Mobile User Location-specific Encryption (MULE): Using Your Office as Your Password*

Ahren Studer
Carnegie Mellon University
astuder@cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

ABSTRACT

Data breaches due to stolen laptops are a major problem. Solutions exist to secure sensitive files on laptops, but are rarely deployed because users view them as inconvenient. This work examines how to provide an unobtrusive system to securely encrypt files on laptops. We observe that only a fraction of users' files contain sensitive information. In addition, the majority of users' accesses to these sensitive files occur while in a trusted location that malicious parties are unable to access. Rather than protecting all of the user's files, we secure user designated sensitive files that are rarely accessed outside of specified trusted locations. Our approach is to use information and services available only in a trusted location to assist in key derivation without user involvement and without authenticating the laptop to any outside service. We study two settings: home use where zero management overhead is needed (i.e., a "plug-and-play" solution) and a corporate setting where staff management of a whitelist of acceptable devices allows a higher level of security. We have implemented both systems and found automatic key derivation introduces a five second delay during the initial access to sensitive files.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication, physical security, unauthorized access*

General Terms

Security, Human Factors

*This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and MURI W 911 NF 0710287 from the Army Research Office, and grants CNS-0831440, CNS-0627357 and CCF-0424422 from the National Science Foundation. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, CyLab, NSF, or the U.S. Government or any of its agencies.

WiSec'10, March 22–24, 2010, Hoboken, New Jersey, USA.

1. INTRODUCTION

Lost/stolen laptops are a major cause of leaked data [25,30]. In previous years, lost or stolen laptops resulted in the exposure of over 30 million unencrypted records [23]. These leaks expose financial account information, health records, social security numbers, and other crucial data. This issue is not limited to corporate laptops that contain hundreds or thousands of records. As financial institutions and the IRS move towards electronic systems (i.e., e-statements and e-filing), more home users are storing sensitive data on their personal laptops. The loss of a personal laptop may leak the owner's account numbers, social security number, or even health records.

A simple solution to preventing such leaks is to authenticate users and encrypt sensitive data [19,29]. However, a recent survey found that only half of security and privacy professionals indicate their companies encrypt data [18]. Without access to policies, we cannot know why so few companies are using encryption. However, without a standard that enforces encryption, most users view the technology as an unnecessary burden. Company IT personnel will also see encryption technologies as a burden, similar to the headache associated with account management. When given the choice, users often prefer convenience over security and only worry about leaking data after losing their laptop [24]. What makes securing files on laptops hard is that users and administrators want a system that "just works" with little or no actions on their part. However, there is often a tradeoff between security & user effort.

In this paper, we examine the extreme case of no user effort and little IT administration to determine what level of security is possible. Our goal is to remove user effort associated with encryption technology while achieving the same or better security compared to traditional password-based approaches. Prior work by Corner & Noble [9] reduced user overhead associated with securing files by leveraging a cryptographic token which shares secrets with the laptop. Provided the token is within radio range of the laptop, the user can access files. Once the token is out of range, files are encrypted. Such an approach has the advantage that all of the user's files are protected, but requires the user to carry a token to allow access to any of the files. We lower the threshold and examine how much security one can achieve with no per laptop secrets on company managed devices and zero user effort (i.e., no password entry, biometric entry, or possession of cryptographic tokens) in the common case.

We observe that only a fraction of the files on users' laptops are sensitive and most users only access those sensitive files when they are in a location they feel is difficult for malicious parties to access (e.g., a home office or a desk at work). Our approach is to encrypt user-specified sensitive files and leave all other files unencrypted and always accessible. Given the majority of accesses to sensitive files occur in a trusted location, Mobile User Location-specific

Encryption (MULE) uses location-specific information from the trusted location to automatically derive a decryption key and allow access to the sensitive files. Once the user is inactive, logs off, or puts the computer to sleep, the files are automatically re-encrypted and the key is deleted from the computer. In the rare case that a user wants to access sensitive files outside the trusted locations, the user can enter a secondary password to gain access. This password-based access also provides a fail-safe mechanism in case location-specific information or services are no longer available in a trusted location.

Convenience is the major advantage of MULE. The user can access files without performing any additional actions in the common case (accessing non-sensitive files or accessing sensitive files in a trusted location). Ideally, only in rare cases would MULE require user effort.

A malicious entity in possession of the laptop with unconstrained access to a trusted location could access sensitive data. We propose a new attacker model, the *Outsider Thief (OT)*, to more accurately reflect the threat of a laptop thief. We describe this adversary model in more detail in Section 2.

We have implemented our system and find that the time needed to acquire the location-dependent key is less than 5 seconds depending on the security level, type of location-specific information, and technique used. Once the key is known, access to encrypted sensitive files is transparent to applications. Our implementation also includes the necessary tools to automatically re-encrypt sensitive files when the system is idle for a set period of time or put to sleep (e.g., the owner closes the laptop).

Contributions. We investigate the level of security one can achieve with encrypted sensitive files when no user effort is required. Rather than relying on something users know, have, or are, we explore using **where the user** is to perform access control. We propose two new mechanisms that derive keys based on location-specific information to allow the laptop to infer that it is physically located in a trusted location, without proving to any other device that it is in the trusted location. We evaluate an implementation of our system on readily available hardware.

2. PROBLEM FORMULATION

The goal of this work is to provide encryption of sensitive data while requiring minimal administrative effort and zero user effort during common accesses and moderate effort otherwise. Such encryption should, with high probability, prevent an adversary who steals the laptop from accessing files. Minimal administrative effort applies to a corporate setting and means IT personnel at most have to keep a list of not-yet-stolen laptops (i.e., there are no per-laptop secrets on a corporate server). A scheme requires zero user effort if a user can access sensitive files without entering a password or biometric, carrying around a hardware token, or adding additional hardware to their laptop. The main challenge is how to protect the key needed to decrypt sensitive data without requiring user or administrative effort.

To avoid user effort in the common case, we need a system that allows the laptop to automatically derive the key based on location-specific information when in a trusted location (the common accesses). The laptop can use this location specific key to decrypt sensitive files and provide the user access.

Location-based access control [10] addresses the different problem of proving to an outside system that a device/user is in a location and thus should have access to a resource. Our problem involves a laptop that wants to prove to itself that it is in a specific location (i.e., retrieve a decryption key). The laptop can leverage information and computation from other devices already in the location to perform this proof, but the other devices perform no authentication of the laptop and require no per-laptop secrets.

2.1 Assumptions

Sensitive Data Access Patterns. We assume users rarely access sensitive data outside of trusted locations. For example, users will work on taxes in their home or access customer accounts in the office. This assumption remains true for individuals that travel for work. Companies often disallow individuals from taking sensitive files out of the office [8]. For example, human resource employees often access employee records. However, while traveling to recruiting events, company policy may dictate that laptops must not contain unencrypted copies of employee records. Some users rarely access sensitive files in the same location. For example, a consultant that frequently travels may have no real office. For users without a trusted location, MULE provides no real advantage.

Available Hardware and Software. We assume laptops are equipped with a video camera and trusted computing technology, MULE users will accept the cost of installing a small device in the trusted location, and corporations that use MULE will have a whitelist of company laptops or a blacklist of stolen company laptops and a Public Key Infrastructure (PKI) or at least the means to distribute authentic public keys to their employees.

The majority of commodity laptops have webcams mounted into the frame around the display and already come equipped with trusted platform modules (TPMs) [33], an inexpensive coprocessor that enables a number of security related operations. Extracting keys from the TPM is infeasible without expensive hardware and extensive time.

Given the lack of laptop accessible location-identifying information in home and work offices, we assume users or their companies will install a Trusted Location Device (TLD) as part of MULE. The TLD provides location-specific information and responds to any machine that wants to run the key derivation protocol. Instead of performing location-based access control, the TLD performs no authentication of the requesting device. TLD secrets, location-specific information, and inputs to the key derivation process form the foundation of the secrecy of location-specific keys in MULE. The TLD is a spare machine connected to an inexpensive ($20) microcontroller to transmit location-specific information. The TLD also requires zero user effort after plugging it into the trusted location and minor maintenance in the corporate setting to ensure protection of keys (see Section 3 for more details).

