Bump in the Ether: A Framework for Securing Sensitive User Input

Jonathan M. McCune Adrian Perrig Michael K. Reiter
Carnegie Mellon University
{jonmccune, perrig, reiter}@cmu.edu

Abstract

We present Bump in the Ether (BitE), an approach for preventing user-space malware from accessing sensitive user input and providing the user with additional confidence that her input is being delivered to the expected application. Rather than preventing malware from running or detecting already-running malware, we facilitate user input that bypasses common avenues of attack. User input traverses a *trusted tunnel* from the input device to the application. This trusted tunnel is implemented using a trusted mobile device working in tandem with a host platform capable of attesting to its current software state. Based on a received attestation, the mobile device verifies the integrity of the host platform and application, provides a trusted display through which the user selects the application to which her inputs should be directed, and encrypts those inputs so that only the expected application can decrypt them. We describe the design and implementation of BitE, with emphasis on both usability and security issues.

1 Introduction

Using security-sensitive applications on current computer systems exposes the user to numerous risks. User-level malware such as keyloggers, spyware, or Trojans often monitor and log every keystroke. Through keystrokes, an adversary may learn sensitive information such as passwords, bank account numbers, or credit card numbers. Unfortunately, current computing environments make such keystroke logging trivial; for example, X-windows allows any application to register a callback function for keyboard events destined for any application. Giampaolo created a simple 100-line C program that is able to capture keyboard input events that the user intended for some other application under X11. Similar vulnerabilities exist in Microsoft Windows, e.g., `RegisterHotKey` and `SendInput` can be used in combination from an application that does not currently have input focus to capture user input to the application with input focus. We conclude that it is desirable to reduce the involvement of the window manager in sensitive I/O activities as much as possible.

Besides the ease of eavesdropping on keystrokes, another serious risk to the user is the integrity of screen content. Malicious applications can easily overwrite any screen area. This introduces the threat that a user cannot trust any content displayed on the screen since it may originate from a malicious application. Additionally, malicious software that can capture the screen content of running applications may be able to extract valuable secrets (e.g., the user is filling out an electronic tax form with banking software).

In such an environment, it is challenging to design a system that provides the user with guarantees that the correct operating system and the correct application are currently running, and that only the correct application will receive the user's keystrokes. In particular, we would like a computing environment with the following properties:

- The user obtains user-verifiable evidence that the correct OS and the correct application loaded.
- The user obtains user-verifiable evidence that only the correct application is receiving keystroke events.

We designed and prototyped Bump in the Ether (BitE), a system that proxies user input via a trusted mobile device, circumventing the window manager and traditional input paths via a user-verifiable trusted tunnel. We name the system Bump in the Ether because part of the inspiration for this work came from devices referred to as "bumps in the cord", which were devices used to "scramble" conversations on plain old telephone service (POTS) in an effort to foil eavesdroppers.

Tunnels in BitE are end-to-end encrypted, authenticated tunnels between a trusted mobile device and a particular application on the user's host platform. Figure 1 shows a comparison of the legacy input path versus input through a trusted tunnel. To reduce the user's need to trust the window manager, we use the display on a trusted mobile device as a trusted output mechanism. We design an OS module that directly passes sensitive keystrokes from the user's mobile device to the correct application, bypassing the X-windowing system.

We assume the user's computing platform is capable of attesting to its current software state. For the remainder of this paper, we assume the user's computing platform is equipped with a Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG), and that the BIOS and OS are TPM-enabled and perform integrity measurements of code loaded for execution [26, 37]. The user's mobile device is used to verify these measurements.

Trusted tunnels use per-application cryptographic keys which are established during an application registration phase. The process of establishing a trusted input

session over the trusted tunnel is contingent on the mobile device's successfully verifying an attestation from the integrity measurement architecture (IMA [26]) and TPM on the user's host platform, as well as the user selecting the correct application from a list presented by her mobile device.

2 Background and Related Work

We provide background and review related work on mobile devices, secure window managers, and trusted computing primitives.

2.1 Mobile Devices

Balfanz and Felten explored the use of hand-held computing devices (e.g., PDAs) as smart-cards, and found some advantages because the user can interact directly with the hand-held for sensitive operations [1]. The authors generalized their work into a design paradigm they call *splitting trust*, where a smaller, trusted device performs security-sensitive operations and a large, powerful device performs other operations. BitE can be considered a system designed in accordance with the principles of splitting trust.

The Pebbles project attempts to let handhelds and PCs work together when both are available, as opposed to the conventional view that handhelds are used when PCs are unavailable [20]. BitE uses a trusted mobile device to help improve input security on a PC.

Ross et al. develop a framework for access to Internet services, where both the sensitivity of the information provided by the service and the capabilities of the client device are incorporated [25]. This framework depends on either a trusted proxy infrastructure or service providers running a trusted proxy. While promising, this scheme is not widely deployed today.

Sharp et al. develop a system for splitting input and output across an untrusted terminal and a trusted mobile device [31]. Applications run on a trusted server or on the mobile device itself, using VNC [23] to export video to the trusted and untrusted displays in accordance with a security policy. The user has the ability to decide on the security policy used for the untrusted keyboard, mouse, and display. An initial user study yielded encouraging results, but this technique is best described as a tool for power users. In contrast, BitE is designed for interaction with applications running on a local workstation, and for users who may have very little understanding of computer security.

Sharp et al. propose an architecture for fighting crimeware (e.g., keyloggers and screengrabbers) via split-trust web applications [30]. Web-applications are written to support an untrusted browser and a trusted mobile device with limited browsing capabilities. All security-critical decisions are confirmed on the mobile device. This architecture raises the bar for web-based attackers, but it also raises usability issues which are the subject of future work. BitE is designed to improve the security of interaction between a user and applications on her local workstation. While that application can be a web browser, BitE does not specifically address web security issues.

2.2 Secure Window Managers

A goal of BitE is to ensure that only the correct application is receiving input events, and to provide user-verifiable evidence that this is so. While much prior work has addressed this issue, none of it is readily available for non-expert users on commodity systems today. We now review related work chronologically.

Several government and military computer windowing systems have been developed with attention to security and the need to carefully isolate different grades of information (e.g., classified, secret, top secret). Early efforts to secure commercial window managers resulted in the development of *Compartmented Mode Workstations* [5, 6, 11, 22, 24, 38], where tasks with different security requirements are strictly isolated from each other. These works consider an operating environment where an employee has various tasks she needs to perform, and some of her tasks have security requirements that necessitate isolation from other tasks. For example, Picciotto et al. consider trusted cut-and-paste in the X window system [21]. Cut-and-paste is strictly confined to allow information flow from low-sensitivity to high-sensitivity applications, so that high-sensitivity information can never make its way into a low-sensitivity application. Epstein et al. performed significant work towards trusted X for military systems in the early 1990s [8, 9, 10]. While these systems are effective for employees trained in security-sensitive tasks, they are unsuitable for use by consumers.

Shapiro et al. propose the EROS Trusted Window System [29], which demonstrates that breaking an application into smaller components can greatly increase security while maintaining very powerful windowing functionality. Unfortunately, EROS is incompatible with a significant amount of legacy software, which hampers widespread adoption. In contrast, BitE works in concert with existing window managers.

Microsoft's Next-Generation Secure Computing Base (NGSCB) proposes encrypting keyboard and mouse input, and video output [18]. In NGSCB, special USB keyboards encrypt keystrokes which pass through the regular operating system into the *Nexus*, where they are decrypted. Once in the Nexus, they can be sent to a trusted application running in Nexus-mode, or they can be sent to the legacy OS. Applications running in Nexus-mode have the ability to take control of the system's primary display, which was designed to be useful for establishing a trusted tunnel.

