Device-Based Isolation for Securing Cryptographic Keys

Karim O. Elish, Yipan Deng, Danfeng (Daphne) Yao, Dennis Kafura
Department of Computer Science
Virginia Tech
Blacksburg 24060, USA
{kelish, yipanvt, danfeng, kafura}@cs.vt.edu

ABSTRACT

In this work, we describe an effective device-based isolation approach for achieving data security. Device-based isolation leverages the proliferation of personal computing devices to provide strong run-time guarantees for the confidentiality of secrets. To demonstrate our isolation approach, we show its use in protecting the secrecy of highly sensitive data that is crucial to security operations, such as cryptographic keys used for decrypting ciphertext or signing digital signatures. Private key is usually encrypted when not used, however, when being used, the plaintext key is loaded into the memory of the host for access. In our threat model, the host may be compromised by attackers, and thus the confidentiality of the host memory cannot be preserved. We present a novel and practical solution and its prototype called *DataGuard* to protect the secrecy of the highly sensitive data through the storage isolation and secure tunneling enabled by a mobile handheld device. DataGuard can be deployed for the key protection of individuals or organizations.

Keywords

data security, confidentiality, cryptography, device-based isolation, mobile device

1. INTRODUCTION

In the seminal work on computer virus [11], Cohen described isolationism as a potential prevention against the propagation of viruses (on multi-user computers), which refers to no dissemination and no sharing of information across information boundaries. However, absolute isolation clearly would significantly hinder the usefulness of the computing environment. Yet, partial isolation can be strategically implemented to realize specific security goals. For example, Borders *et al.* proposed a solution for achieving data confidentiality that requires disconnection from the network when the data is being accessed [7].

In this work, we describe a practical and powerful device-based isolation approach for information security and demonstrate its application in preserving the confidentiality of cryptographic keys. The confidentiality of cryptographic keys is the security foundation of virtually all cryptographic/security protocols, and is essential for individuals and organizations including certificate authorities (e.g., Symantec). Device-based isolation is defined by us as isolating the storage and operations related to data with different security requirements (e.g., confidentiality requirement) through multiple computing devices. In addition, the isolation should not hinder the use and access of the data for practical applications.

In a strong adversary model, the host and its kernel may be compromised by attackers or malicious software, and thus the confidentiality of the host memory - including cryptographic keys stored there - cannot be guaranteed. For example, simply storing the plaintext private key on a removable medium (eg., a regular USB drive) is not a complete solution because the key still needs to be loaded to the host for use. For this reason, the cryptographic (private) key is usually encrypted on external storage when it is not being used. When needed, the encrypted key is decrypted (which typically requires the user to enter a passphrase). The plaintext key is then loaded into the memory of the host for access.

A number of techniques have been developed to preserve the confidentiality of cryptographic keys. Using a tamper-resistant smart card for the key storage requires a special reader, which may not be available. In a more complex solution, the host may be equipped with an on-chip trusted platform module (TPM) that allows the host to attest its system integrity to a trusted server when the plaintext key is loaded; the encrypted private key can only be decrypted when the host's system integrity is verified. However, one major limitation of this approach is that TPM cannot prevent any run-time compromise taking place after the attestation [15]. Thus, the keys may be exposed. In general, the fundamental and practical security problem of protecting the secrecy of the private key is not adequately addressed in the security literature.

In this paper, we present a novel and practical technique to enhance the confidentiality of private keys. Our design separates private keys from an untrusted host to a special-purpose, trusted handheld device, while still enabling the host to use the key (e.g., for decryption and signing). Our approach recognizes that ensuring system integrity - the assurance that applications and the host's operating system including the kernel have not been compromised by attackers or malicious software - of a general-purpose computer with Internet connectivity is difficult in general. However, a

dedicated special-purpose computing device is relatively easier to secure and less susceptible to attacks. Such a device may be an inexpensive smartphone or tablet computer. We use a trusted device for dedicated key storage and cryptographic operations, which allows the key to be used without being exposed to the untrusted host. Accordingly, our solution can be easily deployed by any entities (individuals or organizations) using the public key cryptosystems to enhance their protection of private keys.

Our device-based isolation framework - named DataGuard - provides two security functions: *i)* it protects highly sensitive cryptographic keys by isolating them from the untrusted general computing environment, and *ii)* it supports the secure use of the key through a protocol by which the host may request a message to be signed with the protected private key. Without our device-based isolation of secret keys, the cleartext key value would exist in the memory of the host when it is being used, and thus a compromised host may lead to key compromise. In our prototypes, we show several independent instantiations of DataGuard with various types of machines, including a smartphone, a tablet computer, and a Linux box for embedded system. We also implemented two types of communication methods (wired and wireless) between the host and the device - Bluetooth and universal serial bus.

Our contributions are summarized as follows.

1. We revisit the feasibility of isolationism for security and propose device-based isolation as a general approach for securing data confidentiality, which leverages the proliferation of personal handheld devices. We describe the general requirements associated with realizing a device-based isolation solution.

2. We present the design, implementation, and experimental evaluation of *DataGuard* for securing cryptographic private keys as a concrete embodiment of the device-based isolation methodology. DataGuard allows one to isolate the storage and operation associated with keys on a trusted device, yet enables the use of the keys for decryption and signing by the host as usual.

