CFB can be implemented based on ECB mode with an external shift register and IV. This requires one call to the EES device for each cipherblock. Experiments with this method with a prototype EES card suggest that this method carries a significant bandwidth penalty, however, since each ECB call to the card takes about 38ms and a separate call is required for each 8-64 bit block of the stream.

A more efficient implementation takes advantage of CFB's limited error propagation. CFB mode, like CBC mode, is self-synchronizing, with complete recovery from missing or damaged ciphertext once the shift register has exhausted. CFB can therefore recover from an incorrect IV. The sender can prefix a "dummy" block to the ciphertext input stream and the receiver can feed back a freshly generated, random IV and employ a bulk CFB block decrypt directly (just as with CBC mode).

2.2.4 Output Feed Back (OFB)

OFB mode also uses the result of successive encryption through a shift register (initialized with the IV) as a keystream generator. The block encryption function is used to generate both the encrypt and decrypt streams. Subsequent stream values are not affected by the data. Note that the entire stream depends on the key and IV and therefore requires that both sender and receiver be able to load the same IV to generate the same streams. LEAF feedback cannot therefore use OFB mode directly, since the stream will never recover from an incorrect IV. However, OFB mode can be simulated using the ECB block encrypt function and an externally implemented shift register and stream XOR mixer. The IV must still be sent to the receiver (without the LEAF, of course) to initialize the external shift register. This method carries a high performance penalty, just as with the manual implementation of CFB mode described above.

A more efficient method exists to generate an OFB stream, however, using Cipher Feed Back (CFB) mode to simulate the stream generator for large blocks. The sender generates an IV/LEAF and encrypts with OFB mode directly in the standard manner and sends only the IV to the receiver. Two passes are required on the receive side. First, the receiver generates a new IV and LEAF and encrypts a single block of all zeros with CFB mode. Now the receiver can recreate the sender's OFB keystream mask for n blocks by CFB encrypting n blocks of zeros prefixed with the block created in the previous step XORd with the real IV. The subsequent blocks, after CFB encryption, can be XORd with the ciphertext (shifted by one block) to recover the cleartext. Note that while the XOR mixing must be performed separately on the host processor, only two calls to the EES device are required to decrypt an arbitrary length ciphertext (up to the maximum block decrypt size supported by the devices).

3 Interoperable Rogue Applications

A more interesting (and useful) class of rogue applications includes those that can interoperate with "legal" peers (those that make no effort to circumvent the escrow system), still without allowing law enforcement access. Such applications have much greater utility (and are a much greater threat to the escrow system) than non-interoperable rogues, because they have all the benefits of interoperability with other EES devices without the risk of exposure to wiretapping.

In the previous section, we discussed techniques for rogue applications to communicate with one another without sending the LEAF. Such applications could be modified to adapt their behavior to send the LEAF only when communicating with a legal peer. A simple way to construct such an application is to "test the water" by sending the peer device a bogus LEAF and then, if the exchange fails (because the peer is operating legally), sending a valid LEAF. Such a "two phase" protocol is not completely satisfactory, however, because it still renders traffic vulnerable to LEAF monitoring when communicating with legal applications. Furthermore, such a protocol cannot work with non-interactive applications such as electronic mail, file encryption, fax, etc.

A more general approach is to construct a LEAF field that will be accepted as valid by the receiver but that does not actually contain the encrypted session key.

3.1 Brute-Force LEAF Search

Recall that the LEAF structure contains three components: the unit serial number of the transmitter, the unit key-encrypted session key, and a 16 bit checksum, all encrypted as a block under the family key. Because the receiving chip knows only the session key, the IV, and the family key, but not the other chip's unit key or serial number, LEAF verification must be entirely on the basis of the 16 bit checksum. The checksum, which is presumably based only on the session key, IV and other LEAF data, cannot be extracted from or inserted into a LEAF without knowledge of the family key (and the encryption mode). It is therefore not possible for a rogue application to extract the checksum from a valid LEAF and re-insert into an invalid LEAF, or to damage only the encrypted session key in an otherwise-valid LEAF.

A rogue sender could simply use a different session key when generating the LEAF; this LEAF would appear internally consistent with a valid checksum but would contain the wrong session key. "Old" LEAFs are detected and rejected by the receiving chip, however, apparently by using the cleartext of the session key (rather than the unit-key encrypted session key) in the computation of the LEAF checksum.