Common to the majority of these schemes is a mechanism by which some portion of the computer's screen is trusted. That is, an area of the screen is controlled by

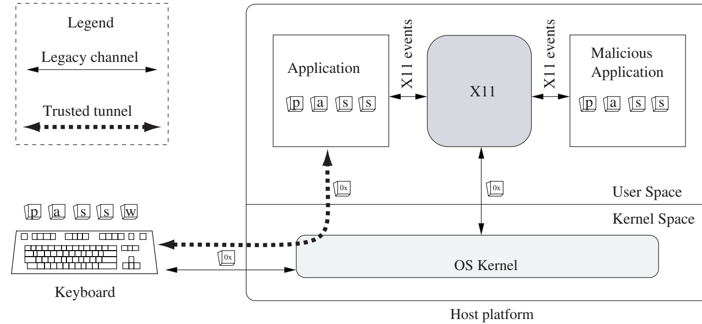

Figure 1: Traditional flow of keystrokes vs. trusted tunnels. On a traditional computer system, keystrokes are first sent to the OS kernel, which passes them to X-windows, which then sends keyboard events to all applications that register to receive them. Unfortunately, malicious applications can register a callback function for keyboard events for other applications. Our trusted tunnels protect keystrokes and only send them to the desired application.

some component of the trusted computing base (TCB) and is inaccessible to all user applications. However, if an application can use a "full-screen" mode, it may be able to spoof any trusted output. Precisely defining trusted full-screen semantics that a non-expert user can operate securely is, to the best of our knowledge, an unsolved problem. Considering the value that the user receives from being able to maximize applications, and the role of multi-media applications on today's commodity PCs, we believe the ability to run applications in full-screen mode on the system's primary display is an indispensable feature. Still, there is no effective way to establish a trusted tunnel if there is no trusted display. Due to the complexity of X and the likely confusion of untrained users, it is difficult to implement a trusted screen area in an assurable way. BitE uses the trusted mobile device's screen—a physically separate display—as a trusted output device.

We emphasize that, despite the large body of work on trusted windowing systems, the majority of users do not employ any kind of trusted windowing system. Thus, we proceed under the assumption that users do not want to change their windowing system. In the remainder of the paper, we show that BitE can increase user input security under these conditions.

2.3 Trusted Computing Primitives

BitE leverages two features of the user's TPM-equipped computing platform: attestation and sealed storage. To enable these features, the platform's OS must be equipped with an integrity measurement architecture (IMA). The values resulting from integrity measurement are used in attestations and in access control for sealed storage. The remainder of this section provides more detail on these mechanisms.

TPMs have 16 platform configuration registers (PCRs) that an IMA can extend with *measurements* (typically cryptographic hashes computed over a complete executable) of software loaded for execution. The IMA extends the appropriate PCR registers with the measurement of each software executable just before it is loaded.

TPMs can generate a Storage Root Key (SRK) that will never leave the chip. The SRK enables *sealed storage*, whereby data leaving the TPM chip is encrypted under the SRK for storage on another medium. Data can be sealed with respect to the values of certain PCR registers, so that the unsealing process will fail unless the same values are present that were present during the sealing process. Several other keys are maintained by the TPM and kept in sealed storage when not in use. One of these is the Attestation Identity Key (AIK), which is an RSA signing keypair used to sign attestations. To the remote party trying to verify the attestation, the public AIK represents the identity of the attesting platform.

An *attestation* produced by the IMA and TPM consists of two parts: (1) a list of the measurements of all software loaded for execution, maintained by the IMA functionality in the OS; and (2) an AIK-signed list of the values in the PCR registers, called a PCR *quote*. A remote party with an authentic copy of the public AIK can compute the expected values for the PCR registers based on the measurement list, and check to see whether the signed values match the computed values. The end result is a *chain* of measurements of all software loaded since the last reboot. The security requirement is that all software is measured before being loaded for execution.

Sailer et al. developed an open-source IMA for Linux [26]. They show that it is difficult to manage the integrity measurements of a complete interactive computer system, since there will be hundreds or even thousands of measurements in the course of a normal system's uptime, and the order in which applications are executed is reflected in the resulting PCR values. This ordering is not a problem for attestation, since the measurement list can be validated against the PCR values. However, it is an issue for sealed storage, since data is sealed with respect to PCR values, and not particular items on the measurement list. Thus, to access data in sealed storage, not only must the same software be loaded, but it must have been loaded in the *same order*.

During the boot process, however, a well-behaved system always loads in the same order. Hence, integrity measurement of the system from boot through the loading of the kernel, its modules, and system services loaded in a deterministic order will be consistent across multiple boot cycles on a well-behaved host platform.

One problem with IMA as described is that integrity measurements are made at *load-time*. Thus, run-time vulnerabilities may go undetected if malicious parties can exploit the difference between time-of-check and time-of-use (so-called TOCTOU attacks). IMA can be used in combination with systems which provide run-time attestation [32] or verifiable code execution [28] to achieve even stronger platform security guarantees.

3 Bump in the Ether

We provide a brief overview of BitE and present the assumptions under which it operates. We then discuss the necessary setup which must take place before BitE can be used. We describe the use of BitE to secure user input in Section 4.

3.1 System Overview

The primary goal of BitE is to enable end-to-end encrypted, authenticated input between the user's trusted mobile device and an application running on her host platform. BitE is built around a trusted mobile device that can proxy user input, show data on its own display, efficiently perform asymmetric and symmetric cryptographic operations, and store cryptographic keys. This trusted device runs a piece of software called the BitE Mobile Client. The BitE Mobile Client verifies attestations from the host platform, manages cryptographic keys used in establishing trusted tunnels for user input, and provides a trusted output mechanism which can inform the user of security-relevant events.

We summarize the two setup steps and common usage for BitE.

1. Building an association between the trusted mobile device and the host platform (Section 3.3.1). This is performed once for each trusted mobile device and host platform pair.
2. Registering applications on the host platform for use with BitE (Section 3.3.2). This is performed once for each registered application on a particular trusted mobile device and host platform pair.
3. Sending user input to a registered application (Section 4). This is performed every time secure input to a registered application is required. We consider two kinds of registered applications, those which are BitE-aware, and those which are not.

3.2 Threat Model and Assumptions

Users use their computers to process sensitive information, for example, banking applications, corporate VPNs, and management of financial information. Attackers are interested in stealing such information, often with the intent to commit identity theft. One technique which attackers use is user-space malware, including Trojans and spyware such as keyloggers and screengrabbers.

BitE protects user input against user-space malware. We assume attackers are capable of passive monitoring and active injection attacks on the network link between the user's trusted mobile device and her host platform. We assume the user's mobile device is not compromised, although we discuss the possibility of using mutual attestation between the mobile device and host platform to detect a compromised mobile device in Section 5.3.

The mobile device must simultaneously connect to the user's input device (e.g., keyboard) and her host platform. We assume that a secure (direct physical connection, or authenticated and encrypted wireless connection) association between the user's input device and her mobile device can be established. However, there are environments where the user is wary of trusting an unknown keyboard for fear of hardware keyloggers. In such environments, the user can enter sensitive information directly into her mobile device, avoiding the use of the suspicious keyboard. Physical attacks such as "shoulder surfing" and keyboard emanation attacks [41] are beyond the scope of BitE. Thus, we do not discuss them further. For the remainder of the paper, we consider the user's input device (e.g., wireless keyboard) as an extension of her mobile device. Subsequent discussions will focus on interaction between the user, her mobile device, and her host platform.