   We implement a prototype of DataGuard that supports two types of communication methods and runs on three different device platforms. We further demonstrate how DataGuard can be integrated with Google Chrome browser for real-time decrypting or signing without exposing the private key to the host. We have conducted series of experimental evaluation to test the performance of DataGuard in terms of performing cryptographic operations related to decryption and signing on the external trusted device.

The rest of the paper is organized as follows. Section 2 presents the design of DataGuard framework. Section 3 provides two security protocols to illustrate the applications of DataGuard. Sections 4 and 5 discuss our implementation and experimental evaluation results respectively. Section 6 briefly reviews related work. Finally, Section 7 concludes the paper and outlines directions for future work.

2. DESIGN OVERVIEW

In this section, we present the architecture and the main components of the DataGuard framework. Then, we detail our security goals, threat model, and assumptions. Finally, we describe the applications of DataGuard framework.

2.1 DataGuard Architecture

Our device-based isolation design consists of two main components: *i)* a host running a DataGuard daemon, and *ii)* an external trusted device running a DataGuard application, which are described as follows.

1. **Host** is a personal computer which is vulnerable to malware attacks. However, the host's owner needs to utilize the private cryptographic keys to decrypt ciphertext or sign messages. Hence, we need to provide the security enhancement to this host by protecting the secret cryptographic keys from being compromised.

   DataGuard daemon is a software application used to connect the host with the external trusted device. It initiates a communication channel with the trusted device and uses this channel to execute the commands requested from the DataGuard application running on the trusted device.

2. **External trusted device** is an external device used to enhance the data security of the host by storing the cryptographic secret keys of the host and performing the cryptographic operations. DataGuard is deployed on this device. The selection of the trusted device should support the following criteria in order to be used in our framework:

   - It should have the capability to perform computation.
   - It should have storage space.
   - It should have the ability to communicate with other devices.
   - It should have I/O capability with a display screen.

   Any device that possesses these characteristics can be used as a trusted device in our framework.

   DataGuard application is a software application running on the external trusted device. It is used to communicate with the host and performs the security-related cryptographic operations (namely decryption and signing) which are described in details in Section 4.

2.2 Security Goals, Threat Model, and Assumptions

- **Security goals:** The primary goal of our framework is to enhance the data confidentiality of a host by protecting its sensitive data from being compromised by malicious attacks. In particular, our security goals are three-fold:

  1. To ensure the confidentiality of cryptographic private keys.
  2. To ensure the confidentiality of ciphertext being decrypted.
  3. To ensure the integrity of digital signature produced.

- **Threat model and security assumptions:** In our threat model, we assume that the host including the user-space and kernel-space data and code are not secure and vulnerable to malicious attacks. Specifically, the attacker may attempt to: *i)* gain knowledge of a private key, *ii)* gain knowledge of the message encrypted with the corresponding public key, and *iii)* forge digital signatures on behalf of the legitimate key owner.

  We assume that the external device's data and code are trusted and not compromised. This assumption is reasonable, as the device can be a special-purpose computing device with no or limited network connectivity and limited permitted operations.

2.3 Why the External Device is Trusted?

A general purpose computer may be vulnerable to malicious attacks as it is typically connected to the Internet. DataGuard provides a secure communication channel between the external device and the host as shown in Figure 1 (b). In this case, the external device is more isolated and easier to secure compared to the conventional approach illustrated in Figure 1 (a). The external device is not exposed directly to the Internet. We refer to the external device as a *trusted device*.

The external device is easier to secure compared to a general purpose computer because of two reasons as follows.

- *Singular communication*: it only securely communicates to the host.
- *Singular service*: it only provides one specific key usage service. Its data and code are small as a result of limited operations.

Therefore, our assumption on the integrity of the trusted device is practical.

The storage and usage environments of the cryptographic keys that we aim to protect are isolated from the general computing environment. This property is achieved because the keys are stored and operated on the trusted device, and never appear on general purpose computers (i.e., the host).

2.4 Use Cases of DataGuard

The DataGuard framework can be used by *i)* individuals and *ii)* organizations to protect the cryptographic secret keys used in many security related operations.

- **Personal use:** Individual users who perform cryptographic operations, such as decryption and digital signing, can use our device-based isolation method easily to protect their personal private keys used in these operations. DataGuard provides no extra cost to the users as they can use their handheld devices such as smartphones to isolate the storage and operations associated with their private keys.
- **Organizational use:** DataGuard framework can be also used by the enterprise, government, or military organizations to protect their master private keys. For example, certificate authorities such as Symantec need to carefully protect the confidentiality of the master private key that is used to generate digital certificates.

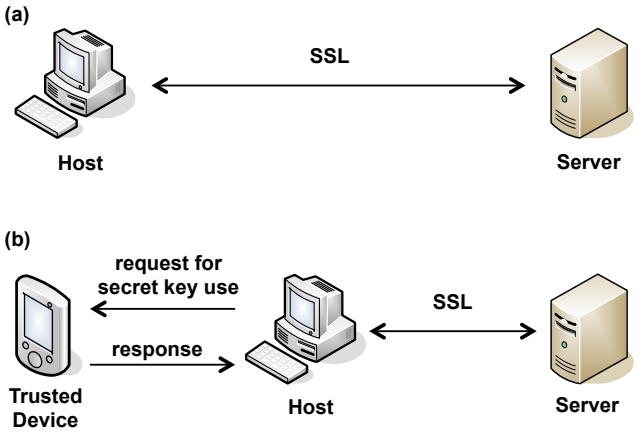

Figure 1: (a) Conventional SSL setup between a host and a server (b) Our DataGuard design with the trusted device to enhance the protection of a cryptographic private key.