We assume corporations keep track of company owned laptops using some type of unique identifier that the laptop knows. This could be a value the company assigns to the laptop (e.g., a network assigned name or IP address) or a value found in hardware (e.g., the MAC address of the laptop). When the laptop is lost/stolen, IT or property management personnel will remove the laptop from the company whitelist (or add it to a blacklist) of laptops that are allowed various services (e.g., access to the corporate wireless network). We leverage this company assigned unique identifier during calculations such that, once the laptop is stolen, the calculations fail, but no authentication of the identifier is performed—as part of MULE.

Any corporation with a secure web-site already uses a PKI to protect communication with the site. We assume that the company can use this same PKI to identify TLDs. Otherwise, IT personnel can easily create an in-house PKI for free using open source software like OpenSSL [21] and manually distribute the necessary credentials (i.e., CA's public key) to identify TLDs.

2.2 Outsider Thief (OT) Attacker Model

We propose the Outsider Thief (OT)—a realistic attacker to model a laptop thief. The thief has complete control over a stolen laptop, may visit a company office, and can launch attacks on the wireless network, but is unlikely to break into a user's home. After the OT steals the laptop, she can install any software and try to guess the user's login password. We assume the laptop has no malware before it is stolen, otherwise any sensitive data the user accesses could be leaked. Malware defenses are an important problem that is outside of the scope of this work. We assume fear of legal retribution prevents an OT from breaking into a home to access a home user's trusted location. Corporate trusted locations are publicly accessible, but are protected by guards and IT personnel which prevent an OT from successfully compromising devices in the trusted location. An OT can overhear, intercept, and inject messages on the wireless network. However, when outside of a trusted location, an attacker is unable to access the location's constrained channel [14] (e.g., infrared signals that are unable to pass through walls or sounds in an insulated room). We recognize that stronger attacker models exist, but we expect that defending against those adversaries requires additional user effort.

2.3 Requirements for Location-Specific Information Used to Derive Keys

When location-specific information is used for key derivation, the information must fulfill the following requirements to ensure successful and secure operation of MULE.

Easily Accessible. Once the laptop is powered on, placed in a trusted location, and the user is logged in, the laptop should have access to the information required for key derivation.

Unique to a Location. If the information is not unique, the laptop may automatically decrypt a user's sensitive files while outside of the originally defined location, an obvious security vulnerability.

Bounded Range. Location information should only be accessible within the location. Information accessible from outside of a building will apply to more than the location the user trusts.

Significant Entropy. Information used to derive the key within a location needs to have significant entropy so that it is hard to guess. Limited entropy would enable an attacker to guess the necessary values, spoof the location, and recover a key.

3. MULE OVERVIEW

MULE's goal is to protect sensitive files on mobile devices with zero user effort in the common case. Standard user login works independent of MULE and provides a form of weak user authentication. All non-sensitive files on the laptop are left unencrypted and are always accessible. Only user-specified sensitive files are encrypted. Figure 1 depicts an overview of the operation of MULE. When a user tries to access a sensitive file, MULE contacts a Trusted Location Device (TLD) which helps the laptop derive the key needed to decrypt sensitive files with zero user effort. The TLD generates a nonce and transmits it over the constrained channel [14]. We call this TLD generated nonce a location-specific message (m) because the properties of the constrained channel ensure that only devices within the trusted location can access the m associated with the current run of the protocol. A TLD is unable to authenticate requesters without per-laptop secrets and will respond to any key derivation request. However, the key derivation calculations are such that a TLD produces the wrong output if the requester uses the wrong m in calculations (e.g., the client is in a different location). After the TLD has helped derive the key, the user can access sensitive files without having performed any extra actions. During the rare occasion when a user accesses sensitive files outside of a trusted location, MULE will lack the correct location-specific information and key derivation will fail. In that case, we sacrifice some usability to preserve security and ask the user to enter a password as part of a location-independent key derivation scheme. The password allows the TPM on the laptop to decrypt a location-independent key which can decrypt the files. Once a valid key is available, the sensitive files are decrypted. When a user is idle for some set period of time, logs off, or puts the laptop to sleep, the laptop will re-encrypt the files and delete the key.

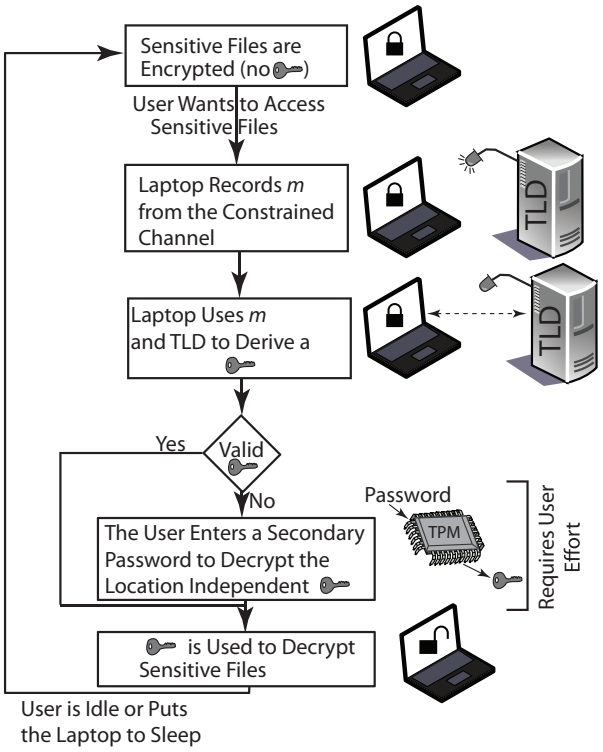


Figure 1: Operation of MULE

User effort is only needed when location-specific key derivation fails.

In this work, we present two location-specific key derivation protocols that leverage the same implementation of a constrained channel. The reason we have two protocols is that home and corporate users are willing to accept different levels of management overhead and use cases. In both protocols, we use an infrared (IR) LED and the laptop's webcam to implement a constrained channel. IR cannot pass through objects (e.g., walls, window blinds, or people). If one is viewing sensitive files on the screen, one should close the blinds to prevent an OT across the street from seeing the display—and also prevent access to the constrained channel. One downside to this constrained channel is the limited number of bits the TLD can reliably transmit to the laptop in a fixed amount of time. Based on the Nyquist frequency, a camera that captures X frames per second is limited to $X/2$ bits per second when using an on/off encoding scheme.[1] We could add multiple LEDs to encode more bits per frame. However, the user would have to pay careful attention to how the laptop is positioned within a trusted location to ensure the LEDs are easily differentiable so the laptop can successfully decode m. Instead, we design key derivation protocols which are

[1]The LED on is a 1. The LED off is a 0.

secure despite the use of a 20 to 30 bit long m which changes with each run of the protocol.

In Section 4, we describe the Home Key Derivation (HKD) protocol. In the home scenario, we assume the user wants to turn an existing computing device (e.g., old desktop or wireless router) into a TLD, hang the IR LED over the desk in the office (or other trusted location), and leave the system alone. Without access to the constrained channel, an OT can sit outside of the user's home, intercept all wireless communication, and find it infeasible to recover the key. If the laptop is ever stolen, the user can feel secure knowing a thief needs to break into her home to steal the TLD or attack the location-independent mechanism to recover a key which will decrypt sensitive files.

In Section 5, we describe the Corporate Key Derivation (CKD) protocol. In a business setting, malicious parties may surreptitiously gain access to the office for any number of reasons (e.g., a corporate open house or interview). As such, an OT may access the trusted location with the laptop in her possession. In that case, we leverage the company's guards and IT personnel to maintain the secrecy of the sensitive files. Guards will prevent an OT from breaking into locked rooms and subverting TLDs.[2] The company's IT personnel can maintain a simple white-list of valid MULE clients based on some static unique ID for the laptops (e.g., a network name, IP address, or MAC address). During key derivation, the TLD will verify a laptop supplied name is in the whitelist and use it as input to the key derivation function. Even though the TLD performs zero authentication that the supplied name belongs to the laptop, this ensures that after a laptop is stolen and administrators remove the laptop name from the white-list, the thief is unable to recover the key needed to decrypt the files using the TLD.