Since the checksum is only 16 bits in length, however, another attack is possible.[1] For any session key and IV, 2^{112} of the 2^{128} possible LEAF structures will appear to have a valid checksum. Because the process of decrypting a randomly generated LEAF with the family key will tend to randomize the decrypted bits in the checksum field, any randomly generated 128 bit string will have a $1/2^{16}$ chance of appearing valid for the current session key and IV. Note that the sending chip, like the receiving chip, has a built-in LEAF-testing facility. Once a session key has been negotiated, an attacker can use the local EES device to find a valid-looking-but-invalid LEAF with an expected average of 2^{16} trials. This attack appears to be feasible in practice.

Such a randomly generated LEAF structure will be accepted as valid by the receiving chip and will enable EES communication. The traffic will not be subject to LEAF-based wiretap access, however. When the

[1]The first observation that LEAF checksums may be vulnerable to brute-force spoofing appears have been independently made by Ken Shirriff in a posting to the "sci.crypt" Usenet group on January 27, 1994.

wiretapper decrypts the rogue LEAF with the family key, the checksum field will appear valid but the unit identifier and encrypted session key fields will contain only random bit strings.

3.2 Experimental Results

We measured the time to test randomly selected LEAFs on an EES PCMCIA card. All experiments were conducted with a Mykotronx prototype EES card connected through a Spyrus SCSI PCMCIA reader to a Sun Sparc-10 host running SunOS 4.1.3. We made no effort to optimize the communication with the card or library, using the standard prototype PCMCIA library and device drivers as delivered. Recall that the EES PCMCIA interface is fairly loosely-coupled to the host processor and supports a more restricted set of cryptographic operations than the basic Clipper/Capstone chips themselves. Therefore, LEAF-testing operations on the PCMCIA card are inherently slower than the same operations on a more tightly-coupled EES device or on a special-purpose host with a built-in EES processor. It is also possible that communication with the card can be made faster with the more tightly-coupled PCMCIA interfaces found on most laptop computers. The communication time with the card interface dominates the cost of most operations in the environment we examined; host processor speed was not a significant determining factor. We assume our results to be approximately representative of typical implementations in a worst-case environment.

Our test application required about 38ms to generate (with a pseudorandom generator) a LEAF-size bit string, send it through the PCMCIA library to the EES card and check the result. Since, on average, 2^{16} random LEAFs must be generated and tested before one with a valid checksum is found, a rogue PCMCIA application can search for a valid-looking LEAF in $38 * 2^{16}$ ms, or 2,490,368 ms, which is about 42 minutes.

42 minutes obviously adds too much latency to channel setup time to be useful in real-time applications such as secure telephone calls. For less interactive applications, particularly secure electronic mail, fax and file storage systems, such a delay may be acceptable. Furthermore, the attack has almost linear speedup with parallel processing. With 60 PCMCIA cards, a valid-looking LEAF could be expected in under 45 seconds. Also, it may be reasonable to expect several orders of magnitude reduction in search time with more direct use of a Capstone or Clipper chip. Since those devices are not expected to be made available for unrestricted use outside embedded products (as the PCMCIA cards are), however, it is likely that practical implementations of this attack will be limited to applications that use the PCMCIA interface.

We implemented this attack for a simple encrypted file storage application that we built as a testbed. Other than the 30-50 minutes of latency added by the LEAF search at encryption time (which is performed "offline" from the user interface), the rogue version is functionally identical to the version that follows the approved interface. In a storage application the LEAF-search delay is almost completely transparent, since most user operation can proceed normally prior to its completion. In store-and-forward messaging applications, such as electronic mail, however, the LEAF search delays message delivery. Whether this is acceptable depends on the application; additional computing resources, in the form of EES PCMCIA cards (perhaps borrowed from nearby idle workstations) can reduce the delay. It may also be possible for messaging applications to precompute session keys and bogus LEAFs prior to their use, especially if the number of possible recipients is small. We did not implement any of these conveniences, however.

In interactive applications such as secure telephony, the search time required for LEAF forgery during call setup may render the technique impractical. Other than parallel processing with additional EES devices, there do not appear to be viable shortcuts for reducing this search time. If call setup uses a negotiated key exchange, the originator cannot generally predict the session key and therefore cannot conduct the LEAF search in advance. Neither do there appear to be shortcuts to testing an average of 2^{16} LEAF values. The formulation of the problem, in which the attacker need only discover *some* session key and corresponding LEAF, seems at first blush to admit a so-called "birthday attack" requiring only $\sqrt{2^{16}} = 2^8$ trials. However, because the LEAF checksum is cryptographically protected by the family key, there appears to be no obvious way to perform the constant-time lookup on the checksum required for each probe in such an attack.