3. PROTOCOLS FOR ACCESSING PRIVATE KEYS

In a device-based isolation architecture, cryptographic private keys are stored on a trusted dedicated device, as opposed to a general-purpose networked computer (i.e., host), for the confidentiality of the key. In order to use the key for decryption or signing, the host needs a protocol to utilize the private key *without* learning the key value. In this section, we describe two protocols for using private keys in our DataGuard framework:

1. Decrypting the ciphertext received by the host.
2. Signing an outgoing message.

In what follows, we assume that the DataGuard user (e.g., Alice) generates her public and private keys of a secure public-key cryptosystem (e.g., RSA) on the external trusted device. Her public key is publicly available and certified by a certificate authority, so that others (e.g., Bob) can make sure the public key belongs to Alice.

3.1 Secure Decryption in DataGuard

The secure decryption protocol allows the DataGuard user (Alice) to decrypt the ciphertext received from the Internet. The sender of the message (e.g., Bob) encrypts a secret message using Alice's public key and then sends it to Alice. Alice's host receives the encrypted message and sends it to the trusted device for decryption using the private key stored in the trusted device. Then, the decrypted/original message is displayed on the screen of the trusted device to ensure the confidentiality of the original message. Figure 2 depicts the secure decryption protocol in DataGuard. The detailed operations are as follows:

1. Alice's host receives the ciphertext C from Bob who encrypts the message M with the public key of Alice to ensure the confidentiality of M.
2. The DataGuard daemon on Alice's host initiates a request for connecting to the trusted device.

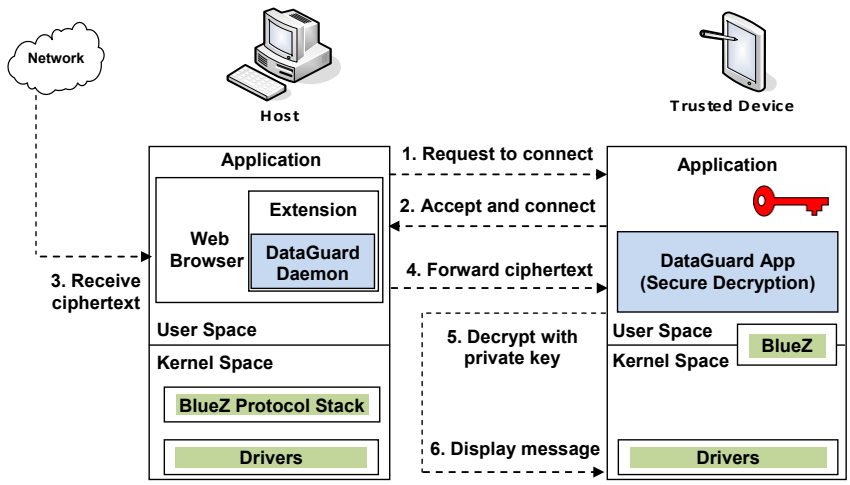

Figure 2: Secure decryption procedure in DataGuard. Red key is the private key of the DataGuard user.

3. The trusted device provides a PIN/passkey to verify its identity before connecting to Alice's host. Upon the verification, the trusted device is paired with Alice's host where the connection is established.
4. Alice's host forwards C to the secure decryption application running on the trusted device.
5. The secure decryption application receives C, and decrypts C with Alice's private key K stored in the trusted device to obtain the message M. Alice may need to enter her passphrase on the trusted device in order to decrypt the private key (see also Section 2).
6. Message M is displayed on the screen of the trusted device.

An application of this secure decryption protocol mechanism is secure email, i.e., decrypting encrypted email messages. The DataGuard user receives an encrypted email. The host relays the encrypted email to the trusted device. The trusted device performs the decryption and displays the plaintext message on its screen. As a demonstration, we implemented the DataGuard daemon as a part of Chrome browser extension. This browser extension gets the encrypted information from a webpage (e.g., Gmail) and sends it to the trusted device for decryption and display, as shown in Figure 2. Details of our prototype including screenshots are in Section 4.

3.2 Secure Signing in DataGuard

In this protocol, Alice uses her external trusted device to digitally sign a message. Whenever Alice needs to send a signed message to Bob, the message is forwarded to the trusted device for signing before Alice sends the message to Bob. The steps of generating a message signature within our framework architecture are as follows.

1. Alice has composed a message M on her untrusted host that she wants to securely sign and send to Bob.
2. The DataGuard daemon on Alice's host initiates a request for connecting to the trusted device.
3. The trusted device provides a PIN/passkey to verify its identity before connecting to Alice's host. Upon the verification, the trusted device is paired with Alice's host where the connection is established.
4. The host sends the message M to be signed to the secure signing application running on the trusted device.
5. The secure signing application receives M, and upon Alice's approval signs M with Alice's private key K stored in the trusted device to generate a digital signature S. S is sent back to the host. (Alice may need to enter her passphrase on the trusted device in order to decrypt the private key, as in the secure decryption protocol in Section 3.1.)
6. Upon receiving S, the host sends M together with S to Bob.
7. Bob receives M and verifies the authenticity of S with respect to M and the public key of Alice.