Outside of the trusted locations, during designation of a trusted location, or when automatic key derivation fails, MULE uses a location-independent key storage and retrieval technique that uses TPMs and a secondary password (different from the user's login password) to securely manage a key that can access the sensitive files (see Section 6).

In Section 7, we discuss implementation details of the protocols, management of encryption/decryption of files for application transparency, key management such that multiple keys can access the files, and management of the automatic re-encryption of files so that a user does not unknowingly leak data.

4. HOME USER SCENARIO

Within the home setting, MULE needs a key derivation technique that "just works" once the TLD is powered on and in the trusted location. Pairing the laptop with the TLD could establish a strong shared key which could be used to secure later key derivations, but requires user effort and is vulnerable to human error [35]. As such, MULE should perform all of the tasks necessary once the laptop is in a trusted location. The simplest key derivation protocol would have the laptop derive a decryption key from a fixed m. Without access to the constrained channel, an OT will be unable to derive the key. However, an attacker in possession of the laptop could quickly brute force the key, given the limited entropy of m. Instead, the TLD can possess a strong secret to help derive the laptop's decryption key and use m to ensure the requesting laptop is in the room and protect communication over the wireless network. With Encrypted Key Exchange (EKE) [1], the TLD and the laptop can leverage m as a weak secret to establish a session key. The session key can protect the laptop's subsequent file decryption key request. Instead of using m to establish a key which is used to protect the derivation of a different key, we would like a protocol that allows the laptop with knowledge of m to successfully acquire a file decryption key with less communication and computation overhead. Since a new m is randomly generated for each run of the protocol, we can use a more efficient protocol. With blind-signatures [7], a laptop can store a secret k and use a TLD signature ($k^d \mod N$) as a decryption key, without revealing either value to the TLD or an eavesdropper. If the signature request is concealed using symmetric encryption with m as the key, the laptop with the correct m can derive the key in a single round of communication over the wireless channel. Provided symmetric encryption functions as a Pseudo-Random Permutation (PRP), a device with the wrong m (a device outside of the room) will receive seemingly random output from the TLD.

In the remainder of this section, we describe the key derivation protocol, how the laptop first associates a key with a trusted location, and an analysis of our protocol.

4.1 Home Key Derivation (HKD)

The HKD protocol consists of 4 main steps: initialization, input hiding, TLD calculations, and key recovery. During initialization, the laptop verifies it is interacting with a known TLD.[3] At the same time, the TLD generates random location-specific information (m) for use during this instance of the protocol and transmits it using the constrained channel. During input hiding, the laptop multiplies a value stored on the laptop (k) with a random number to generate a blinded request and uses m to encrypt the blinded value. The TLD's calculations include using m to decrypt the ciphertext and recover the laptop's blinded message, signing the value, and returning the result to the laptop. To recover the key needed to decrypt sensitive files (a signature on k), the laptop unblinds the signature. Figure 2 contains a summary of the HKD protocol. The goal of the protocol is to ensure that only a device with knowledge of m can successfully retrieve the correct signed value from the TLD, without revealing the laptop's long-term value k or a signature on k. If a client uses an incorrect m (the key for the cipher), the decryption at the TLD will produce the wrong value and result in a different signature.

Initialization.

When the protocol starts, the TLD sends its RSA public key (see Step 1 in Figure 2 where N is the RSA modulus and e is the public exponent). At this time, the laptop verifies it is talking to a known TLD by checking metadata stored with the encrypted sensitive files. If this TLD is unknown, the laptop considers this an untrusted location and stops key derivation (see Step 2). At the same time, the TLD randomly generates the location-specific information m of length ℓ and transmits it over the constrained channel (see Steps 3 & 4). Since the constrained channel provides a slow rate of transfer, ℓ is 20 to 30 bits to reduce the transmission time to a few seconds. Note that here we use infrared as a constrained channel, but any medium that allows the TLD to confine the transmission of m to the physical room will work.

Input Hiding.

After receiving m, the laptop uses a random number R to blind the laptop's long term secret k, and uses m to encrypt the result in Steps 5 to 7. Here, R is a random number relatively prime to N. Blinding the value hides k and temporarily conceals m. Blinding

[2]The TLD can be secured behind doors and use a wired connection to send data to microcontrollers which transmit data over the constrained channel.

[3]Section 4.2 discusses how the laptop first learns the appropriate TLD and public key for a trusted location.

Initialization:

1.	$TLD \rightarrow L$:	K^+_{TLD}	TLD responds with its public key (N,e).
2.	L :	if$(!(k = get_k(K^+_{TLD}))$ quit;	L retrieves the long-term secret for this K^+_{TLD} from its harddrive and quits if no record exists.
3.	TLD :	$m \xleftarrow{R} \{0,1\}^\ell$	TLD generates a random value of length ℓ to act as the current location-specific information.
4.	$TLD \xrightarrow{CC} L$:	m	TLD transmits the location-specific information over the constrained channel.

Input Hiding:

5.	L :	$R \xleftarrow{R} \mathbb{Z}^*_N$	L generates a random number relatively prime to N,
6.		$b \leftarrow R^e k \mod N$	uses R raised to the public exponent e to blind the long term secret,
7.		$c \leftarrow Encrypt_m\{b\}$	and uses m as a key to a symmetric cipher and encrypts the blinded value.
8.	$L \rightarrow TLD$:	c	L sends the concealed value to TLD.

TLD Calculations:

9.	TLD :	$v \leftarrow Decrypt_m\{c\}$	TLD uses m to decrypt the message
10.		$\sigma \leftarrow v^d \mod N$	and signs the value.
11.	$TLD \rightarrow L$:	σ	TLD returns the signed value.

Key Recovery:

12.		$K \leftarrow \sigma R^{-1} \mod N$	L unblinds the result to retrieve $k^d \mod N$. Note: $(R^e k)^d R^{-1} \mod N = R^{ed} k^d R^{-1} \mod N = R k^d R^{-1} \mod N = k^d \mod N$

Figure 2: The HKD protocol between a laptop (L) and a TLD.

with the random value R^e prevents the TLD (or any device with knowledge of m) from recovering k from $R^e k \mod N$. Blinding also ensures that the value encrypted is different for each protocol run. This prevents an entity outside of the room from intercepting c and recovering m from a brute-force attack. When testing a potential key for the cipher, the output of decryption appears as a random value and an attacker cannot verify if the revealed value matches the laptop's original message, thus verifying the correctness of a guess of m. In Step 7, an implementer must select a cipher such that a failed decryption does not leak information about m. If the size of the blinded value is a multiple of the cipher's block size, Electronic Code Book (ECB) mode is sufficient. However, the addition of deterministic padding to the plaintext would allow an attacker to recover m based on c (e.g., if padding is a series of 0s the attacker will try different values of m until decryption results in a plaintext ending in 0s). If the length of the blinded value is not a multiple of the cipher block size, a stream cipher or ciphertext stealing can encrypt the blinded value such that no predictable plaintext is used.

TLD Calculations.

After receiving c (Step 8), the TLD uses the correct m to decrypt the message (Step 9). If c was generated with m, the TLD has a copy of $R^e k \mod N$ (or whatever the laptop sent). If c was generated with a different key, the TLD will have a pseudo-random value that differs from the laptop's original input with high probability. After using the private exponent d to complete the blind signature, the TLD returns the result to the laptop as σ (Steps 10–11). At this point, m **is no longer location-specific information**. An attacker can take c and σ and recover m by finding the x such that $Decrypt_x\{c\} == \sigma^e \mod N$. However, learning this m is useless since the TLD uses a new random m for each run of the protocol and blinding conceals k and k^d.

Key Recovery.

Once the laptop has the TLD's response (σ from Step 11), the laptop will use $R^{-1} \mod N$ to recover the file decryption key/signature ($k^d \mod N$) from the TLD's response. Provided the laptop and TLD were using the same value for m, the laptop will now possess a deterministic signature on k (static across time) which it can use as a decryption key. If the laptop used incorrect location-specific information ($m_{Laptop} \neq m_{TLD}$), the end result will contain a signature on a pseudo-random value—the TLD signed a different value than the laptop sent—that will fail to decrypt the sensitive files.