We use attestation to verify the integrity of code loaded for execution on the user's host platform, so we need *not* assume that the host platform's software integrity is intact. However, as discussed in Section 2.3, the integrity measurement architecture used for attestation has TOCTOU limitations. Specifically, attestation allows us to detect modified program binaries before they are loaded. If a loaded application is compromised while running, however, IMA will not detect it.

We must trust the OS kernel on the host platform with which the user wishes to establish a trusted tunnel for input. One reason we must trust the OS kernel is because of its ability to arbitrarily read and modify the memory space of any process executing on the system—we cannot trust an application without also trusting the kernel on which it runs.

3.3 BitE Setup

We now describe application setup with BitE, which consists of two steps: (1) an association between the trusted mobile device and the host platform must be created, and (2) applications for which BitE will be used to secure their input must be registered with the BitE system.

3.3.1 Device Association

An association between the trusted mobile device and the host platform consists of two parts: (1) the BitE Mobile Client's ability to verify attestations from the host platform, and (2) keys for mutually authenticated and encrypted communication between the BitE Mobile Client and the BitE Kernel Module. The purpose of the attestation is to detect malware on the host platform; the purpose of the mutually authenticated and encrypted communication is to thwart active injection and passive monitoring attacks on the wireless connection between the BitE Mobile Client and the BitE Kernel Module.

To enable the BitE Mobile Client to verify attestations, it must be equipped with a set of expected measurements of acceptable software configurations for the host platform. However, the set of all possible software configurations that may be running on the host platform is unmanageable, as that set may include any piece of software that the user installs (willingly or unwillingly) on her host platform.

Our solution is to verify the measurements of those software components which are expected to change infrequently on a healthy system: the boot stack, the kernel, its modules (including the BitE Kernel Module), and well-ordered system services (those software components whose measurements are expected to occur in the same order between boot cycles). The BitE Mobile Client must be equipped with the expected values for the measurements of this infrequently-changing software so that it can verify attested measurements. In our current prototype, this is a manual process by which we add the expected filenames and measurements to a configuration file on the BitE Mobile Client. Another solution would be to add a configuration program which runs last during the boot process on the host platform, so that it can save a copy of all measurements of well-ordered services. Note that this still has the drawback that we are simply assuming the system to be secure when the configuration program runs; this is best done on a new system before it is connected to the Internet or other potential source of malware.

To authenticate the origin of an attestation, and to verify that the measurement list received matches the PCR quote from the host platform's TPM, the BitE Mobile Client must be equipped with the public *Attestation Identity Key* (AIK) from the host platform's TPM. The AIK is required by the BitE Mobile Client to verify the digital signature on attestations from the TPM in the host platform. Our current design only performs one-way attestation (host platform to mobile device); however, we discuss the possibility of mutual attestation in Section 5.3.

The BitE Mobile Client and the BitE Kernel Module must be able to establish mutually authenticated, encrypted communication to resist active injection and passive monitoring attacks on the wireless link between the user's mobile device and her host platform. Standard protocols exist for this purpose (e.g., SSL [12]) provided that authentic cryptographic keys are in place. We use the notation $\{K_{KM}, K_{KM}^{-1}\}$, $Cert_{KM}$ and $\{K_{MC}, K_{MC}^{-1}\}$, $Cert_{MC}$ for the asymmetric (e.g., RSA) keypairs and certificates (e.g., self-signed X509) for the BitE Kernel Module and the BitE Mobile Client, respectively.

We assume for simplicity that the attacker is not present during the exchange of AIK, $Cert_{KM}$, and $Cert_{MC}$. Thus, our current prototype exchanges these keys in the clear. We note that securing this key exchange is possible, though challenging. A potential solution is to use location-limited side channels to exchange pre-authentication data. Seeing-is-Believing and related techniques could be applied here [2, 16, 34].

3.3.2 Application Registration

Applications must be registered with the BitE Kernel Module and the BitE Mobile Client so that their integrity can be verified during subsequent attempts by the user to send input to them. The integrity measurement of each application serves as access control for application-specific cryptographic keys which are used to establish the trusted tunnel for user input.

To obtain the expected measurement value of an application, the user first indicates her desire to register a new application to the BitE Mobile Client. She then performs an initial execution of the application to be registered on her host platform. The IMA automatically measures this application and its library dependencies and stores them in the IMA measurement list (see [26] for details). We assume the system state can be trusted during application registration (i.e., there is no malicious code executing). Note that other dependencies may exist that we wish to measure. For example, configuration files can have a significant impact on application security. Automatic identification of configuration files associated with a particular application is complex, and beyond the scope of this paper. Alternatively, a trusted authority could provide the measurement values for a trustworthy version of the application.

The BitE Kernel Module generates a symmetric key K_{App_i} (for application i) to be used in subsequent connections for the derivation of encryption and MAC session keys for establishing the trusted tunnel. These per-application cryptographic keys are kept in TPM-protected sealed storage [37]. They are sealed under the PCR values which represent the boot process up through the loading of well-ordered system services (as described in Section 3.3.1). Handling which PCR registers receive which measurements is an issue which requires some care. We introduce the issue, and then present two possible solutions.

We dedicate a subset of the available PCRs for measurements of the well-ordered system services. However, once the appropriate measurements are in these PCRs, the TPM will allow the secrets in sealed storage to be

unsealed. A temping solution is to invalidate these PCRs after the BitE Kernel Module reads the sealed secrets by "measuring" some random data; however, this prevents the BitE Kernel Module from adding new secrets to sealed storage during the current boot cycle.

One option is to use dedicated PCR registers for the well-ordered system services, but not invalidate these registers after the BitE Kernel Module reads the sealed secrets. We must then trust the OS to restrict access to the TPM and hence the interface with sealed storage. This way, the BitE Kernel Module can add new secrets to sealed storage at any time. Our current prototype uses this option, which is more convenient, but less secure, than the next option.

An alternative is to allow application registration only immediately after a reboot. Thus, the application to be registered is the only software run during that boot cycle that is not part of the well-ordered system services, resulting in a more secure system state for registration, so that it is safer to leave the contents of the PCRs dedicated to well-ordered system services intact. Note that the reboot into "registration mode" may need to be a special, reduced-service reboot if there is non-determinism in the order in which some services are executed. For example, a host platform which starts a web server early in the boot process may execute a CGI script before the rest of the system has finished booting and invalidate some PCR values.

The IMA measurement for application i, the newly generated symmetric key K_{App_i}, and the user-friendly name of the registered application (e.g., *OpenOffice Calc*), are sent over a mutually authenticated, encrypted channel (established using $Cert_{KM}$ and $Cert_{MC}$ in, e.g., SSL with ephemeral Diffie-Hellman key agreement [12]) to the BitE Mobile Client, where they are stored for future use.

4 Operation

We now describe the actual process by which a user using BitE securely enters input into a registered application. At this point, the association between the trusted mobile device and the host platform is established, and the user's application(s) have been registered with the BitE system. The goal is to establish an end-to-end trusted tunnel for input between the user's trusted mobile device and an application running on her host platform. We define a trusted tunnel to be a mutually authenticated, encrypted network connection. We describe BitE using Linux and X11 terminology [39, 40]. However, our techniques can be applied to other operating environments, e.g., Microsoft Windows or Apple OS X, contingent on the existence of some means for attesting to the current software state.

We first consider applications designed with knowledge of BitE (Section 4.1), and later describe a wrapper which can be used with legacy applications (Section 4.2). We also consider conflict resolution which becomes necessary when multiple applications require a trusted tunnel concurrently (Section 4.3). Figure 2 shows a procedural overview of secure input using BitE.