An application of the secure signing protocol is electronic purchasing where a customer sends a signed purchase order to a vendor. The customer (e.g., Alice) fills in the purchase order form on the host. Then, the daemon on Alice's host sends the form to the trusted device for signing with the private key of the customer. The generated signature is sent back to the host where the customer can submit the form to the vendor to complete the purchase order. As a demonstration of its application, we implemented the DataGuard daemon as a Chrome browser extension in our prototype. This web browser extension can get the sensitive information from a webpage and send it to the trusted device for signing. Details of our prototype implementation are in Section 4.

The secure signing protocol supported by our framework is essential for data security. The digital signature ensures the authenticity of the message by preventing the others from pretending to be the creator of the message. Moreover, the digital signature ensures the integrity of the message by ensuring that the content of the message has not been tampered with since it was digitally signed.

4. IMPLEMENTATION

In this section, we describe the implementation details of our DataGuard prototype. DataGuard applications (Secure Decryption App and Secure Signing App) are developed on Google Android 2.1. Android is a software stack for mobile devices that includes an operating system, middleware and key applications [6]. Android provides a Java-based development platform and supports Java SDK 1.6 as we use Java Cryptographic libraries to provide security features such as java.security.* and java.crypto.*. We implement the DataGuard daemon as a stand-alone program, and as a web browser extension for Google Chrome to demonstrate how it easily integrates with the browser.

4.1 Communication Interface

DataGuard architecture requires the establishment of a secure channel between the host and the trusted device. Therefore, we implemented and supported two types of communication methods: *i)* wired using universal serial bus (USB) and *ii)* wireless using Bluetooth.

We focus on Bluetooth because it is more convenient to use. It supports a Service Discoverable Protocol (SDP) which allows each service to bind to a UUID, and supports encryption for the communication based on the passkey agreed by the communicating devices which ensures the confidentiality of the communication. Bluetooth is publicly available in almost all the devices including mobile devices and laptops.

BlueZ [3] is the Linux Bluetooth Protocol Stack. BlueZ is used in our prototype. We support the following functions during the implementations of Bluetooth communication:

- **Bi-directional Bluetooth communication**: it is set up by directly accessing the RFCOMM communication channel which is a serial port emulation protocol. RFCOMM provides a simple reliable data stream to the application.
- **Bluetooth message parsing**: we wrap the internal DataGuard message data structure in the payload of the Bluetooth message. Handling the message is the key to maintain a reliable interaction between the DataGuard daemon on the host and the DataGuard applications on the trusted device.
- **Bluetooth message construction and response**: the message data structure consists of a header with both the message type and message size inside, followed by the actual message payload.

4.2 Implementation of DataGuard Daemon and Applications

The DataGuard daemon runs as a background daemon in the host. It listens to the communication channel for an allowed connection from the trusted device. The DataGuard daemon is written in C and it has 661 lines of code. The daemon is designed to only contain limited logic which simply serves as a relay for the DataGuard applications running on the trusted device. In particular, the daemon performs the following functions:

- **Service registration**: this involves assigning a Universally Unique Identifier (UUID) to the service interface, so that only one device which knows about this UUID is able to connect to it.
- **Forwarding a ciphertext for decryption**: this interface is implemented as a Google Chrome extension to support the secure decryption protocol. We developed this Chrome extension with JavaScript to be able to manipulate the webpage and select the ciphertext to be decrypted. The DataGuard daemon, which is implemented as part of this extension, forwards the selected ciphertext to the trusted device for decrypting and displaying on the trusted device's screen.
- **Obtaining digital signature of a message**: this interface is implemented as a Google Chrome extension to support the digital signing protocol. We developed this Chrome extension with JavaScript to be able to manipulate the webpage and select the message to be signed. The DataGuard daemon, which is implemented as part of this extension, forwards the selected message on the host to the trusted device for obtaining a digital signature.

We developed two Android-based applications to support the security protocols presented in this paper, namely *i)* a secure decryption application which performs the secure decryption service and *ii)* a secure signing application which performs the digital signing service. Both applications provide a graphical user interface to assist the user in using these services. The secure decryption application has 1149 lines of Java code, while the secure signing application has 1249 lines of Java code. Both applications are deployed on the trusted device.

The *secure decryption application* receives the encrypted message from the host to be decrypted with the private key stored in the trusted device. Once the application decrypts the ciphertext, the original message is displayed on the screen of the trusted device to ensure the confidentiality of the message. In our prototype, we support two different algorithms (RSA and AES) with different key lengths to decrypt the ciphertext.

Similarly, the *secure signing application* receives the message from the host to be signed and generates the signature based on the stored private key in the trusted device. In other words, the application takes the message as the input and generates the signature as the output which is sent back to the daemon on the host. We support two different algorithms (DSA and RSA) with different key lengths to digitally sign messages.

The use of an external trusted device in our framework to achieve the isolation is practical and inexpensive. The external trusted device in our solution can be any computing device that satisfies our requirements mentioned in Section 2.1. To further demonstrate this property, we realized our prototype on a Linux device designed for embedded systems in addition to our smartphone and tablet prototypes. We chose to implement our prototype on these devices because they have the capability to run software and services, not just a special hardware which may be expensive to build. In addition, these handheld computing devices are still a commodity computer which is inexpensive.