In the remainder of this section, we discuss how our protocol allows TLD identification with zero user effort and why our protocol is secure against an OT.

4.2 Trusted Location Designation

Before using HKD, a laptop must acquire the correct public key (N,e) associated with a trusted location. Given the attacker is unable to send or receive information on the constrained channel, learning the correct public key for a trusted location requires zero user effort. When designating a location as a trusted location, the laptop generates a random 128 bit or larger k. After performing HKD with the new k and a potential public key, the laptop can verify if the device which controls the constrained channel used the public key the laptop received over the wireless channel. Given an OT is unable to access the constrained channel, this verification ensures the TLD—which controls the constrained channel—has the public key the laptop received. Once the laptop knows it is interacting with the TLD with the correct public key, the laptop saves k and the public key (N,e) as metadata with the encrypted sensitive files so future runs of HKD can access all of the necessary data. All a user has to do during designation of a new location is to recover the location-independent key (see Section 6) so the sensitive files can be decrypted using HKD or the location-independent mechanism. Section 7.3 has more details on how we manage sensitive files encrypted under multiple keys.

After a sample run, the laptop can check if it has the correct public key by verifying the signature it received from the potential TLD. If the signature is valid, there is a $1-2^{-\ell}$ chance that the public key used is the correct public key for this trusted location. There is a small probability ($2^{-\ell}$) an attacker impersonated the TLD and was able to guess m. With a correct guess of m, the attacker could correctly decrypt c and produce a valid signature for the claimed public key. For a stronger guarantee, the laptop could run HKD multiple times before accepting N,e. Given m is randomly generated independently for each run of the protocol, the laptop will detect an attack after n runs of the protocol with probability $1-2^{-n\ell}$.

4.3 Security Analysis

In this section, we discuss how HKD prevents an attacker from recovering $k^d \mod N$ (the value used as the file decryption key). Provided RSA is secure, blind signatures conceal the original message [7], m and R are random, and the block cipher used is a good pseudo-random permutation, an OT attempting to discover $k^d \mod N$ will be unsuccessful. Section 4.2 already discussed why an attacker is unable to pose as a TLD during setup and generate $k^d \mod N$ once in possession of the laptop (if the impersonation was successful, the attacker would know the private key needed to generate the signature). The remainder of this section discusses how the key remains secure when the laptop is still in the user's possession and after the OT has stolen the laptop.

While the user still has possession of the laptop, an attacker can eavesdrop on HKD to try and recover the key from messages from multiple protocol runs, or try to directly run the protocol with the TLD. After eavesdropping on one run of the protocol, an eavesdropper will have a blinded message ($R_1^e k$) and a blind signature ($R_1 k^d$). An attacker that could recover k or k^d from these messages would be able to defeat blind signatures (something infeasible assuming blind signatures conceal the original message and the attacker lacks the randomly generated R values). After eavesdropping on n protocol runs, the attacker will have n messages ($R_1^e k, R_2^e k, ..., R_n^e k$) and n signatures ($R_1 k^d, R_2 k^d, ..., R_n k^d$). Even with n pairs, the attacker is still unable to recover k or k^d given each R_i is random and the number of unknowns matches the number of equations. Without knowledge of k, direct interaction with the TLD provides little information to the attacker. As we discuss in the next paragraph, it is also infeasible to retrieve the desired signature without knowledge of the current m.

With possession of the laptop, an attacker will know k and attempt to recover $k^d \mod N$ by calculating the signature, deriving the value based on previously recorded messages, or via interacting with the TLD. If an attacker were able to generate $k^d \mod N$ without knowledge of d (or the factors of N) the attacker can compromise RSA (something infeasible assuming RSA is secure). An attacker can try to recover k^d using k and data from previously recorded runs of the protocol (e.g., the n pairs of messages ($R_1^e k, R_2^e k, ..., R_n^e k$) and signatures ($R_1 k^d, R_2 k^d, ..., R_n k^d$)). Now, the attacker can recover the various R_i^e by multiplying a message with $k^{-1} \mod N$. However, the attacker is unable to isolate k^d from the signatures. The RSA assumption dictates that given N and R_i^e (the values an attacker has) it is infeasible to recover R_i (the value the attacker needs). As such, the attacker needs a different approach to learn k^d.

If an attacker can submit k to the TLD concealed with the correct m, the attacker can recover k^d and decrypt the user's sensitive files. However, we assume an OT is unable to access the constrained channel and thus lacks knowledge of m. Instead, the attacker can try to guess m and interact with the TLD by itself or intercept communication between a legitimate user and the TLD to recover the current m. On her own, the attacker has a $2^{-\ell}$ chance of guessing the correct m for a given run of the protocol. A geometric distribution describes the probability of success after x attempts ($x-1$ failures followed by one success). As such, an attacker will need 2^ℓ attempts on average to successfully guess m. To make the attack less feasible, the TLD can rate limit requests, forcing an attacker to invest more time to perform the large number of attempts. If an attacker wants to leverage information from a legitimate run of HKD, the attacker must intercept the first message and try to determine the current m which produces the other user's blinded message $R_O^e k_O$. However, the block cipher and the random selection of R_O prevent the attacker from verifying if the guess for m was correct. Given the randomness of R_O^e and the pseudo-randomness of the cipher, the majority of decryptions look like potential legitimate messages. As such, interception of c_O provides little help to an attacker trying to recover the current m. After the TLD responds, the attacker can recover the last m from c_O and σ_O, but once the TLD responds it will use a different m for the next run of the protocol.

Without access to the constrained channel, an attacker is unable to recover the m needed to interact with the TLD and will be unable to recover the signature needed to decrypt the user's sensitive files. The best an attacker can do is recover m after the TLD responds. However, at that time the TLD will expect a different m for the next run of the protocol.

5. CORPORATE USER SCENARIO

In the corporate setting, an OT is able to access the constrained channel. If HKD was used, an OT could recover file decryption keys for stolen laptops. The company PKI and an administrator maintained list of company laptops allow us to design a protocol such that legitimate users can access sensitive files on a laptop with zero user effort, but an OT in possession of the laptop is unable to access the same data. In the remainder of this section, we discuss why the HKD fails to work in the corporate settings, how the Corporate Key Derivation (CKD) works, and why CKD is secure.

With access to the office, a malicious party can receive or send data on the constrained channel and circumvent any security provided by HKD. With the ability to receive data from the constrained channel, an attacker could return to the office after stealing a laptop and use the TLD to automatically derive the file decryption key. With the ability to send on the constrained channel, an OT could pose as a TLD during initial MULE setup, trick a laptop into deriving a file decryption key based on the attacker's RSA key pair, and generate the file decryption key once in possession of the laptop.

The Corporate Key Derivation (CKD) must authenticate the TLD, cease to work once the laptop is reported stolen, and only succeed when the laptop has access to the constrained channel. With a company PKI/trusted company public key, TLDs possess authority signed certificates to identify themselves as legitimate. A TLD certificate and Transport Layer Security (TLS) [11] can prevent an attacker from spoofing the TLD or eavesdropping on other devices' communication with the TLD. However, execution of HKD over TLS is insufficient, because an OT can steal a laptop, return to the office, and acquire the file decryption key. The TLD could authenticate laptops and only perform HKD for laptops not yet reported stolen. However, this means significant administrative overhead with per-laptop secrets on the TLD, or the addition of laptops to the company PKI and the maintenance of a Certificate Revocation List (CRL) to identify stolen laptops. Instead, the TLD uses the laptop's long-term secret and a company assigned laptop identifier (without any authentication the laptop is the one it claims to be) as input during key derivation, and refuses to derive keys for laptops that report an identifier not in the company whitelist (or quits if the laptop is present in a blacklist). The TLD uses a keyed hash/Message Authentication Code (MAC) as an efficient way to generate an output given a laptop secret and ID pair, without leaking any information about the key used in the MAC. To ensure derivation only succeeds for devices in the trusted location, the laptop xors its long-term secret with the location specific m, and the TLD xors m with the laptop's input. If a laptop uses an m that differs by one or more bits, the TLD will use the wrong input to the MAC and fail to derive the correct key. Encryption/decryption of the input using m as the key would provide the same results as xor, but requires additional computation. The Corporate Key Derivation (CKD) can protect

Initialization:

1.	$L \overset{TLS\_init}{\leftrightarrow} TLD$	: Cert'$_{TLD'}, \sigma_{TLD'}$	TLD sends a certificate and signature as part of the TLS handshake.
2.	L	: if(!verify($\sigma_{TLD'}, Cert_{TLD}$) quit	L uses the locally stored certificate to verify this is a trusted TLD.
3.	TLD	: $m \xleftarrow{R} \{0,1\}^{\ell}$	TLD generates a random value of length ℓ as the current location-specific information
4.	$TLD \xrightarrow{CC} L$	: m	and transmits the location-specific information over the constrained channel.