4.1 BitE-Aware Applications

Establishment of a trusted tunnel is initiated by a BitE-aware application when it requires sensitive input (e.g., passwords or credit card numbers) from the user. The application sends a message to the BitE Kernel Module to register an input-event callback function implemented by the BitE-aware portion of the application. If the BitE Kernel Module has no other outstanding requests, it begins the process of establishing the trusted tunnel. We discuss extensions to allow the user to manually initiate a trusted tunnel in Section 6.2.

There are three steps in the establishment of a secure input session:

1. Verification by the BitE Mobile Client of an attestation produced by the host platform, including verification that the desired application was loaded (Section 4.1.1).
2. Interaction between the user and her trusted mobile device to confirm that it is indeed her desired application which is requesting secure input (Section 4.1.2).
3. Establishment of the session keys which will be used to encrypt and authenticate the actual keystrokes entered by the user (Section 4.1.3).

4.1.1 Attestation of Host Platform State

The BitE Mobile Client must verify an attestation of the software which has been loaded on the user's host platform. This serves two purposes: (1) to ensure the the the well-ordered system services on the host platform have not been modified since the mobile device / host platform association, and (2) to ensure that the trusted tunnel for input is established with exactly the same application that was initially registered. The BitE Mobile Client can verify the signature on the attestation with its authentic copy of the public AIK, and it can verify that the measurement list is consistent with the signed PCR quote. It can then compare the measurement values with those present during application registration (Section 3.3.2). If the values match, we consider the host platform to have successfully attested its software state to the BitE Mobile Client. The trusted mobile device now has assurance that the loaded versions of the well-ordered system services and the user's desired application match those recorded during registration. If this verification fails, then the process of establishing a trusted tunnel for input is aborted, and the user is notified via the display on her trusted mobile device.

Note that the software state of a host platform is comprised of *all* software loaded for execution; we discussed the difficulty of managing this state space in Section 2.3.

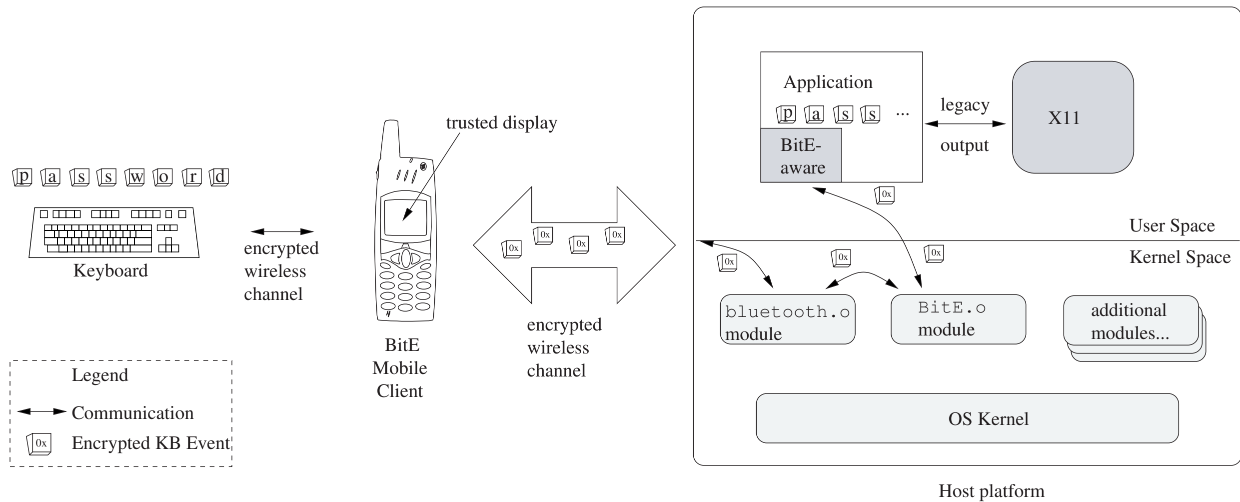

Figure 2: BitE system architecture. The user presses keys (e.g., types a password) on the keyboard. The keypress events are sent over an encrypted channel to the BitE Mobile Client. The BitE Mobile Client re-encrypts the keyboard events with a cryptographic key that is specific to some application. On the host platform, the encrypted keyboard events are passed to the BitE Kernel Module, and then to the application, where they are decrypted.

We verify measurements of the well-ordered system services and the user's desired application, but other software may be executing. It is such unknown user-level software against which the trusted tunnel offers protection.

4.1.2 User / Mobile Device Interaction

The trusted mobile device's display serves as a trusted output channel to the user. This enables us to minimize the amount of trust we place in the window manager on the host platform. Upon verifying an attestation from the host platform, the BitE Mobile Client has assurance that the correct application was loaded. Before session keys can be established to form the trusted tunnel, it is necessary to involve the user via her trusted mobile device to ensure that the application with which the user intends to interact and the application asking for her input are the same. This property can be challenging to achieve without annoying the user. A viable solution is one that is easy to use, but not so easy that the user "just hits OK" every time.

Our solution is to display a list of registered applications on the BitE Mobile Client. The user must scroll down (using the arrow keys on her keyboard, or navigational buttons on the trusted mobile device itself) and then select (e.g., press enter) the correct application. Note that since all input from the user's keyboard passes through the trusted mobile device the user does not actually need to press buttons on the mobile device. The mobile device will interpret the user input from her keyboard appropriately. We believe user confusion will be minimal, but a user-study is needed to validate this solution. Refer to Figure 3 for more information on the interaction between the user and her trusted mobile device.

We are concerned about users developing habits that might increase their susceptibility to spoofing attacks. Thus, we randomize the order of the list so that the user cannot develop a habit of pressing, e.g., "down-down-enter," when starting a particular application that requires a trusted tunnel. Instead, the user must actually read the list displayed on her mobile device and think about selecting the appropriate application. We believe selection from a randomized list achieves a good balance between security and usability, provided that the length of the list is constrained (for example, that it always fits on the mobile device's screen).

Once the list is displayed, the BitE Mobile Client signals the user, e.g., by beeping. This serves two purposes: (1) to let the user know that a secure input process is beginning; and (2) to let the user know that she must make a selection from choices on the mobile device's screen. Item (2) is necessary because a user may become confused if her application seems unresponsive when in reality the BitE Mobile Client on her mobile device is prompting her for a particular action.

