High-Bandwidth Encryption with Low-Bandwidth Smartcards

Matt Blaze
AT&T Bell Laboratories, Murray Hill, NJ 07974
mab@research.att.com

December 3, 1995

Abstract

This paper describes a simple protocol, the *Remotely Keyed Encryption Protocol (RKEP),* that enables a secure, but bandwidth-limited, cryptographic smartcard to function as a high-bandwidth secret-key encryption and decryption engine for an insecure, but fast, host processor. The host processor assumes most of the computational and bandwidth burden of each cryptographic operation without ever learning the secret key stored on the card. By varying the parameters of the protocol, arbitrary size blocks can be processed by the host with only a single small message exchange with the card and minimal card computation. RKEP works with any conventional block cipher and requires only standard ECB mode block cipher operations on the smartcard, permitting its implementation with off-the-shelf components. There is no storage overhead. Computational overhead is minimal, and includes the calculation of a cryptographic hash function as well as a conventional cipher function on the host processor.

1 Introduction

Cryptographic smartcards are an important building block in many modern security applications. In particular, their tamper-resistant packaging, low cost, inherent portability, and loose coupling to the host make them especially attractive for use as secret key storage tokens when the host cannot be trusted to itself store a secret key. Unfortunately, however, these same properties also limit the utility of smartcards; loose coupling and low cost usually mean that a card cannot process data at nearly the bandwidth of the host to which it is attached.

In some applications, such as challenge-response authentication protocols and digital signatures of message digests, the low bandwidth of smartcards is not an issue; the secret key stored on the card is used only occasionally and speed requirements are minimal. In other applications, however, including file encryption, encrypted realtime traffic and encrypted multimedia and video, a much larger volume of traffic is encrypted and decrypted under the card's secret key. Here the bandwidth to the card can be a serious bottleneck, with the speed of the system limited by the latency and bandwidth of the card interface and the computational capacity of the card.

It is therefore often desirable to shift as much work as possible from the slow, computationally limited card to the fast, more powerful host. This typically involves a tradeoff among security, performance, and cost. At one extreme, we could engineer the smartcard and interface so that its performance matches that of the host processor. This is not always technologically feasible, however, and obviously can increase the total cost of the system. At the other extreme, we could limit the use of the card to key storage, copying the key back to the host processor for use there prior to performing any cryptographic operations. Revealing the key entails a change to the usual smartcard security model, however, since the host processor must now be trusted to safeguard the key.

In applications that require high-bandwidth bulk encryption with smartcard-based key management, we would prefer a scheme that allows work to be shifted to the host processor without also increasing its trust requirements. Previous work in the area of "asymmetric capacity" cryptography has focused on public-key cryptosystems in which parts of the computational burden can be shifted from one communicating party to the other [BCY93] and does not address this particular problem. Other work, *e.g.,* [BFS90] [BFKR90], is concerned with hiding instances of specific types of distributed computation and cannot be applied directly to encryption with block ciphers. In this paper, on the other hand, we present a simple protocol, the *Remotely Keyed Encryption Protocol (RKEP),* for use with a conventional secret key cryptosystem in which a secure, but slow, smartcard shifts most of the work to its insecure, but fast, host processor.

2 The RKEP Protocol

The players in our scheme consist of a *host* and a *card.* The host wants to encrypt and decrypt large blocks under a secret key stored on the card. While the host is by definition trusted to process the plaintext that it is actually handling, it is not allowed to know the key. The smartcard knows the key K, but is computationally and bandwidth limited and cannot process entire blocks in the time required by the host. Our protocol allows the host to perform a single, fixed-size low-bandwidth interaction with the card to obtain enough information to encrypt or decrypt a given arbitrary length block. Without online access to the smartcard, however, the host cannot encrypt or decrypt other blocks, even given past card access.

RKEP requires that the smartcard and host share a block cipher algorithm, such as DES [NBS77], that operates on b-bit cipherblocks and that is keyed with a k-bit key. (Strictly speaking, there is no requirement that the host and card implement the same cipher function; if two different ciphers are used, however, the security of the system is limited to that of the weaker cipher.) There must be a secure (secret and unspoofable) channel between the host and the card. The host operates on large blocks of plaintext (P) and ciphertext (C), each consisting of a series of n individual b bit cipherblocks, denoted $P_1 ... P_n$ and $C_1 ... C_n$, respectively. $I_1 ... I_n$ denote temporary "intermediate" cipherblocks used internally on the host by the protocol. For the purposes of this discussion, we assume that $k \leq b$ (we will remove this restriction below, however). We denote encryption of plaintext block p under key K as $E_K(p)$ and decryption of ciphertext block c under key K as $D_K(c)$. $\oplus$ denotes bitwise exclusive-OR. We assume that the host can calculate efficiently a public function $H(t)$ that returns a b-bit cryptographic (one-way and collision-free) hash of arbitrary length bitstring t. Finally, we assume the card has a public function $M(c)$ that maps the b-bit string c to a k-bit key string.