Apply m:

5.	L	: $x \leftarrow k \oplus m$	L xors m with the long term secret value (k) stored on the laptop
6.	$L \xrightarrow{TLS} TLD$	: ID_L, x	and sends its unique ID and the xored value to the TLD.

TLD Calculations:

7.	TLD	: $k \leftarrow m \oplus x$	TLD recovers the laptop's long term secret,
8.	TLD	: if ($ID \notin$ whitelist) quit	verifies the ID is in the whitelist,
9.	TLD	: $K_{ID_L} \leftarrow MAC_{K_{TLD}}(ID_L \| k)$	uses a keyed hash to derive the key for this ID_L, k combination,
10.	$TLD \xrightarrow{TLS} L$	: K_{ID_L}	and returns the calculated value to the laptop.

Figure 3: The CKD protocol between a laptop (L) and a TLD.

sensitive data from a laptop thief, provided the laptop is reported stolen before an OT can enter the corporate trusted location.

In the remainder of this section, we describe the CKD protocol and discuss why it is secure. Trusted location designation within CKD is similar to designation within HKD, except the laptop can verify the public key for the TLD via the corporate PKI or a public key distributed manually (e.g., IT personnel installs it with other software).

5.1 Corporate Key Derivation (CKD)

In the CKD protocol, the TLD uses TLS to prevent attackers from posing as the TLD or interfering during legitimate key derivation, xors an input with m to implicitly check if the other device is in the trusted location, and uses a MAC to derive keys. Figure 3 contains the various steps involved in CKD. The protocol is divided into three main steps: initialization, application of m, and TLD calculations.

Initialization.

The laptop initiates a TLS connection with the TLD and uses locally stored certificates or public keys to verify the TLD. At the same time, the TLD generates a random value to use as location-specific information and transmits it over the constrained channel.

Application of m.

The laptop xors the location-specific information with its long-term secret (k) as part of the implicit check that the laptop is in the trusted location. The laptop sends the result of the xor operation and the laptop's ID to the TLD.

TLD Calculations.

The TLD performs two operations to ensure only laptops with access to the constrained channel that are on the company whitelist acquire decryption keys. The TLD xors the received value with m to recover k and verifies the provided ID is in the whitelist. Provided the ID is in the whitelist, the TLD uses the TLD key (K_{TLD}) to generate the MAC of the laptop ID concatenated with k. If the laptop is outside of the room and uses the wrong m, the TLD will use the wrong k as input to the MAC. With the wrong input, the output will fail to decrypt sensitive files. If the whitelist indicates the laptop is stolen (i.e., the ID is absent from the list), the TLD quits the current run of CKD. With a MAC, the TLD can use the same secret (K_{TLD}) to securely generate different outputs for different IDs. For example, if laptop ID_1 is stolen, it is infeasible for an attacker to recover $MAC_{K_{TLD}}(ID_1||x)$ when posing as ID_2 (i.e., a secure MAC ensures the attacker is unable to find y such that $MAC_{K_{TLD}}(ID_2||y) == MAC_{K_{TLD}}(ID_1||x)$).

In Section 5.2, we discuss how secrecy of k and the use of a whitelist of valid IDs ensures attackers are unable to use the CKD to recover the keys needed to access sensitive files on a stolen laptop.

5.2 Security Analysis

The security of the laptop's decryption key K_{ID_L} relies on the security of TLS, the secrecy of the laptop's long-term secret before the laptop is stolen, and the timely update of the whitelist after the laptop is stolen. We assume TLS is secure and laptops are able to correctly identify the TLD based on secure distribution of public keys (a company CA key or a copy of the certificate for the TLD). As such we only consider how an attacker can attack the CKD to recover K_{ID_L} before or after stealing the laptop and how tunneling of m can cause the decryption of sensitive files while the laptop is still in the legitimate user's possession.

Before the laptop is stolen, an attacker is unable to recover K_{ID_L} because it lacks knowledge of the long-term secret k. Given TLS authenticates the TLD, an attacker is unable to impersonate the TLD and trick the laptop into sending k to the attacker. At this time, an attacker can pose as the laptop and interact with the TLD while using the laptop's ID. However, without knowledge of k, the attacker is unable to know what inputs will generate the correct output from the TLD. Without the encrypted files, the attacker will be unable to test a potential key and verify the guess was correct. The attacker can collect a large set of potential keys by sending a large number of requests to the TLD. However, k is a long sequence of bits (i.e., 128 bits or more) randomly generated during assignment of the office as a trusted location. The chance of an attacker correctly guessing k and thus retrieving the key from the TLD while the user still has possession of the laptop is negligible.

Once the laptop is stolen, the user will report the theft to the company's IT department which will consequently remove the laptop's ID (ID_L) from the whitelist (or add ID_L to a blacklist). Even though the attacker now knows k and can interact with the TLD, the MAC function and the secrecy of K_{TLD} ensure the attacker will be unable to acquire the key needed to access the sensitive files. Once the ID_L is removed from the whitelist, an attacker will be unable

to retrieve values from the TLD that are derived using ID_L. To learn K_{ID_L}, the attacker can recover K_{TLD} and derive the key itself or find different inputs ID_2 and k_2 which, when sent to the TLD, produce the same output. Provided the TLD is securely locked up in a physically guarded room and lacks any software vulnerabilities, the only way to recover K_{TLD} is to send inputs to the TLD and analyze the responses to recover K_{TLD}. Provided the MAC function used is secure, this type of attack is infeasible. Next an attacker could try to find a still legitimate identifier (ID_2) and an input (k_2), such that $MAC_{K_{TLD}}(ID_L||k) == MAC_{K_{TLD}}(ID_2||k_2)$. An attacker with the ability to discover such an ID_2, k_2 pair would be able to perform selective forgery of the MAC function. Assuming the MAC function is secure, selective forgery is infeasible. The only way an attacker can successfully acquire K_{ID_L} is to steal the laptop and return to the trusted location before the theft is reported. An attacker can cause a laptop still in the legitimate user's possession, but outside of a trusted location, to derive the correct key with CKD. Xoring k with the shorter location-specific information ensures that only devices with access to the constrained channel can successfully derive keys. If the laptop xors k with the wrong value, the TLD will xor with m and produce a different value as input to the MAC. If the laptop is unable to access the constrained channel the chance of accidentally guessing the correct m is $2^{-\ell}$. However, an attacker can relay information from the constrained channel in the trusted location to a location outside of the trusted location. Assuming users quickly report stolen laptops to the company, this tunneling action only accidentally reveals sensitive files to legitimate users still in possession of the laptop; the TLD will refuse to derive the correct key for a reported stolen laptop.

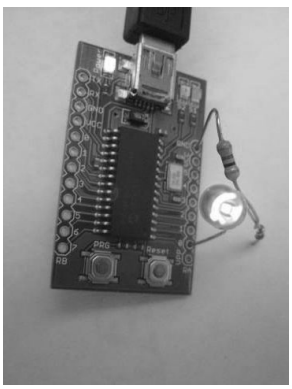
In this section, we have presented the CKD protocol and discussed why it allows the secure automatic key derivation without per laptop secrets on the TLD, provided the laptop is reported stolen before an attacker can access the trusted location.