We programmed an ADVANTECH Box-PC ARK-1360, shown in Figure 3, as an external trusted device which is a small fan-less box that can be carried easily by a user. We installed a BTA-3210 USB 2.0 Micro Bluetooth Dongle, which supports Bluetooth 2.1 with 10 meters effective range. BlueCove [1], a JSR-82 implementation of Java Library for

Figure 3: External trusted device: ADVANTECH Box-PC ARK-1360.

Bluetooth, was used for providing similar Java API for accessing the underlying Bluetooth protocol stack BlueZ. Also, we attached a monitor to ADVANTECH Box-PC to provide screen output capability.

The linux-box device is truly secure isolated device. In order to make a smartphone or a tablet more secure and trusted in practice, some existing solutions can be integrated with our framework, such as [16, 25] to build a secure mode into Android-based devices or provide access control capabilities to grant secure environment in which specific operations can be performed and have limited access to unnecessary resources. Therefore, any smartphone or tablet can be suitable trusted device to be used in practice.

In summary, we have successfully demonstrated our DataGuard design in three independent instantiations of the trusted device with a smartphone, a tablet computer, and a Linux embedded device.

4.3 Integration with Chrome Browser on the Host

We implement the DataGuard daemon as a web browser extension for Google Chrome to show the practicality of our framework. Specifically, we implemented the DataGuard daemon as a plugin based on the Netscape Plugin Application Programming Interface (NPAPI) to be able to use it as a browser extension. In addition, we used Nixysa [4] to automatically generate glue code for NPAPI plugins to expose the code written in C to JavaScript.

We developed two Chrome extensions, namely a secure decryption extension and a secure digital signing extension. The secure decryption extension performs two main functions: *i)* it accesses and parses the webpage to select the ciphertext which needs to be decrypted, and *ii)* it forwards the selected ciphertext to the trusted device for decryption and display. Figure 4 shows the screenshots of the secure decryption extension and the trusted device. In this example the user receives an encrypted email message in Gmail and the extension forwards the encrypted email to the trusted device for decryption and display. The host uses HTTPS for its connection with the remote Gmail server. Similarly, the secure digital signing extension accesses and parses the webpage to select the message to be signed, and forwards the selected message to the trusted device for obtaining a digital signature which is sent back to the extension.

4.4 Security Analysis of DataGuard

In this section, we analyze the security guarantees of DataGuard in terms of the three security goals set in Section 1 – key confidentiality, message confidentiality in decryption, and message integrity in signing.

Key Confidentiality. One of the primary strengths of DataGuard is the protection of the private key confidentiality. Our framework provides secure storage for cryptographic private keys by isolating and storing them on a dedicated trusted device instead of keeping them on the untrusted host. All cryptographic operations involving the private keys take place in the external trusted device as well. Therefore, the values of private keys are not revealed to the host and the confidentiality of private keys is preserved. We ensure the confidentiality of the private keys even if the host is compromised.

Message Confidentiality. In the secure decryption protocol, the confidentiality of the message is preserved because the decryption is performed and displayed on the trusted device. The host only has the access to the ciphertext. With a (semantically) secure encryption scheme, the host learns nothing about the message from the ciphertext.

Message Integrity. In the secure signing protocol, the integrity of the message can be preserved and signature cannot be forged, as the digital signature can only be generated by the trusted device uniquely possessing the private key. Therefore, all three security goals are achieved in DataGuard.

Bluetooth is chosen to provide bi-directional communication channel in our solution. The host and the trusted device are securely connected to each other by identifying the Bluetooth service ID, where the Universal Unique Identifier is used before they get connected with passkey agreed by them. In addition, all communication between them can be encrypted and authenticated using standard protocols. Therefore, the attacker will neither be able to eavesdrop on the communication channel nor masquerade as a real host or as a trusted device. The trusted device has secure communication with the host.

We consider the case where the DataGuard daemon or the OS kernel is tampered with by stealthy malware that escapes any detection on the host (e.g., anti-virus scan). The confidentiality of the private key is still preserved, as its storage and involved operations are isolated on the trusted device. In addition, the message confidentiality in the secure decryption protocol and the signature unforgeability in the secure signing protocol hold as well. The host controlled by the attacker may ask the trusted device to sign arbitrary messages, thus the signing person (e.g., Alice) who controls the trusted device is responsible for distinguishing the messages that should be signed from rogue ones.

However, the availability of decrypting and signing services provided by the trusted device may not be guaranteed in the face of compromised host. The malware-controlled host may drop packets in the communication with the trusted device or refuse to pair with the trusted device or to transmit data, which is undesirable. Such an issue is out of the scope of DataGuard security model. A mitigation strategy is to disinfect the host as soon as any disruption in the communication is discovered. In an event, Alice will be able to easily detect such forms of attack and take appropriate countermeasures.