Because no widely-deployed "official" EES PCMCIA applications existed at the time of this writing, there were no third-party supplied systems available against which we could exploit LEAF forgery techniques. We have every reason to believe, however, that building interoperable rogue versions of any non-interactive PCMCIA application that implements an open protocol would be a straightforward matter.

4 Discussion

The EES failure modes described in this paper do not have the same semantic implications as protocol failures in the classic sense. None of the methods given here permit an attacker to discover the contents of encrypted traffic or compromise the integrity of signed messages. Nothing here affects the strength of the system from the point of view of the communicating parties; indeed, in some sense these techniques increase the security of EES-based protocols by eliminating the LEAF as a source of attack.

Instead, these methods attack an unusual aspect of EES requirements – the attempt to enforce access for a third party who is not an active participant in any part of EES-based communication. Once the system has been deployed, there is little further that the third party (the wiretapper) can do to protect its interests. In effect, the wiretapper actively participates in the protocol only by providing the narrow interface to the tamper-resistant EES module that requires that a LEAF be loaded prior to executing a decrypt operation. Our attacks thwart the wiretapper by using that interface in unexpected ways.

In considering countermeasures to these attacks, it is useful to divide the properties of the EES system into three somewhat overlapping categories:

- Fundamental. This category includes the properties of any key escrow system in a particular application domain (e.g., widely available components, FIPS-81 compatibility, identifiable LEAFs, etc.). These properties cannot be changed without affecting the applicability of the system.

- Architectural. The basic properties of the system decided upon early in the design process (e.g., the size of the LEAF field, the crypto-synchronization protocol, etc.). Changing the architecture requires re-engineering of a significant fraction of system components.

- Implementation. The characteristics of the actual EES devices and software. These can be changed by replacing or modifying the components in question.

In this paper we have focused primarily on weaknesses that are either fundamental or that arise from the EES architecture. In particular, we did not attempt to discover or exploit "bugs" in the prototype EES devices.

It is not clear that it is possible to construct an EES system that is both completely invulnerable to all kinds of exploitation as well as generally useful. Let us consider modifications to the EES interface that frustrate the various attacks.

Non-interoperable applications are particularly hard to prevent, since they are free to use the EES interface in any way they choose. LEAF feedback techniques can be discouraged by having devices recognize (and refuse to accept) locally-generated LEAFs. This would make the EES system difficult to deploy in legitimate secure storage applications, however, and such restrictions could be circumvented easily by using two devices on the receiving side, one for LEAF generation and one for decryption.

The interoperable LEAF-search method can be made less attractive by increasing the time required to check a LEAF. The ability to do this is limited by the fact that any reduction in LEAF-checking performance also degrades the performance of legal applications. EES PCMCIA cards, on which LEAFs can be feasibly searched for, already require approximately 38 ms to load an IV and LEAF. Slowing this to, say, two seconds, would noticeably increase the setup time for legitimate interactive traffic but only adds a factor of 50 to the time required by the offline rogue LEAF searcher (who could compensate with as much parallel processing as desired).

Alternatively, EES devices could limit the number of incorrect LEAFs they will accept (perhaps self-destructing after some threshold has been reached), or could impose a longer delay before returning the result of an attempt to load an invalid LEAF. These approaches are difficult to engineer reliably, however, and greatly increase the vulnerability of the system to denial-of-service attacks by an adversary who can inject noise into a receiver's datastream.

A more robust solution increases the size of the LEAF checksum to 32 or 64 bits, making exhaustive search infeasible. Since there is no "extra room" in the existing 128 bit LEAF package, any increase in checksum size would necessitate either increasing the LEAF size or reducing the size of the other LEAF fields. Increasing the size of the LEAF package to, say, 192 bits would provide room for an additional 64 bits of checksum redundancy but would would likely require significant re-engineering of many existing EES components, from the processors themselves to the protocols and applications that use them. Within the constraints of the 128 bit package, checksum size can increase only at the expense of either the unit ID or encrypted session key fields. The 32 bit unit ID field appears to be at the minimum possible size given the intended scope of the EES program (a previous version of the LEAF with a 25 bit unit ID was considered inadequate [NIST94a]). It may be possible to use bits from the encrypted session key field to increase the checksum size, at some expense in law enforcement wiretap access performance. If only 64 bits of the encrypted session key were included in the LEAF, the wiretapper could exhaustively search for the remaining 16 bits at decrypt time. Such a search, with properly optimized hardware, would likely add at most a few seconds to the decrypt time and would enable 32 bit LEAF checksums within the existing LEAF size constraints.