6. LOCATION-INDEPENDENT KEY

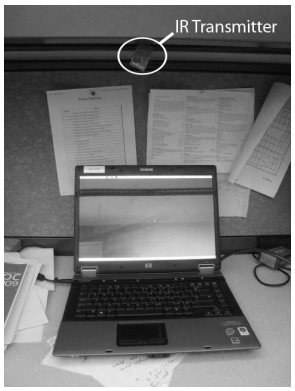
Users outside of a trusted location may need access to sensitive data. Given these accesses are infrequent, we assume users will accept the solution to require some interaction and time to maintain a high level of security. When outside of a trusted location, we use a secondary password – one different than the user's login password – to retrieve a location-independent key. The simplest solution is to use the password itself as the key. However, an attacker can brute force a password in a matter of hours. Similar to Bitlocker [19], we use the laptop's TPM to bind [33] a key based on the user's password. Here we discuss how MULE uses the user's password and the TPM to protect the location-independent key.

During installation, MULE generates and encrypts the location-independent key (K_{Ind}) that is later un-bound whenever the user accesses sensitive files outside of any trusted location. When the user first installs MULE, the system generates a random K_{Ind} and takes as input from the user a secondary password. Next, MULE asks the TPM to generate a non-migratable asymmetric key pair $(K^{+}_{MULE}, K^{-1}_{MULE})$ (i.e., the key is only accessible on this TPM) and require the user's secondary password to permit operations which utilize the secret key. The TPM can encrypt K_{Ind} without requiring the password. MULE stores the encrypted K_{Ind} in the user's home directory until the user tries to access protected files and HKD or CKD fail to derive a key. At that time, MULE asks the user for the secondary password. MULE passes the entered password and the encrypted copy of K_{Ind} to the TPM. In response, the TPM will only permit calculations which use K^{-1}_{MULE} and decrypt and return K_{Ind} when given the correct secondary password.

Without the secondary password, an OT would need to invest a significant amount of time (e.g., decades) to access user's sensitive files. To access files using the location-independent key, an OT needs to guess the password or one of the keys. We assume K_{Ind} is at least a 128-bit randomly generated value and the TPM uses a 2048 bit RSA private key so it is computationally infeasible for attackers to guess either of the keys. If an OT tries to guess the user's secondary password, the TPM's built-in guessing attack defenses and the fact that only that TPM can use K^{-1}_{MULE} and decrypt the data (i.e., the attack is non-parallelizable) will prevent the attacker from accessing the files for several decades on average (see Section 8.2 for more details about guessing attacks against the TPM used in our implementation).



(a) IR transmitter



(b) An Example Trusted Location

Figure 4: The IR transmission device and the desk setup as a trusted location.

7. IMPLEMENTATION

We have implemented MULE on a HP 6730b with a 2.4 GHz Intel Core 2 Duo processor and 2GB of RAM running Ubuntu 2.6.28 that uses EncFS[4] to store sensitive data as encrypted filesystems. For a TLD, we use a Dell Optiplex 755 with a 3.2 GHz Intel Core 2 Duo processor with 4GB of RAM connected to a Universal Bit Whacker[5] with an IR LED (see Figure 4(a)). All of the cryptography involved in the protocols was performed using OpenSSL. We use AES with a 128 bit key (m followed with 0s) as a cipher for HKD. HKD signature generation uses 2048-bit RSA with TLD side blinding to prevent timing attacks [3]. The MAC in CKD was implemented using HMAC with SHA1. TLS in CKD uses ephemeral Diffie-Hellman with 2048-bit RSA authentication during setup with AES256 and SHA1 to protect communication. We use video for Linux two (V4L2) to directly capture frames from the laptop's webcamera. As a trusted location, we attached the IR transmitter to the underside of a bookshelf on a desk, pointing towards the back of the desk (see Figure 4 (b)). In this setup, the laptop can only see the LED (i.e., be in the trusted location) while the laptop is open and on the desk. In the remainder of this section, we discuss how our implementation of MULE transmits data over the IR channel, manages encrypted copies of sensitive files under multiple keys, provides automatic access to encrypted files, and protects users who forget to close sensitive files.

[4] `http://www.arg0.net/encfs`

[5] `http://www.schmalzhaus.com/UBW/index.html`

7.1 Transmission of Data Over the IR Channel

Our constrained channel uses a simple on/off encoding with two frames per bit (i.e., LED brightness above a threshold for 2 consecutive frames means 1). The laptop performs 3 steps to capture a message over the constrained channel: locate the position of the LED, tell the TLD to begin transmission, and decode the message. When the TLD is not transmitting a message, the LED repeatedly transmits the sequence 1010 to help the laptop determine the position of the LED. Once the camera is on, the laptop records six frames and looks for pixels that follow an alternating on-on-off-off pattern. Any pixels that match the pattern are considered the LED and are used to build a mask such that any other pixels are ignored. With the mask determined, the laptop tells the TLD to begin transmission. The TLD generates a 20 bit random sequence and prepends 0101 as the header. The laptop records the output from the masked images (i.e., ignoring any non-LED pixels), looking for the 0101 start sequence followed by 20 bits for m and quitting once the LED returns to transmitting 1010.

7.2 Location Independent Key Implementation

Our laptop includes an Infineon SLB 9635 TT v1.2 TPM which allows our implementation to protect the RSA decryption key with the user's password such that a party needs that TPM and the password to decrypt the location-independent key. Our implementation uses the TCG Software Stack (TrouSerS)[6] to interact with the TPM and our own code to manage the encrypted location-independent key (K_{Ind}).

7.3 Multiple Keys & Encrypted Filesystems

In MULE, a single encrypted filesystem is protected under the location-independent key (K_{Ind}) and at least one location-dependent key ($K_{Loc_1}, K_{Loc_2}, ...$). Knowing one of the keys should grant access, but the different keys produce different cipher text (thus different values). Storing multiple copies of the filesystem, each encrypted under a different key, wastes space and can lead to outdated information when one copy is changed and the keys needed to access the other copies are unknown. Our approach is to randomly generate a filesystem key (K_{FS}) to encrypt the filesystem. MULE uses OpenSSL to encrypt copies of K_{FS} under the location-independent key and any relevant location-dependent keys (i.e., $\{K_{FS}\}_{K_{Ind}}$, $\{K_{FS}\}_{K_{Loc_1}}$, $\{K_{FS}\}_{K_{Loc_2}}, ...$). Once MULE acquires any of the keys, it can decrypt K_{FS} and mount the sensitive files using EncFS. MULE also stores any location relevant information (the public key for HKD, the certificate for CKD, and various long-term secrets) with each encrypted key to simplify key derivation (e.g., identify the TLD needed to derive the key to decrypt K_{FS}).

7.4 Encrypted Filesystem Management

One of the major goals in this work is to reduce user overhead associated with accessing encrypted files. This includes both mounting the encrypted filesystem when the user tries to access sensitive files and unmounting the filesystem to ensure no sensitive information is leaked when the laptop is lost.

7.4.1 Mounting the Filesystem

To make accessing the encrypted filesystem unobtrusive, we replace the folder that contains the decrypted files (i.e., the mount point of the encrypted filesystem) with a MULE script. When a user double clicks on the "folder", our script runs, generates a key for EncFS, overwrites the script with the mounted filesystem, and opens the filesystem in a new window. When using a terminal, a user must execute the "folder" rather than simply change directories (i.e., "`cd ~/; ./sensitiveFolder`" rather than"`cd ~/sensitiveFolder`").

7.4.2 Unmounting the Filesystem

To protect data, we need to ensure that the encrypted filesystem is unmounted before an OT can steal the laptop. If the encrypted filesystem is left mounted when the laptop is put to sleep, an OT could steal the laptop, wake up the system, and access the files without having to discover a key. We could leave unmounting the filesystem up to the user. However, it is dangerous to assume a user will remember to unmount a filesystem that is automatically mounted. To prevent leaks, MULE installs a number of scripts to automatically unmount the filesystem, re-insert the scripts which perform automatic mounting, and closes applications accessing sensitive files when the user logs off, has been idle for too long, or the laptop goes to sleep (e.g., the user closes the laptop and leaves the trusted area). We recognize that automatically closing applications negatively impacts usability. Fortunately, auto-save functionality will help reduce the loss of data due to unexpected application termination. In addition to a script launched at log-off, our implementation replaces the Gnome screen-saver with a copy of our script and places a copy of the script in `/usr/lib/pm-utils/sleep.d/`. Respectively, these scripts ensure that, once the screen-saver starts (i.e., the user has been idle) or the machine sleeps or hibernates, the sensitive files will be unmounted.