The encryption of n-cipherblock plaintext block P producing ciphertext C is shown in Figure 1. Decryption of C is shown in Figure 2.

Note that for ciphers where $k > b$, it is easy to adapt RKEP so that several cipherblocks (enough to produce k key bits) are sent to and encrypted on the card to calculate K_P.

Host	*Card*
do $i = 2$ to n	
$\quad I_i = P_i \oplus H(P_1)$	
$I_1 = P_1 \oplus H(I_2 ... I_n)$	
send I_1 to card	
	$C_1 = E_K(I_1)$
	$K_P = M(E_K(C_1))$
	send C_1, K_P to host
do $i = 2$ to n	
$\quad C_i = E_{K_P}(I_i \oplus C_{i-1})$	

Figure 1: RKEP encryption of P to obtain C

Host	*Card*
send C_1 to card	
	$K_P = M(E_K(C_1))$
	$I_1 = D_K(C_1)$
	send K_P, I_1 to host
do $i = 2$ to n	
$\quad I_i = D_{K_P}(C_i) \oplus C_{i-1}$	
$P_1 = I_1 \oplus H(I_2 ... I_n)$	
do $i = 2$ to n	
$\quad P_i = I_i \oplus H(P_1)$	

Figure 2: RKEP decryption of C to obtain P

3 Discussion

3.1 Limitations, observations and attacks

It is important to understand the basic limitations of RKEP and of smatcard-based encryption generally. At best, data encrypted by a host following RKEP can be decrypted only with the aid of online access to a card with the correct key, and decryption by a host following RKEP will only produce the intended cleartext when the encrypting host had online access to a card with the correct key. That is, the security semantics approximate encryption and decryption performed entirely on a smartcard *with respect to a peer that is following the protocol.* Nothing, of course, prevents two "conspiring" peers from exchanging encrypted data without using the card at all, or from bilaterally choosing to re-use a key from some past card transaction. Indeed, even protocols that require encryption entirely on the card can be circumvented by peers that choose to follow some other protocol to encrypt their traffic.

Ideally, we expect the protocol to have the property that without online access to the smartcard, a host can neither encrypt nor decrypt data under the card's key, even given past access to the card. That is, encryption and decryption without the card should be no easier than breaking the underlying cipher. In particular, the session key generated by the card for a given block should have no more than $1/2^k$ probability of being the correct session key for some other block.

Assuming strong block cipher and hash functions, the ciphertext for each cipherblock in C appears to depend on the "card secret" key K and the plaintext of all bits in P.

RKEP appears to be as secure as the underlying cipher against decryption by a third party (or even a previous card holder) without the card. Since the session key depends on K and $H(P)$, there should be no useful correlation between the K_P used to encrypt one large block and the key for another. Obviously, it is possible (with probability $1/2^k$) that the K_P for some block is the same as the K_P for some previously processed block. This is not an actual weakness, however, since it is no more likely that such a key will be correct than is any other randomly chosen key.

It may be possible to exploit past card access to assist future encryption without the card. Several attacks allow a host that has had past access to the card to encrypt some chosen bits without the card with less effort than breaking the underlying cipher or otherwise learning K. A "birthday" attack against the hash function allows encryption of a b bit chosen cipherblock in $O(2^{b/2})$ time. This attack requires that the attacker have previously probed the card $2^{b/2}$ times. Other tradeoffs are also possible, allowing fewer past probes in exchange for more work per forged encryption.

An attacker who has used the card once and records the corresponding values of P, C, I and K_P is able to encrypt m chosen bits anywhere in $P_2...P_n$ with $O(2^m)$ trial encryptions without online card access. The attacker uses the old C_1 and therefore K_P and randomly changes bits in $C_2...C_n$ until the chosen bits decrypt to the desired values. The rest of the bits in P are random, however.