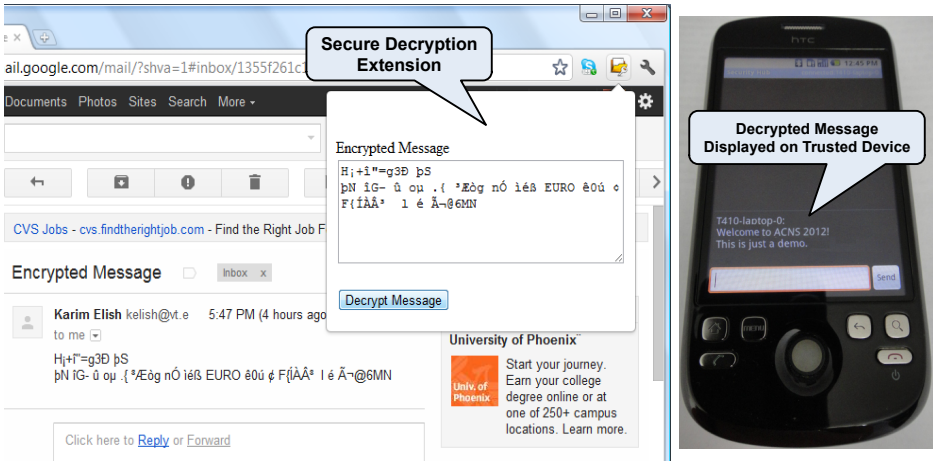

Figure 4: Screenshots of DataGuard prototype showing the secure decryption extension on the host and the trusted device. The ciphertext received by the host via Gmail is decrypted on the handheld without exposing the private key to the untrusted host.

5. PERFORMANCE EVALUATION

We conduct several experiments to evaluate the efficiency of our DataGuard prototypes, including the running time of cryptographic operations on the trusted device and communication bandwidth between the host and the trusted device.

We use a Lenovo T410 laptop as a host which has 2.40GHz dual core Intel Core i5 processors, 2GB memory, a Bluetooth chipset compatible with Bluetooth 2.1 specification, and Ubuntu 10.04 LTS. We perform our evaluations on two different trusted handheld devices: Android Development Phone (ADP) and Samsung Galaxy Tab 10.1. Both of them meet the requirements for the trusted device described in Section 2.1. DataGuard applications (namely Secure Decryption App and Secure Signing App) are deployed on them. Android Development Phone (ADP) is a smartphone that runs Google Android OS 2.1 and has a 576MHz processor with 512MB ROM, 288MB RAM, and Bluetooth 2.0. Samsung Galaxy Tab 10.1 is a tablet computer that runs Google Android OS 3.1 and has a dual-core 1GHz processor, 32GB ROM, 1GB RAM, and Bluetooth 2.1. We ran our experiments on both ADP and tablet and reported the results from both devices. All results are averages of five runs [1].

We measured the Bluetooth throughput to examine the empirical communication bandwidth of Bluetooth between the host and the trusted device [2]. In our experiments, the host sends various amount of data (ranging from 10KB to 100MB) to the handheld devices (both ADP and tablet, respectively), and we measure the duration of the transmission. Our experimental results show that a maximum throughput of 2Mbps can be achieved on both handheld devices, which is sufficient for the purpose of DataGuard. The empirical throughput reported by others previously is 2.1Mbps [2]. We did not measure the bandwidth of USB communication but it is expected to be up to the manufacturer's maximum throughput which depends on the USB standard used.

Although the descriptions in previous sections are based on public-key cryptosystems, DataGuard supports symmetric-key cryptosystems such as AES for encryption and DSA for digital signature. For completeness, we implement and evaluate all relevant operations of both public-key and symmetric-key schemes in DataGuard, including encryption/decryption and signing/verification.

Figure 5 shows the decryption performance of different ciphertext sizes on both handheld devices by using AES (128-bit, 192-bit and 256-bit keys) and RSA (512-bit, 1024-bit, and 2048-bit keys), respectively.

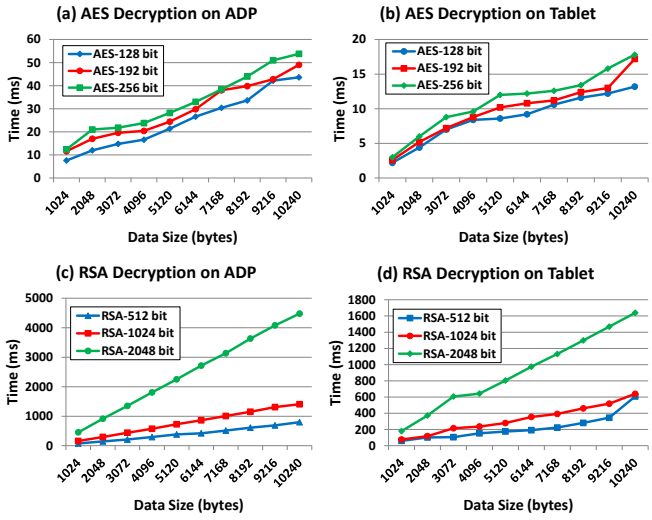

Figure 5: Decryption performance of AES and RSA on handheld devices.

We also measured the performance of encryption on both handheld devices in AES and RSA. The performance of symmetric key decryption/encryption (AES) is faster than RSA as expected. (Encryption results are not shown.) In practice, the public-key encryption scheme such as RSA is only used for encrypting (short) symmetric keys (e.g., AES keys) that are used for encrypting the message. Therefore, we

[1]Variances are negligible and not shown.

[2]The theoretical throughput of Bluetooth (2.0 and 2.1) reported by the manufacturer is up to 3Mbps.

expect the decryption time on the handheld devices to be fast.

Figure 6 shows the signing and verification performance on both handheld devices by using DSA (512-bit, 1024-bit) keys and RSA (512-bit, 1024-bit, 2048-bit) keys, respectively.