Finally, a more drastic approach, which thwarts non- interoperable as well as interoperable rogues, is to sharply restrict the availability of EES devices to those users and applications that are trusted not to abuse them. PCMCIA cards, being inherently portable, would need to be handled with particular care to avoid their use by unauthorized individuals. Of course, it is not at all clear that such restrictions could be made effective or consistent with the goals of the EES program, which aims to make the system widely available to the public.

5 Conclusions

The EES attempts to balance the seemingly conflicting goals of making widely available a strong cryptographic system while also ensuring government access to encrypted traffic. Rogue applications defeat EES by making use of the cipher without the government "back door." Whether rogues threaten the viability of the EES program depends on whether they can be easily deployed for a significant fraction of the traffic in their target application areas.

We have identified two classes of rogues. The most general, those that can take unilateral action to interoperate with legal EES systems, are potentially the most damaging to the EES program. These applications are functionally similar to their non-rogue counterparts and have all the advantages of general interoperability without the risk of wiretapping. The techniques used to implement them do carry enough of a performance penalty, however, to limit their usefulness in real-time voice telephony, which is perhaps the government's richest source of wiretap-based intelligence. The second class, those that can interoperate only with other rogue devices explicitly designed to thwart the LEAF, are also the easiest to implement and the hardest to prevent. Devices in this class are not as great a threat to the EES program as those in the former class because they do not conform to official interoperability standards. However, if the population of legal devices is substantially smaller than that of rogue devices in a particular market, lack of legal interoperability may not be a significant disadvantage.

It is worth noting that, with EES PCMCIA cards, a rogue system can be constructed with little more than a software modification to a legal system. Furthermore, while some expertise may be required to construct a rogue version of an existing system, it is likely that little or no special skill would be required to install and operate the modified software. In particular, one can imagine "patches" to defeat key escrow in EES-based systems being distributed over networks such as the Internet in much the same way that other software is distributed today. Experience with "pirate" cable TV descramblers, cellular telephone access codes, and copy-protected PC software suggests that rogue modifications to circumvent controls on widely-deployed systems tend to emerge quickly even when moderate safeguards against such modifications are present. EES PCMCIA-based systems appear to be particularly vulnerable to such abuse because the interface to the system is controlled completely in software on the user's host computer. The barriers to constructing a rogue software system are much smaller than those to modifying and deploying hardware-based rogue products, and the development and proliferation of software modifications is very difficult to regulate in the presence of open standards and communications networks.

6 Acknowledgments

Steve Bellovin, Whitfield Diffie, Joan Feigenbaum, Peter Honeyman, Steve Kent, Jack Lacy, Tom London, Dave Maher, Andrew Odlyzko, Rob Pike, Jim Reeds, Mike Reiter and Bruce Schneier offered innumerable comments that have greatly improved this paper. The suggestion to use bits from the encrypted session key to augment the checksum size arose from discussions with Steve Bellovin. We would like to especially acknowledge the generous assistance of various individuals at NSA in providing us with prototype EES PCMCIA cards and technical data. We are particularly grateful for the spirit of openess and collegiality displayed by the members of NSA in reviewing these results.

The author remains, of course, solely responsible for any errors in this paper.

The name "Tessera" is a trademark of Tessera, Inc., which neither produced nor licensed the government-supplied EES PCMCIA cards to which we refer in this paper. We know of no connection between Tessera, Inc. and the EES program. Previous references to EES PCMCIA cards as "Tessera cards" appear to have been made in error.

7 Postscript

Some of the results in this paper are based on experiments conducted with pre-release prototype EES PCMCIA cards and software obtained from NSA. The production version of the EES PCMCIA system will likely exhibit different performance characteristics and have a different interface from the version we examined. The reader is cautioned to view any experimental results presented here as a "proof of concept" and not as representative of the exact performance of the final system. We understand that NSA intends to incorporate features to discourage these attacks into future versions of EES devices.

References

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory,* November 1976.

[Mar93] J. Markoff. Communications plan to balance government access with privacy. *New York Times,* April 16, 1993.

[NBS77] National Bureau of Standards. Data Encryption Standard, *Federal Information Processing Standards Publication 46*, Government Printing Office, Washington, D. C., 1977.

[NBS80] National Bureau of Standards. Data Encryption Standard Modes of Operation, *Federal Information Processing Standards Publication 81*, Government Printing Office, Washington, D.C., 1980.

[NIST94] National Institute for Standards and Technology. Escrowed Encryption Standard, *Federal Information Processing Standards Publication 185*, U.S. Dept. of Commerce, 1994.

[NIST94a] National Institute for Standards and Technology. *Technical Fact Sheet on Blaze Report and Key Escrow Encryption.* June 15, 1994.