Depending on the protocol parameters (in particular, the cipherblock size b) and the performance characteristics of the card, neither of these attacks is likely to pose a serious threat to most practical applications. They can be prevented by choosing a cipher with a sufficiently large b or with the use of standard cryptographic integrity techniques. Of course, none of this is a proof of security; there may be other, as yet undiscovered, attacks that allow more efficient encryption or decryption without online access to the card.

It appears that any cryptographically strong block cipher (DES, IDEA, Skipjack, etc.) and hash function (SHA, MD5, etc.) can be used with this scheme.

RKEP can be adapted for use as a simple integrity mechanism by setting some bits of each large plaintext block to some fixed value (say, all zeros). Tampering with the ciphertext is detected by checking these bits on decryption.

3.2 Performance

Regardless of the value of n, any size block can be encrypted or decrypted on the host with only one card interaction (with two cipherblock operations). However, no bit of P is available until all bits have been processed. n should be therefore be chosen to yield the largest size P that the host naturally processes as a unit. Ordinarily, this will follow from some aspect of the application, such as the file system block size or video frame buffer size. n can be varied as a parameter of the system, even among successive blocks. If n is fixed, the large blocks are suitable for use with any standard cryptographic "mode of operation" [NBS80].

The scheme requires no communications overhead in transmitting or storing the ciphertext; the plaintext and ciphertext sizes are equal.

Any size block can be encrypted or decrypted with one card interaction, with the card performing exactly two cipherblock encrypt / decrypt operations in each. If the hash function H is efficient to compute relative to the cipher function, the overhead on the host is comparable to that of simply performing the entire encryption there with K. The additional host overhead includes processing each bit in P with the H function and the $\oplus$ operation. There may also be additional latency introduced while waiting for responses from the smart card.

In the simplest implementation, the host simply performs the procedures given in Figures 1 and 2 directly for each large block. The total time required for an encryption or decryption of the n cipherblock block P is simply the sum of the operations: one application of H on a single cipherblock on the host, one application of application of H on $n-1$ cipherblocks on the host, two block cipher operations on the smartcard (plus any communications cost associated with transmitting and receiving two cipher blocks between host and card), and $n-1$ applications of the block cipher function on the host. Observe that, in such an implementation, the host is idle while the card is calculating and transmitting its two block cipher operations. The card is similarly idle while the host is calculating its hash and cipher functions.

An implementation can be optimized by employing a "pipeline" to yield closer to 100% host utilization when several blocks are to be encrypted or decrypted in succession. On encryption, once the host has finished calculating I_1 for some block, it can transmit the value to the card and move on to the next block. When the card is ready with the C_1 and K_s values for the first block, the host can return to processing that block. Similarly, on decryption, a host can transmit the next block's C_1 value as soon as the previous block's I_1 and K_s value is received. It is possible to overlap the processing of arbitrarily many blocks in this manner, at the cost of "buffer" memory proportional to the number of blocks to be overlapped.

3.3 DES/SHA and Fortezza Implementations

We implemented RKEP using a version of the AT&T smartcard as the key storage and encryption engine for the Cryptographic File System (CFS) [Bla93]. CFS is an encrypting file system for Unix-like operating systems. Files are automatically and transparently encrypted and decrypted as they

Scheme	4KB Block Encrypt	Bandwidth
Software-only	14 ms	2.4 Mbps
Smartcard-only	4100 ms	7900 bps
RKEP	31 ms	1.0 Mbps
Optimized RKEP	19 ms	1.7 Mbps

Figure 3: Encryption Performance of Pentium-based CFS systems

are read and written, and therefore the performance of the system is highly dependent on the speed of the encryption function. A software DES implementation on a modern (*e.g.,* Pentium, Sparc-20, etc.) workstation provides nearly transparent performance under typical workloads. The AT&T smartcard, however, is connected to the host computer via a slow (9600 bps) serial link. Effective encryption bandwidth to the card (taking into account communication overhead and card processor latency) is approximately 8000 bps, with a minimum latency of about 10ms for a single ECB encrypt/decrypt. The smartcard has a basic key storage facility (protected by a user password) and the DES ECB encrypt / decrypt function. Faster (*e.g.,* PCMCIA-based) smartcard interfaces are available for some smartcards and host configurations, but are not universally available, especially for Unix computers. Our smartcard system represents something of a "worst case" configuration.