RSA with 512-bit and 1024-bit keys are much faster than 2048-bit key as shown in Figures 6(a) and 6(b). The tablet performs cryptographic signing operation faster than ADP as expected. Signing in DSA with 512-bit and 1024-bit keys is faster than RSA with the same key lengths on both computing devices, which is expected. However, for signature verification, RSA with 512-bit and 1024-bit keys is slightly faster than DSA with the same key lengths, respectively.

Our experimental results show feasible performance of DataGuard as all cryptographic operations can be performed efficiently on the external trusted handheld devices without significant overhead, and the communication bandwidth of Bluetooth is sufficient for transmitting large amounts of data (e.g., ciphertext or digital signature) in a timely fashion.

6. RELATED WORK

In this section we review and discuss related work on external device-based security. Moreover, we briefly review recent works on studying the security of mobile platform especially for Android OS since we focus our implementation on Android-based mobile handheld devices.

McCune et al. [18] proposed a framework called Bump in the Ether (BitE) to ensure the integrity of sensitive user input to the applications running on the host. BitE captures the user's keypress on the keyboard and send it to a trusted mobile device to encrypt the keyboard events. The mobile device, in turn, sends the encrypted keyboard events to specific application on the host, where they can be decrypted and used. In [19], the authors introduced a Bumpy system that protects the sensitive user input for web applications by processing the input in an isolated code module on the user's system, where they can be processed for a remote webserver. Our work differs from BitE and Bumpy in that they address the integrity of user input to the applications, while we address the confidentiality requirement of secret data (cryptographic keys) used in sensitive operations (decryption and signing). Bumpy uses a different approach for protecting user inputs, i.e. isolated code module on the user's system not an external trusted device for processing or encrypting the inputs.

Storage Capsules system is a recent approach introduced by Borders et al. [7] to protect confidential files on a personal computer. It allows a compromised host to securely view and edit sensitive files by isolating the primary operating system in a virtual machine and disabling network and other device output before it is accessing confidential files. Once the files editing is done in Storage Capsules, it restores system state to a snapshot taken before accessing Capsule system and resumes network and other device output. Storage Capsule can be considered as an alternative to our device-based isolation solution where private keys can be stored and used in a secure manner. However, Storage Capsule performs the isolation on the same machine, while our work uses a separate device for isolation to achieve the data confidentiality.

Zic and Nepal [28] designed and implemented a prototype personal trusted device called the Trust Extension Device (TED) that provides users with a portable trustworthy environment for performing transactions on computer which is connected to Internet. TED is implemented on a small portable device i.e. flash drive. Also, Nepal et al. [20] designed and built a mobile and portable Trusted Computing Platform (TCP) based on Trusted Computing Group (TCG) [5] specification. Their solution provides a TCP on a USB device to enable the mobility and portability of trust for enterprise applications. The solutions in [28] and [20] do not provide a complete isolation from untrusted host as their proposed device does not have any input/output devices on its own, instead it relies on the input/output devices of the untrusted host machine. Hence, these I/O devices are not isolated and subjected to malicious attacks such as keyboard loggers and screen scrapers. On the other hand, our work offloads all the security features to be run on an external trusted device and does not relay on I/O devices of the host which provides a complete isolation from untrusted host.

Some solutions used mobile handheld devices to provide secure authentication techniques [17, 27], provide secure remote access to the user's home computing environment [23], propose a realistic mobile ticket system [9], and enhance web browsing security on public terminals [26].

A different, but related in the spirit of data protection, solution is represented by Samarati et al. [24], where the authors addressed the privacy issues such as data confidentiality in data outsourcing scenarios. Another different, but related, technique is represented by [8], where the authors proposed a new protocol for authentication using minimally trusted servers.

A number of recent proposals have studied Android platform security to identify potential security risks of Android OS [10, 12, 14], or to improve the overall security of Android-based mobile devices [13, 21, 22]. Chin et al. [10] analyzed the vulnerability in inter-application communication in Android apps and found a number of exploitable vulnerabilities. Enck et al. [12] analyzed the security of 1,100 top free Android apps from the official Android Market to understand security characteristics such as use/misuse of personal/phone identifiers information. Felt et al. [14] studied 940 Android apps and found that about 1/3 of the studied apps are requesting more permissions than they need. Kirin [13] framework provides a lightweight certification of Android apps at install time to block the installation of potential unsafe apps based on certain undesirable permission combination. Ongtang et al. [21] proposed Porscha framework to provide content protection for Android by enforcing security policies. Saint [22] is another framework which proposes install-time permission granting policies and runtime inter-application communication policies to address the current limitations of Android security. The existing work reviewed above on Android security is vital to enhance the security of Android Mobile OS and can be integrated to our work to provide a generic robust solution for enhancing data security on the host.