Note that a look-alike (e.g., Trojan, spoofing attack) application will be unable to get the mobile device to display an appropriate name, because the look-alike application was never registered with the BitE system. Only

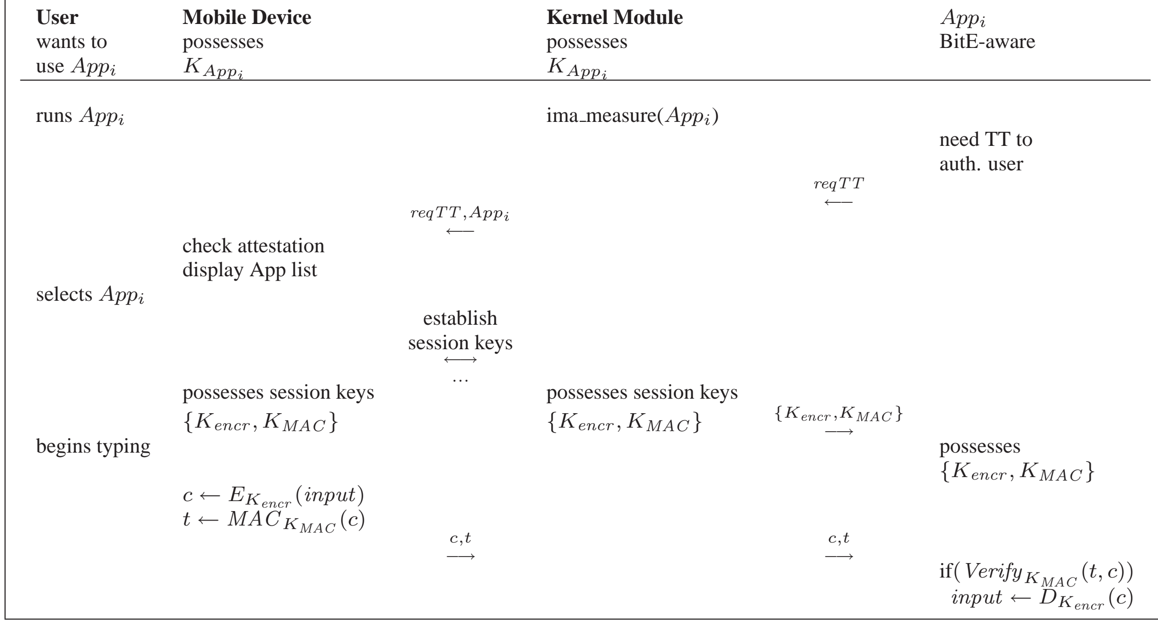

Figure 3: Simplified application execution and trusted tunnel establishment with registered BitE-aware application so that the user can enter sensitive information to that application. In this figure we assume the user's input device(s) is an extension of the trusted mobile device, so we do not show input device(s). We also assume host platform / BitE Mobile Client association (Section 3.3.1) and application registration (Section 3.3.2) have already been successfully completed. $input$ consists of padding, a sequence number, and the actual input event. TT = Trusted Tunnel.

applications that were initially registered are options for trusted tunnel endpoints.

If the user is satisfied, she selects the option given by her mobile device corresponding to the name of the application with which she wants to establish a trusted tunnel. If she suspects anything is wrong, she selects the *Abort* option on her mobile device. It is an error if the user selects any application other than the one which is currently requesting a trusted tunnel. That is, the BitE Mobile Client will report an error to the user (the application she selected from the list is not the same application that requested a trusted tunnel). It is a policy decision to decide how to handle this type of error. One approach is to fail secure, and prevent the user from entering sensitive input into her application until a successful retry.

Variations on this user interface that might also be effective in practice are discussed in Section 6.2.

4.1.3 Session Keys

At this point, the BitE Mobile Client has verified an attestation from the host platform, proving that the desired application and the correct well-ordered system services have been loaded. In addition, the user has selected the same application on her mobile device that requested secure input. To complete the encrypted, authenticated tunnel for input, session keys must be established.

Keys are established for encrypting and authenticating user input-related communication to and from the BitE Mobile Client. Even though user input is a one-way concept, bidirectional communication is necessary to properly support complex key sequences such as auto-repeat and Shift-, Control-, and Alt- combinations. The session keys are derived from the per-application keys established during application registration using standard protocols [17].

The BitE Mobile Client uses the session keys to encrypt and MAC the actual keyboard events such that they can be authenticated and decrypted by the BitE-aware application in an end-to-end fashion. Our current prototype does not consider keystroke timing attacks [33, 35]; incorporation of countermeasures for such attacks is the subject of future work.

Figure 3 presents step-by-step details on the process of input via the BitE trusted tunnel. For simplicity, the figure shows user input as a one-way data flow. K_{encr} and K_{MAC} are the encryption and MAC keys for input data flowing from the BitE Mobile Client to the application.

Once the trusted tunnel is established, the user can input her sensitive data. When the application is finished receiving sensitive input, it notifies the BitE Kernel Module that it is finished receiving input via the trusted tunnel. At this point, the BitE Kernel Module tears down the encrypted channel from the BitE Mobile Client to the application, and reverts to listening for requests for trusted tunnels from other registered applications. The BitE Mobile Client notifies the user that the secure input session has finished.

4.2 BitE-Unaware (Legacy) Applications

We now describe BitE operation with an application that is *unaware* of the BitE system. That is, this section describes how BitE is backwards-compatible with existing applications. Legacy applications were written without knowledge of BitE, so there is no way for a legacy application to request a trusted tunnel. Hence, all input to a legacy application must go through a trusted tunnel. The basic idea is that we run legacy applications inside a *wrapper* application (the BitE-wrapper) that provides input events to that application (e.g., `stdin` or X keyboard events).

The legacy application is automatically measured by the IMA, and it is registered with BitE in the same way BitE-aware applications are registered. If the application changes after its initial registration, the BitE Kernel Module will not release the application-specific keys necessary to establish session keys for encrypting and authenticating keyboard events. Note that the application-specific keys can be unsealed from sealed storage as long as the measurements of the well-ordered system services are as expected. Once the BitE Kernel Module has access to the application's keys, it will only release them to the application if that application's measurement matches the expected value.

The most challenging part of interacting with a legacy application is that it contains no BitE-aware component that can handle the decryption and authentication of keyboard events. Instead, the BitE-wrapper does the decryption and authentication of keyboard events. It is necessary to prevent the legacy application from receiving keyboard events from the window manager (or other user-level processes), while allowing it to receive input from the wrapper application. This is easy to achieve for console applications (e.g., just redirect `stdin`); however, it is challenging for graphical applications. We now consider the necessary BitE-wrapper functionality for X11 applications.

X11 applications (*clients* in the context of X) register to receive certain types of event notifications from the X server. Common event types include keyboard press and release events. Applications register to receive these events using the `XSelectInput` function. The BitE-wrapper application can intercept this call for dynamically linked applications using the `LD_PRELOAD` environment variable. With `LD_PRELOAD` defined to a custom BitE shared library, the run-time linker will call the BitE `XSelectInput` instead of the X11 `XSelectInput`. Thus, the BitE Kernel Module is hooked into the application's input event loop. The BitE Kernel Module can generate its own input events to send to the application simply by calling the callback function that the application registered when it called `XSelectInput`.

4.3 Handling Concurrent Trusted Tunnels to Prevent User Confusion

While there are no technical difficulties involved in maintaining multiple active trusted tunnel connections from the BitE Mobile Client to applications, there are user-interface issues. We know of no way to disambiguate to the user which application is receiving input without requiring user diligence. For example, a naive solution is to display the name of the application for which user input is currently being tunneled on the mobile device's screen. This requires the user to look at the screen of her mobile device and ensure that the name matches that of the application with which she is currently interacting.

To prevent user confusion, we force the user to interact with one application at a time in a trusted way. If we allow users to rapidly switch applications (as today's window managers do), then the binding of user intent with user action is dramatically weakened. The rapid context switching makes it easy for the user to become confused and enter sensitive input into the wrong application. An adversary may be able to exploit this weakness. Some users may find this policy annoying; we discuss an alternative policy in Section 6.2.1.

We consider two example applications which we assume to be BitE-aware and that require a trusted tunnel for user authentication:

1. Banking software which requires the user to authenticate with an account number and a password.
2. A virtual private network (VPN) client which requires the user to authenticate with a username and a password.

Suppose the user needs to interact with both applications at the same time, for example, to compare payroll information from her company with entries in her personal bank account. In today's systems, there is nothing to cause the user to serialize her authentication to these applications. She may start the banking software, then start the VPN client, then authenticate to the banking software, then authenticate to the VPN client. In the BitE system, assuming the banking software and VPN client are BitE-aware, the BitE Kernel Module considers this behavior to be a concurrent request by two applications for establishment of a trusted tunnel.