The software-only implementation of DES used in the standard version of CFS encrypts, on a Pentium-90 workstation, at about 2.4Mbps. The bandwidth of the smartcard is slower than the software by a factor of about 300. Clearly, the 9600 bps smartcard is unsuitable as a file encryption engine in an online file encryption application such as CFS. Under RKEP, however, encryption with the smartcard has only a small performance penalty compared with the software-only system.

We used a reasonably well-optimized implementation of SHA as the hash function H. This version of SHA hashes large blocks at approximately 18Mbps, and can hash a single 64 bit DES cipherblock in about 20 microseconds. We selected the large block size to mirror the block size of the file system, 4K bytes (32768 bits). As expected, the performance of the smartcard-based system under RKEP is about half that of the original software-only system and far better than using the smartcard by itself. A "pipeline" RKEP implementation that attempts to reduce latency by pre-fetching blocks on decryption and deferring writes improves the performance considerably, approaching that of the original software system. We compare the measured performance of the four systems in Figure 3.

We have also implemented a version of CFS with RKEP based on a prototype of the US DoD *Fortezza* PCMCIA card. The Fortezza card implements the (classified) Skipjack algorithm [NIST94] and has key management facilities that permit secure key establishment and storage. Because Skipjack is classified we could not implement it in software on the host. Skipjack, like DES, has a 64 bit codebook, so we chose triple DES (3DES) as a comparable host cipher. (The security of the system, of course, is no greater than that of the *weaker* of 3DES and Skipjack.) The performance results of RKEP in this application were similar to those of the smartcard-based system; optimized RKEP was about one 30% slower than the software-only implementation and many times faster than the Fortezza-only system. However, its exact performance characteristics were highly sensitive to the implementation of an experimental prototype PCMCIA device driver, and we therefore omit detailed measurements here. (An aside: our implementation defeated the key escrow scheme used in Fortezza by re-generating and re-loading the Law Enforcement Access Field each time a key was loaded. This was done for reasons having less to do with our dislike of key escrow than with architectural constraints; the CFS system's internal structure did not easily accommodate the LEAF field and it was more convenient to regenerate it as needed than to find a place to store it in the file system.)

4 Conclusions

RKEP is appropriate in any application in which a trusted "cryptographic module" is relied upon for key security but cannot perform bulk encryption at the rate required by the host application. Cryptographic modules, which are often packaged in inexpensive, removable smartcards and PCMCIA devices, are architecturally different from traditional "co-processors". In particular, most co-processors are designed and connected for the express purpose of improving performance (*e.g.,* a floating-point arithmetic processor that is tightly coupled, or even part of, the host CPU). Cryptographic co-processors, on the other hand, often function primarily to provide an encapsulated, portable security environment, and their architecture reflects this different purpose. Frequently, this means that bulk encryption through a cryptographic co-processor results in much lower data rates than would be possible in software on the host.

While we have been primarily concerned with applying RKEP to bulk file encryption using smartcards, the protocol has application in several other configuration as well. "Set-top boxes" used in advanced cable television systems, for example, often require highly tamper-resistant cryptographic processing but cannot rely on high-bandwidth special hardware due to cost constraints. Cellular telephone systems have similar design constraints.

Because RKEP requires no special software or protocol support on the cryptographic module, it can be implemented with virtually any off-the-shelf smartcard, PCMCIA device, or encryption chip that can perform ECB encryption and decryption. The module's interface need only have the ability to return the result of single cipherblock ECB operations. All other special processing is handled by the host.

5 Acknowledgements

The author thanks Joan Feigenbaum, Matthew Franklin, Jack Lacy, Dave Maher, Mike Reiter and the anonymous reviewers for their many helpful comments on previous drafts.

References

[BFKR90] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with Low Communication Overhead. *Advances in Cryptology – Crypto '90, Lecture Notes in Computer Science,* volume 537, Springer, Berlin, 1991, pp. 62–76.

[BFS90] D. Beaver, J. Feigenbaum, and V. Shoup. Hiding Instances in Zero-Knowledge Proof Systems. *Advances in Cryptology – Crypto '90, Lecture Notes in Computer Science,* volume 537, Springer, Berlin, 1991, pp. 326–338.

[BCY93] M. J. Beller, L. Chang and Y. Yacobi. Privacy and Authentication in a Portable Communications System. *IEEE Journal on Selected Areas in Communications,* August, 1993.

[Bla93] M. Blaze. A Cryptographic File System for Unix. *Proc. 1st ACM Conference on Computer and Communications Security,* Fairfax, VA., November 1993

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

[NBS80] Data Encryption Standard. National Bureau of Standards, *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.