8. EVALUATION

In this section, we evaluate the performance of our implementation. We first discuss the amount of time from when a user clicks on an encrypted filesystem and when the filesystem is mounted and displayed in a new window. We also discuss how our implementation performs when an attacker attempts to guess the secondary password associated with the location-independent mechanism.

8.1 Time to Mount an Encrypted FS

Table 1 contains the time in milliseconds for various operations associated with each protocol. The values here represent the average and standard deviation across 20 runs of each protocol. In the remainder of this section, we discuss a number of results. The home protocol is faster than the corporate protocol due to less computation. The location-independent mechanism provides the fastest key recovery mechanism. The performance is slower than expected for some operations because we are using a bash script to call a number of different programs. Finally, our key derivation protocols require almost 5 seconds, but our constrained channel limits the potential speedup.

When comparing the performance of the HKD and CKD protocols we find that the home protocol is faster due to less computation. The major differences are the result of the connection setup and the computation performed by the TLD. HKD has a faster setup since TLS is not used. However, HKD requires more computation for the TLD because key derivation involves signature generation. With our current setup, both the blind signature and TLS use 2048-bit RSA keys. However, with TLS the keys are used to sign ephemeral Diffie-Hellman values which are then used to establish a shared key. This connection setup takes roughly 141ms versus the 10ms in HKD. During actual key derivation, HKD is slower since the TLD must generate a signature as opposed to the symmetric operations associated with TLS and the MAC-based key derivation.

[6] `http://trousers.sourceforge.net`

Operation Average Standard Deviation	Home Delay in ms (σ)	Corporate Delay in ms (σ)	Loc-Ind Delay in ms (σ)
Connect to TLD	10.40 (3.69)	140.72 (8.6)	11.71 (4.671)
Film Sequence	4329.85 (17.12)	4378.99 (42.781)	1671.19 (39.518)
Decode Sequence	30.24 (5.33)	41.01 (8.58)	—
Calculations & Send Data	2.13 (0.05)	1.96 (0.09)	—
TLD Operations	47.62 (15.54)	19.54 (14.45)	—
Unblind Result	0.06 (0.002)	—	—
Close TLS	—	5.36 (3.48)	—
TPM-based Decryption	—	—	981.5 (5.01)
Decrypt K_{FS}	9.57 (0.13)	9.66 (0.18)	9.63 (0.15)
Mount FS	10.92 (0.26)	10.90 (0.30)	11.1 (0.25)
Open Window	78.3 (2.5)	79.2 (2.7)	76.8 (2.3)
Total	4627.6	4803.6	2806.6

Table 1: Average time and standard deviation for various operations for HKD, CKD, and the Location Independent Mechanism (after HKD fails)

There are two reasons why the location-independent mechanism is fastest: the constrained channel has long setup times and a slow transmission rate and we ignore the time associated with entering the password. For the home and corporate protocols, the majority of the time associated with key derivation is "Film Sequence" which includes video initialization, detecting the LED, and capturing the sequence. Video initialization takes on average 1212.8 ms. Once the program is able to capture frames, the system records 11 frames—5 to allow for automatic brightness adjustment and 6 for LED detection—in 466 ms. The system then takes on average 2706 ms to record the next 60 frames or 30 bits at 2 bits/frame. After filming these frames the next 30 ms are used to decode the 20 bit sequence m. In total, the use of IR and a webcam as a constrained channel consumes almost 4.5 seconds. In comparison, the location-independent mechanism spends 1208 ms initializing video and 518 ms capturing frames to determine no LED exists (1.7 of the total 2.8 seconds). During evaluation, our script read the secondary password from a file to remove human variability from the results. However, spawning a password entry window and manually entering the secondary password would add significant time to the location-independent protocol that is highly dependent on the user's ability to quickly remember and type in the secondary password.

By using a bash script to call a number of separate programs some operations take longer than expected. Specifically, the script calls our programs which perform the home or corporate protocol, our TPM-based decryption program if automatic key derivation fails, the OpenSSL command line tool to decrypt K_{FS}, EncFS to mount the filesystem, and the file browser Nautilius to open a window with the mounted filesystem. As a result, a number of generally fast operations contribute considerable overhead. For example, decrypting K_{FS} requires a single AES decryption, but takes almost 10 ms in our current implementation.

Overall, the automatic key derivation schemes can provide access to files in less than 5 seconds. This does seem like an exceptionally long time, but is only incurred during the first access to a set of sensitive files. Once the files are mounted, the only additional overhead is a result of using EncFS. The majority of that time is associated with receiving the location-specific information. If we were to use a different constrained channel or use an IR receiver rather than a webcam, we could increase the transmission rate and speed up the protocol while maintaining the same security level. However, these other options require additional hardware for the laptop, a requirement that contradicts some of our initial design goals.

8.2 Attacks on the Location-Independent Password

For the location-independent mechanism to remain secure, the TPM must implement some type of defense against guessing the password. Prior works have shown only some TPMs have such a defense [27] so we wanted to evaluate what defenses were present on the Infineon TPM. To test the defenses, we sent a series of decryption requests to the TPM with the wrong password and measured how long a guess took. We also periodically sent a correct password to determine how that impacted the defense.

Our test found that on average testing a wrong password required 626ms. After a single wrong password, the TPM entered a lockout period where it refused to respond to even the correct password for more than 2 minutes. Even after removing the battery and power from the laptop, the defense continued to ignore requests. This represents positives and negatives with respect to MULE. With such strong TPM defenses, attacking a limited entropy secondary password will take a long time. For example, consider an attacker trying to guess the password. An 8 character user-selected password has on average 24 bits of entropy [4]. With a modern computer that can perform half a million guesses a second (2^{19}), it would take 2^{23} guesses on average or 2^4 seconds (less than one minute) to recover the password. With the Infineon TPM, an attacker can only perform roughly one guess every 2^7 seconds and requires roughly 2^{30} seconds or ≈ 34 years to discover the secondary password. However, these defenses will also impede a user who has trouble recalling a password or accidentally mistypes a password.

9. DISCUSSION

If we use a non-migratable binding key on the TPM to encrypt a file decryption key, the current TPM is the only TPM that can access the key. If this TPM were to fail, the user would be unable to access files outside of trusted locations. Users should back up files in case of laptop loss in a way that does not rely on the laptop or the laptop's TPM. With a backup, the user can copy sensitive files to a new laptop and use the new TPM and a new location-independent key.

For both HKD and CKD, TLD secrets are needed to derive keys. To ensure continued operation of the location-based protocols, users or company IT personnel could copy the secrets onto some other medium. If a TLD ceases to function, users could copy the secrets to a new TLD. Home users could simply store the TLD secrets on a USB drive in the trusted location.[7] IT personnel could store the data in a company safe behind locked doors so physical security could prevent unwanted access.

[7]Without access to the location, an OT is unable to access the contents of the USB drive.

10. RELATED WORK

This is the first work we are aware of that balances usability and security by using location-specific key derivation. However, several prior works have examined the related problems of securing data on stolen devices, pairing (i.e., securing communication between two devices), and location verification.