It is a policy decision how to handle concurrent trusted tunnel requests. One option is to default-deny both applications, and alert the user to the contention. She can then retry with one of the two applications, and use it first. This forces the user to establish a trusted tunnel to the first application and fully input her sensitive data to that application. Once her data is input, the first application will relinquish the trusted tunnel, and it will be torn down by BitE. The user can then begin the process of entering her sensitive data to the second application, which will entail the establishment of another trusted tunnel. These one-at-a-time semantics may induce some additional la-

tency for the user before she can begin using her applications, but we consider this to be an acceptable tradeoff in light of the gains in security.

5 Security Analysis

In this section we analyze the security of BitE. During the design of BitE, we tried to make it difficult for the user to make self-destructive mistakes. For example, the BitE Mobile Client will not allow the user's keystrokes to reach the BitE Kernel Module if verification of an attestation fails. The user must respond to messages displayed on her mobile device before she can proceed. Security mechanisms on the critical input path cannot go unnoticed by the user. These mechanisms must provide value commensurate with the difficulty of using them.

We provide some examples of attacks that BitE is able to protect against. We then consider the failure modes of BitE when the assumptions upon which it is constructed do not hold.

5.1 Defensed Attacks

We consider multiple scenarios where the use of BitE protects the user.

Capturing Keystrokes with X Giampaolo shows how easy it is for an attacker to use a malicious application to capture the keystrokes the user intends to go to the active (and assumed benign) application. If the user is using BitE to enter sensitive data, however, this attack does not work (see Figure 1). The user's keystrokes are encrypted and authenticated with session keys (as discussed in Section 4) which are unavailable to the malicious application. Hence, the encrypted keystrokes reach the user's desired application unobserved.

Software Keyloggers Software keyloggers are a significant threat. Sumitomo Bank in London was the victim of a sophisticated fraud scam involving software keyloggers [27]. With BitE, the user's keystrokes travel inside encrypted, authenticated tunnels. Even if an adversary can capture the ciphertext, he will be unable to extract the keystrokes.

Bluetooth Eavesdropping BitE is most convenient for the user when wireless communication mechanisms can be used between the mobile device and the host. As long as the initial exchange of public keys between the BitE Mobile Client and the BitE Kernel Module proceeds securely, all communication between them can be encrypted and authenticated using standard protocols. If keystroke timing attack countermeasures are incorporated [33, 35], timing side-channels can also be eliminated. Since all communication is strongly authenticated, an adversary will not be able to masquerade as a valid BitE Mobile Client or BitE Kernel Module.

Modification of Registered Applications An attacker may be able to modify (e.g., by exploiting a buffer overflow vulnerability in a different application) the binary of a registered application. Such an attack may modify the application's executable such that it may log user input to a file, or send it to a malicious party on the Internet. With BitE, an IMA measurement of the executable was recorded during initial application registration (recall Section 3.3.2). The modified application binary will be detected during trusted tunnel setup when the BitE Mobile Client tries to verify the attestation from the host platform. The BitE Mobile Client will alert the user that the application has been modified.

Kernel Modification A measurement of the kernel binary is part of the integrity measurement which is verified when a trusted tunnel is established. Modification of the kernel image on disk will be detected after the next reboot. As a disk-only modification of the kernel image will not affect the running system until a reboot, the attack is detected by the BitE Mobile Client before it can affect the operation of BitE.

5.2 Failure Modes

We now describe what happens if the assumptions upon which the security of BitE is based turn out to be invalid. Specifically, we discuss the extent to which the failure of our assumptions permit the attacker to perform one or more of the following:

- To observe keystrokes in an ongoing session.
- To observe keystrokes in current and future sessions.
- To register malicious applications.

Compromise of Active Application If the attacker is able to compromise an application while the user has a trusted tunnel established, he may be able to observe the user's keystrokes. This break is limited to the compromised application, however, as the attacker has no way to access keys established between the BitE Kernel Module and other registered applications. This break is feasible because the adversary is exploiting a time-of-check, time-of-use (TOCTOU) limitation of the integrity measurement architecture (e.g., a buffer overflow attack). Incorporation of mechanisms for run-time attestation (e.g., Pioneer [28]) can help defend against this attack.

Compromise of Active Kernel on Host Platform If the operating system kernel on the host platform is compromised without rewriting a measured binary (e.g., exploiting TOCTOU limitations with a buffer overflow attack), the attacker may be in a position to capture sensitive user input despite the BitE system. This gives the attacker access to K_{KM}^{-1} and to the unique application keys K_{App_i}. The attacker can also capture keystrokes from ongoing sessions by reading the session keys out of the memory space of the BitE Kernel Module or the applications.

Compromised Mobile Device Since the mobile device is used as a central point of trust in our system, its compromise will allow an attacker to access all keyboard events. The attacker will have possession of K_{MC}^{-1} so he may be able to masquerade as a trusted BitE Mobile

Client using an arbitrary device, such as one with a very powerful radio transmitter. Further, the attacker will capture all registered applications' unique keys, K_{App_i} for application i, and user-friendly name. This will enable the attacker to establish trusted paths with registered applications, and it will allow the attacker to register new applications.

Hardware Keyloggers Malicious parties may use hardware as well as software techniques to record users' keystrokes. BitE is designed to protect against software-based attacks. To protect against hardware-based attacks, the user would need to carry their own keyboard as well as their mobile device, which we consider to be an excessive burden. However, if the amount of sensitive data that the user must enter is small, she can use BitE without an external keyboard and enter her sensitive data directly into her mobile device.

5.3 Mutual Attestation

Currently, there is no attestation technology available for mobile devices. However, the TCG is currently working on trusted platform standards for mobile devices [36], which may be able to minimize the severity of a mobile device compromise. With such technology, it becomes possible to implement mutual attestation in BitE, where the host platform verifies an attestation from the mobile device in addition to the mobile device verifying an attestation from the host platform.

Mutual attestation can increase an attacker's workload, since compromising only the mobile device is no longer a sufficient attack. To fully circumvent BitE, the attacker would have to compromise both the BitE Mobile Client and the user's host platform. For example, the BitE Mobile Client could store its secrets in sealed storage, rendering them inaccessible to malicious software installed on the mobile device by an adversary.

6 Discussion

In this section we discuss additional issues that arise while using the BitE system. These issues include alternative system architectures, e.g., elimination of the trusted mobile device or TPM, and alternative user interface designs for the BitE system.

6.1 Alternative System Architectures

The BitE system as presented in this paper is designed around a TPM-equipped host platform and a trusted mobile device. We briefly consider alternative design approaches, namely, designs that eliminate the mobile device or the TPM. It is particularly tempting to think that one or both of these requirements may be unnecessary since we include the OS kernel in the TCB for BitE. In addition to its role as resource manager, we must trust the OS kernel because of its ability to arbitrarily read the memory space of any process executing on the system. Other researchers have tried to quantify the extent to which data is exposed by measuring the *lifetime* of data in a system [7, 13]. In BitE, we minimize the data lifetime of user input by minimizing the quantity of code through which cleartext input passes.

6.1.1 Eliminating the Mobile Device

A significant challenge addressed by BitE is for the user to obtain a user-verifiable property of the integrity of the OS and application. It is important to recall two of the roles the mobile device fulfills:

- Verification of integrity measurements from the host platform.
- Trusted visual output to the user.

Integrity measurement on the host platform puts in place the facility for verification by an external entity, but the host platform cannot "self-verify." In BitE, the mobile device fulfills the role of the verifying entity. The mobile device must have trusted visual output to the user so it can appropriately notify the user of the success or failure of this verification.

6.1.2 Input Proxying by the Mobile Device

The OS kernel on the user's host platform is part of the TCB when using BitE. With a trusted OS kernel, there is no technical reason why user input must travel through the mobile device. The BitE Kernel Module already possesses copies of the cryptographic keys shared with the application, and it could encrypt user input from the traditional keyboard driver before it passes through the window manager. However, this design raises an important usability issue. The mobile device must be on the critical input path so that it can ensure that the user cannot proceed unaware of a failed integrity verification. We are concerned that non-expert users will proceed to interact with their application even if a message appears on the mobile device stating that the system is compromised. With the mobile device on the critical input path, it can stop user input from reaching the application while also providing secure feedback to the user.

6.1.3 TPM Alternatives

Finally, we discuss the extent to which the TPM is an essential requirement for BitE. We could leverage software-based attestation mechanisms to verify the authenticity of the OS [28]. However, in open computing environments, it may be challenging to satisfy the assumptions underlying these techniques, e.g., that the verified device cannot communicate with a more powerful computer during the attestation process. Still, software- and hardware-based attestation mechanisms complement each other. Hardware-based attestation mechanisms can provide load-time guarantees, with software-based mechanisms helping to ensure tamper-evident execution. This reduces the threat posed by the TOCTOU vulnerability of the IMA.

6.2 Alternative User Interfaces

An important property achieved by the BitE system is that the user selects the registered application with which she would like to establish a trusted tunnel from a list on her trusted mobile device's screen. We present two alternative operating models that may be more convenient for the user, though they tradeoff the strength of the resulting trusted tunnel.

6.2.1 Active Selection

As described in Section 4, BitE requires the user to select an application from a list presented by the BitE Mobile Client when a registered application requests a trusted tunnel for input. An alternative structure, and one that may be preferable for legacy applications, is one where the user directs the BitE system to establish a trusted tunnel. One possibility for giving the user this ability is to maintain a list of all registered applications on the BitE Mobile Client. When the user wants to send secure input to, e.g., *OpenOffice Calc*, she selects "OpenOffice Calc" from the list on her mobile device.

This system has advantages for legacy applications since they do not actively request a trusted tunnel for input. BitE as described in Section 4.2 requires a legacy application to receive *all* input through the trusted tunnel, which may be inconvenient for the user if she wishes to interact with a second application with a trusted tunnel while her legacy application is still running (and using the trusted tunnel).

The drawback of this *Active Selection* scheme is that it requires user diligence. We are concerned about users forgetting to manually enable the trusted tunnel when they input sensitive data.

6.2.2 Always-On-Top

Another possible configuration for a trusted tunnel system is one where the window manager is involved. In this scenario, the BitE Mobile Client and the BitE Kernel Module maintain multiple active sessions. The user's typing goes to whatever application the window manager considers to be "on top." This configuration for BitE is problematic since the window manager becomes a part of the TCB. Much of the motivation for BitE is that it is able to remove the window manager from the TCB for trusted tunnel input.

7 Prototype and Evaluation

In this section, we describe our proof-of-concept prototype and evaluate some practical considerations necessary for actually building BitE. We have developed a J2ME MIDP 2.0 [19] BitE Mobile Client. It receives keystrokes via an infrared keyboard and sends them via Bluetooth to a prototype BitE Kernel Module loaded on the host platform. We use the BouncyCastle Lightweight Cryptography API for all cryptographic operations. Our prototype is shown in operation in Figure 4.

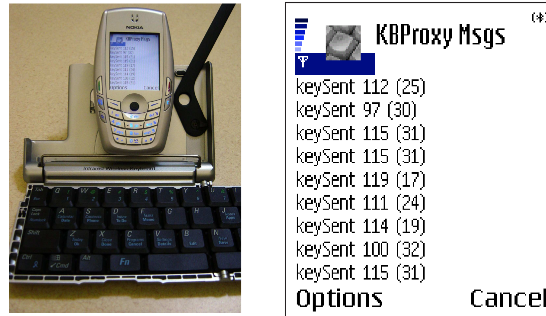

Figure 4: Our prototype BitE system and a debugging screenshot showing the prototype in operation.

The BitE Mobile Client consists of less than 1000 lines of Java code, not including source code from libraries. The BitE Kernel Module consists of approximately 500 lines of C and Java code which interacts with the BitE Mobile Client via Bluetooth, the legacy input system via the `uinput` kernel module, and the integrity measurement architecture of Sailer et al. [26] via the `/proc` filesystem.

7.1 Options for a Trusted Mobile Device

We decided to develop the BitE Mobile Client as a J2ME application because of the wide variety of mobile devices which support J2ME. In particular, millions of smartphones have already been sold which are capable of running the BitE Mobile Client. However, mobile phones are continuously increasing in complexity and thus feature software vulnerabilities of their own. A recent study places the number of existing worms and viruses for mobile phones at approximately 90 [15]. While 90 is a miniscule number when compared to the amount of malcode in circulation which affects desktop computing platforms, it is still a significant figure when considering the amount of trust BitE puts in the mobile device. Efforts are ongoing to improve the security of mobile devices, augmented by the experience gained working to secure more traditional platforms [36]. Still, with today's technology, BitE is best run on a dedicated device whose software can only be upgraded under carefully controlled conditions.

7.2 Encrypted Channel Setup Latency Between Mobile Device and Host

We have performed experiments to determine the overhead associated with asymmetric cryptographic operations necessary to establish encrypted, authenticated communication between a mobile phone and host. We ran our J2ME MIDP 2.0 application on both a Nokia 6620 and a Nokia N70.

Establishing mutually authenticated communication involves performing asymmetric signature and verification operations at both communication endpoints. In our experiments with 1024-bit RSA keys (see Table 1), signing operations take 1757 ms and 1332 ms on average

Action	Nokia N70		Nokia 6620	
	Time (ms)	Variance	Time (ms)	Variance
RSA PSS (sign)	1332	171	1757	297
RSA verify	40	63	54	31
SHA-1 aggregate	91	125	171	110
Data manipulation	906	1000	2087	703

Table 1: Average time (in milliseconds) to perform an RSA signature and verification with a 1024-bit key and a public exponent of 65537; compute an aggregate hash of 401 (for the Nokia 6620) and 325 (for the Nokia N70) SHA-1 measurements from a Debian workstation running Linux kernel 2.6.12.5; and manipulate the IMA measurement list data.

for a Nokia 6620 and Nokia N70, respectively. Signature verification averages 54 ms and 40 ms, respectively. Thus, mutual authentication using either of these phones will take on the order of 3 to 4 seconds, which is a noticeable but tolerable delay. This is because these asymmetric operations are only required for communication setup. Once session keys are established, efficient symmetric primitives can be used for communication. The Nokia N70 consistently outperforms the Nokia 6620, but the margin is rather narrow.

7.3 Keyboard / Mobile Device Communication

We experimented with an infrared keyboard to provide user input to the BitE Mobile Client, and a Bluetooth connection from the BitE Mobile Client to the host platform. This is because our development phones can support only one Bluetooth connection at a time. The use of an infrared keyboard is undesirable because the keystrokes are transmitted in the clear. A better solution is to use a keyboard that physically attaches to the mobile phone. Alternatively, a Bluetooth keyboard capable of authenticated, encrypted communication can be used. We are unaware of any mobile phones available today that support more than one active Bluetooth connection simultaneously, but such devices may become available in the near future.