Encrypting files is a common solution to the stolen laptop problem [9, 19, 29]. We desire a technique to store the key without burdening the user or company personnel or requiring extra laptop hardware. Users are willing to accept the task of password entry to act as a key, but often use relatively weak passwords. Once a laptop is in the attacker's possession, an attacker can brute force passwords to discover the key. In addition to passwords, prior works on user authentication utilize what the user is (biometrics) or what the user has (tokens). Users cannot forget their biometrics, but most systems require additional user interaction (e.g., speech or finger prints [20, 34]) and extra hardware for biometric entry (e.g., fingerprint reader). Some mechanisms also require the system to store a template used to verify that the appropriate biometric was entered, rather than deriving a key from the biometric. Often the template is encrypted on the hard drive and requires an additional key to decrypt the template [13]. Cryptographic tokens provide greater entropy than user's passwords [9, 26], but users may keep the token with the laptop to ensure effortless access [22]. Once an attacker steals the laptop with the cryptographic token, the attacker has access to all of the files. Another approach is to use an online service to store keys for a user [16, 26]. Unlike our CKD, these schemes use per-laptop secrets and require significantly more administrative overhead, reducing the chance of widespread adoption. In addition, when the laptop is offline, for example during a flight, users cannot access their files. Related to harddrive encryption is how to securely store and erase keys such that an OT is unable to recover the key from memory [12]. Once MULE derives the keys to grant users access, appropriate defenses are needed to ensure once the filesystems are unmounted an OT is unable to retrieve the keys or other sensitive data from memory.

In HKD and CKD, a secondary device (the TLD) is used to help derive keys. A number of prior works examine how two devices can establish secure communication [5, 15, 17, 28, 31]. Rather than using location-specific information and HKD or CKD, the laptop could initially pair with a device and use that device as a key escrow. However, this approach would only guarantee that the laptop is in the trusted location during the initial pairing and requires the device to store per-laptop secrets. Once paired, the schemes would have to perform a partial "re-pairing" which verifies the laptop is still in the trusted location. In addition, many of these schemes require user effort to initially pair (e.g., comparing strings, connecting two devices, or checking that no other devices are within range) and/or require additional laptop hardware (e.g., highly accurate timers to measure wireless time-of-flight). HKD and CKD allow the TLD to maintain limited state (at most a secret key and a whitelist of devices) and combine the step of location verification with key derivation without any user effort.

As presented, MULE relies on a weak verification of a location claim based on constrained channels [14]; if the laptop can receive m and interact with a known TLD, the laptop is in a trusted location. Other works have proposed schemes which provide stronger guarantees about location verification based on the ability to quickly respond to challenges [2] or by comparing the reception times at multiple authoritative receivers [6, 32]. These approaches would improve the security of MULE by verifying a device is within a certain radius or physical space surrounding the TLD. However, these stronger verification claims require one or more highly accurate timers to measure the time-of-flight of wireless messages and thus are not cheaply implemented with the radios on today's laptops.

11. CONCLUSION

Users and corporate IT personnel want security solutions that simply work and want to avoid any schemes that require additional effort or administrative overhead. In this work, we designed Mobile User Location-specific Encryption (MULE), a system that requires zero user effort and limited IT administration in the common case. MULE remains secure when facing an Outsider Thief (OT), our model of a laptop thief. Based on the observation that the majority of accesses to sensitive documents occur while located in a trusted location, we designed the Home Key Derivation and Corporate Key Derivation protocols which allow a laptop to automatically derive the key needed to access sensitive files based on a location. For example, with MULE, a user can securely store encrypted copies of bank records and tax returns on a laptop, and automatically gain access when opening those files in the home office. After a thief steals the laptop, the only way to recover the files is to break into the user's home. Given physical copies of various sensitive files already exist in the home, such an invasion presents a loss of data, independent of possession of the stolen laptop. In a corporate setting, an IT administrator only has to remove a stolen laptop's ID from a whitelist of not-yet stolen laptops to ensure the sensitive data remains encrypted. We have implemented MULE on commodity hardware and found that it provides automatic protection of sensitive files with limited delay during the initial access (i.e., less than 5 seconds to automatically derive the key and decrypt the files).

12. REFERENCES

[1] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, 1992.

[2] S. Brands and D. Chaum. Distance-bounding protocols. In *Proceedings of Advances in Cryptology (EUROCRYPT)*, 1993.

[3] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of USENIX Security Symposium*, 2003.

[4] W. E. Burr, W. T. Polk, and D. F. Dodson. Recommendation for electronic authentication. Special Publication SP 800-63, NIST, 2004.

[5] M. Cagalj, S. Capkun, and J.-P. Hubaux. Key agreement in peer-to-peer wireless networks. *IEEE (Special Issue on Cryptography)*, 2006.

[6] S. Capkun and J. Hubaux. Secure positioning of wireless devices with application to sensor networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[7] D. Chaum. Blind signatures for untraceable payments. In *Proceedings of Advances in Cryptology (Crypto)*, 1982.

[8] CNN. Agency chief: Data on stolen va laptop may have been erased. `http://www.cnn.com/2006/US/06/08/vets.data/`, 2006.

[9] M. D. Corner and B. D. Noble. Zero-interaction authentication. In *Proceedings of ACM Conference on Mobile Computing and Networking (MobiCom)*, 2002.

[10] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. Geo-rbac: A spatially aware rbac. *ACM Transactions on Information and System Security*, 2007.

[11] T. Dierks and C. Allen. The TLS protocol version 1.0. Technical Report 2246, IETF, Jan. 1999.

[12] J. A. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. Felten. Lest we remember: cold boot attacks on encryption keys. In *Proceedings of USENIX Security Symposium*, 2008.

[13] A. Jain, A. Ross, and U. Uludag. Biometric template security: Challenges and solutions. In *Proceedings of European Signal Processing Conference (EUSIPCO)*, 2005.

[14] T. Kindberg, K. Zhang, and N. Shankar. Context authentication using constrained channels. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, 2002.

[15] Linksky, J. et al. Simple Pairing Whitepaper, revision v10r00. `http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`, August 2006.

[16] P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. In *Proceedings of IEEE Symposium on Security and Privacy*, 2001.

[17] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proceedings of IEEE Symposium on Security and Privacy*, 2005.

[18] R. Mears and L. Ponemon. Enterprise@risk: Privacy & data protection survey. `http://www.deloitte.com/dtt/article/0,1002,cid=182733,00.html`, 2007.

[19] Microsoft. BitLocker drive encryption: Technical overview. `http://technet.microsoft.com/en-us/windowsvista/aa906017.aspx`.

[20] F. Monrose, M. K. Reiter, Q. Li, and S. Wetzel. Cryptographic key generation from voice. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, 2001.

[21] OpenSSL. The OpenSSL project. `http://www.openssl.org`.

[22] A. Papadimoulis. Security by oblivity. `http://thedailywtf.com/Articles/Security_by_Oblivity.aspx`.

[23] Privacy Rights Clearinghouse. A chronology of data breaches. `http://www.privacyrights.org/ar/ChronDataBreaches.htm`.

[24] K. Regan. No end in sight: Data breach tally approaches 100 million. `http://www.technewsworld.com/story/53222.html`.

[25] B. Rosenberg. Chronology of data breaches 2006: Analysis. `http://www.privacyrights.org/ar/DataBreaches2006-Analysis.htm`.

[26] RSA Labortories. RSA SecurID. `http://www.rsa.com/rsalabs/node.asp?id=1156`.

[27] A.-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. TCG inside?: a note on TPM specification compliance. In *Proceedings of ACM workshop on Scalable trusted computing (STC)*, 2006.

[28] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Proceedings of IEEE Symposium on Security and Privacy*, 2006.

[29] SecureStar. DriveCrypt: Disk encryption and data encryption software. `http://www.securstar.com/products_drivecrypt.php`.

[30] K. Small. Data breaches caused by human error, hardware theft. http://www.itnews.com.au/News/87188,data-breaches-caused-by-human-error-hardware-theft-survey.aspx.

[31] F. Stajano and R. J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols Workshop*, 1999.

[32] N. O. Tippenhauer and S. Capkun. Id-based secure distance bounding and localization. In *Proceedings of European Symposium on Research in Computer Security (ESORICS)*, 2009.

[33] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 103, Trusted Computing Group, July 2007.

[34] U. Uludag, S. Pankanti, and A. K. Jain. Fuzzy vault for fingerprints. In *Proceedings of International Conference on Audio- and Video-Based Biometric Person Authentication (AVBPA)*, 2005.

[35] E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Proceedings of Usable Security Workshop (USEC)*, 2007.