We performed simple usability experiments to analyze keystroke latencies, to ensure that BitE is not rendered useless by excessive latency while typing. We observe no noticeable latency with debug logging disabled. Use of symmetric cryptographic primitives introduces minimal overhead per keystroke. For example, the use of counter-mode encryption could enable the BitE Mobile Client to precompute enough of the key stream so that the only encrypt / decrypt operation on the critical path for a keystroke is an exclusive-or [17]. This reduces the most significant cryptographic per-keystroke operation to the verification of a MAC (e.g., HMAC-SHA-1, [3, 4, 14]). Our experiments below on verifying integrity measurements indicate that SHA-1 operations can be performed efficiently on the class of mobile phones we consider.

7.4 Verifying Attestations on the Mobile Device

We have the open-source integrity measurement architecture (IMA) of Sailer et al. [26] running on our development system. We have implemented the operations necessary for the BitE Kernel Module to send the IMA measurement list to the BitE Mobile Client, and for the BitE Mobile Client to compute the PCR aggregate for comparison with a signed PCR quote from the TPM. Table 1 shows some performance results for our prototype on a Nokia 6620 and N70. To validate an attested set of measurements from the host platform, the measurement list must be hashed for comparison with the signed PCR aggregate. For a typical desktop system, this involves hundreds of hash operations. Our experiments show that the average time necessary to compute a SHA-1 hash of 325 measurements (the number of measurements our development system had performed at the time of the experiment), 91 ms on the N70, is far less than the time necessary to manipulate the data from the measurement list—906 ms. In this experiment, the Nokia N70 is bound by memory access and not CPU operations. Our results show that the expected time for a Nokia N70 to verify an attestation (check the signature on the aggregate values from the PCRs, hash the measurement list, and compare the aggregate hash from the measurement list to the appropriate PCR value) is approximately 2 seconds, which will decrease with future devices. See Table 1 for the results from a corresponding experiment run on a Nokia 6620.

8 Conclusions

Bump in the Ether is a system that uses a trusted mobile device as a proxy between a keyboard and a TPM-equipped host platform to establish a trusted tunnel for user input to applications. The resulting tunnel is an end-to-end encrypted, authenticated tunnel all the way from a user's mobile device to an application running on the host platform. BitE places specific emphasis on these design issues: (1) malware such as software keyloggers, spyware, and Trojans running at user level will be unable to capture the user's input; (2) operation of BitE is convenient and intuitive for users; (3) BitE is feasible today on commodity hardware; and (4) BitE offers some protection for legacy applications.

9 Acknowledgements

We are grateful to Reiner Sailer and Leendert van Doorn for their assistance with the Integrity Measurement Architecture. We would also like to thank the anonymous reviewers for their valuable feedback.

This work was supported in part by National Science Foundation award number CCF-0424422.

References

[1] D. Balfanz and E. W. Felten. Hand-held computers can be better smart cards. In *Proccedings of the USENIX Security Symposium*, August 1999.

[2] D. Balfanz, D.K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the Symposium on Network and Distributed Systems Security*, February 2002.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keyed hash functions and message authentication. In *Proceedings of Crypto*, pages 1–15, 1996.

[4] M. Bellare, T. Kohno, and C. Namprempre. SSH transport layer encryption modes. Internet draft, August 2005.

[5] J. L. Berger, J. Picciotto, J. P. L. Woodward, and P. T. Cummings. Compartmented mode workstation: Prototype highlights. *Software Engineering*, 16(6):608–618, June 1990.

[6] M. Carson and J. Cugini. An X11-based Multilevel Window System architecture. In *Proceedings of the EUUG Technical Conference*, 1990.

[7] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum. Understanding data lifetime via whole system simulation. In *Proceedings of the USENIX Security Symposium*, August 2004.

[8] J. Epstein. A prototype for Trusted X labeling policies. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, December 1990.

[9] J. Epstein and J. Picciotto. Issues in building Trusted X Window Systems. *The X Resource*, 1(1), Fall 1991.

[10] J. Epstein and J. Picciotto. Trusting X: Issues in building Trusted X window systems -or- what's not trusted about X? In *Proceedings of the 14th Annual National Computer Security Conference*, October 1991.

[11] G. Faden. Reconciling CMW requirements with those of X11 applications. In *Proceedings of the 14th Annual National Computer Security Conference*, October 1991.

[12] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol: Version 3.0. Internet draft, Netscape Communications, November 1996.

[13] T. Garfinkel, B. Pfaff, J. Chow, and M. Rosenblum. Data lifetime is a systems problem. In *Proceedings of the ACM SIGOPS European Workshop*, September 2004.

[14] P. Jones. RFC 3174: US secure hash algorithm 1 (SHA1), September 2001.

[15] N. Leavitt. Will proposed standard make mobile phones more secure? *IEEE Computer*, 38(12):20–22, 2005.

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

[17] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[18] Microsoft Corporation. Next generation secure computing base. `http://www.microsoft.com/resources/ngscb/`, April 2006.

[19] Sun Microsystems. Mobile information device profile (MIDP) version 2.0, April 2006.

[20] B. A. Myers. Using handhelds and PCs together. *Communications of the ACM*, 44(11), November 2001.

[21] J. Picciotto. Towards trusted cut and paste in the X Window System. In *Proceedings of the Seventh Annual Computer Security Applications Conference*, December 1991.

[22] J. Picciotto and J. Epstein. A comparison of Trusted X security policies, architectures, and interoperability. In *Proceedings of the Eighth Annual Computer Security Applications Conference*, December 1992.

[23] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[24] D. S. H. Rosenthal. LInX—a Less INsecure X server (Sun Microsystems unpublished draft).

[25] S. J. Ross, J. L. Hill, M. Y. Chen, A. D. Joseph, D. E. Culler, and E. A. Brewer. A composable framework for secure multi-modal access to Internet services from post-PC devices. *Mobile Network Applications*, 7(5):389–406, 2002.

[26] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the USENIX Security Symposium*, 2004.

[27] J. Scott-Joynt. The enemy within. BBC News, Available at: `http://news.bbc.co.uk/2/hi/business/4510561.stm`, May 2005.

[28] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles*, October 2005.

[29] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia. Design of the EROS trusted window system. In *Proceedings of the USENIX Security Symposium*, 2004.

[30] R. Sharp, A. Madhavapeddy, R. Want, T. Pering, and J. Light. Fighting crimeware: An architecture for split-trust web applications. Technical Report to appear, Intel Research Center, 2006.

[31] R. Sharp, J. Scott, and A. Beresford. Secure mobile computing via public terminals. In *Proceedings of the International Conference on Pervasive Computing*, May 2006.

[32] E. Shi, A. Perrig, and L. van Doorn. BIND: A time-of-use attestation service for secure distributed systems. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.

[33] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the USENIX Security Symposium*, 2001.

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

[35] J. T. Trostle. Timing attacks against trusted path. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1998.

[36] Trusted Computing Group. Security in mobile phones whitepaper, October 2003.

[37] Trusted Computing Group. Trusted platform module main specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands, October 2003. Version 1.2, Revision 62.

[38] J. P. L. Woodward. Security requirements for system high and compartmented mode workstations. Technical Report MTR 9992, Rev. 1, The MITRE Corporation, November 1987.

[39] The XFree86 project, Inc. `http://www.xfree86.org/`, April 2006.

[40] The X.Org foundation. `http://www.x.org/`, April 2006.

[41] L. Zhuang, F. Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the ACM Conference on Computer and Communications Security*, October 2005.