7. CONCLUSION AND FUTURE WORK

With the proliferation of personal handheld computing devices, such as smartphones and tablet computers, we proposed a device-based isolation approach and its prototype called *DataGuard* to protect the secrecy of the highly sensitive data (namely cryptographic keys) through the storage isolation and secure tunneling. Our solution does not require highly specialized (system-specific) services and hence

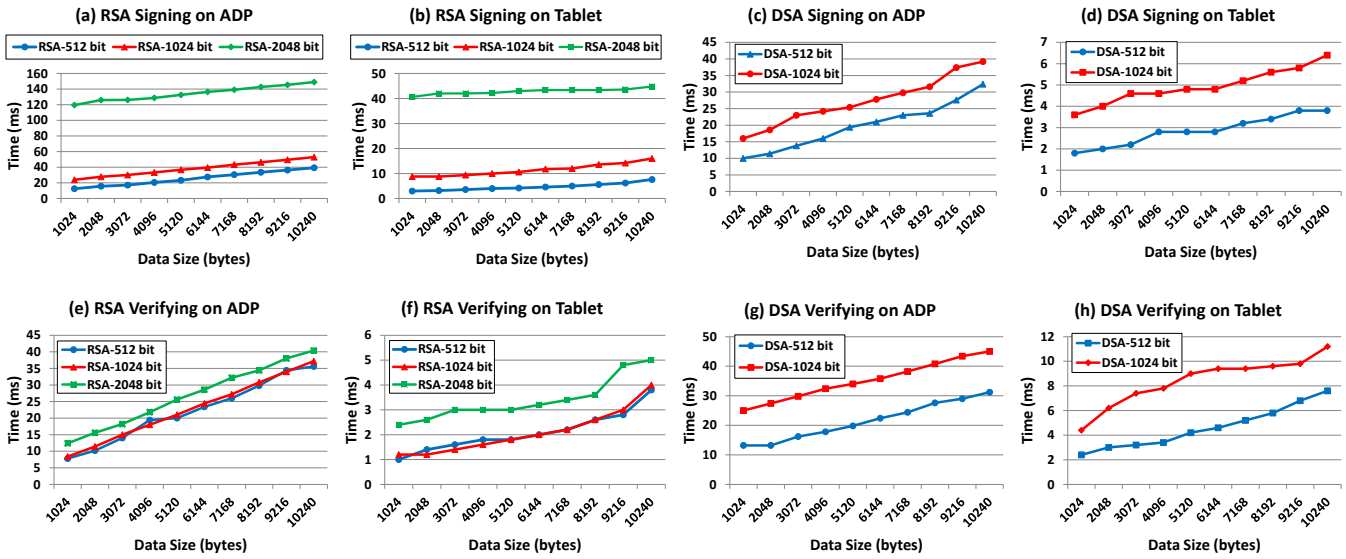

Figure 6: Digital signing and verification performance of RSA and DSA on handheld devices.

it would be easily ported to other platforms. We presented our design and implementation of DataGuard framework along with two security protocols to illustrate how DataGuard is deployed in practice. We conducted extensive experiments to evaluate the feasibility and performance of DataGuard. The results show that our method performs well without significant overhead.

For future work, we plan to explore other applications that utilize mobile handheld devices to provide desirable security enhancement to personal computers. For example, the handheld device may serve as the attestor in the TPM attestation that checks and verifies the system integrity of a TPM-enabled host. We have showed in our paper that our solution is simple to adopt. We plan to perform more systematic studies to evaluate the usability of our approach. We plan to find out users' experiences in using an external device to store secret keys and perform cryptographic operations. The findings will help us improve the transparency of our solution, which will provide a seamless experience to users besides enhanced security guarantees.

8. REFERENCES

[1] Bluecove. http://bluecove.org/.

[2] Bluetooth measuremet fundamentals. http://www.home.agilent.com/agilent/.

[3] Bluez: What is bluez? http://BlueZ.org.

[4] Nixysa. http://code.google.com/p/nixysa.

[5] Trusted Computing Group. http://www.trustedcomputinggroup.org.

[6] What is Android? http://developer.android.com/guide/basics/what-is-android.html.

[7] K. Borders, E. V. Weele, B. Lau, and A. Prakash. Protecting confidential data on personal computers with storage capsules. In *18th USENIX Security Symposium*, pages 367–382. USENIX Association, 2009.

[8] L. Chen, D. Gollmann, and C. J. Mitchell. Authentication using minimally trusted servers. *ACM SIGOPS Operating Systems Review*, 31:16–28, July 1997.

[9] Y.-Y. Chen, C.-L. Chen, and J.-K. Jan. A mobile ticket system based on personal trusted device. *Wireless Personal Communications: An International Journal*, 40:569–578, March 2007.

[10] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 239–252. ACM, 2011.

[11] F. Cohen. Computer viruses theory and experiments. *Computers and Security*, 6:22 – 35, 1987.

[12] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In *Proceedings of the 20th USENIX conference on Security*. USENIX Association, 2011.

[13] W. Enck, M. Ongtang, and P. D. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 235–245. ACM, 2009.

[14] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 627–638. ACM, 2011.

[15] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and personalized computing on public kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 199–210. ACM, 2008.

[16] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter. L4Android: a generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in*

smartphones and mobile devices, pages 39–50, New York, NY, USA, 2011. ACM.

[17] M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *11th International Conference on Financial Cryptography and Data Security, and 1st International Workshop on Usable Security*, Lecture Notes in Computer Science, pages 88–103. Springer, 2007.

[18] J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the ether: A framework for securing sensitive user input. In *USENIX Annual Technical Conference, General Track*, pages 185–198, 2006.

[19] J. M. McCune, A. Perrig, and M. K. Reiter. Safe passage for passwords and other sensitive data. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2009.

[20] S. Nepal, J. Zic, D. Liu, and J. Jang. A mobile and portable trusted computing platform. *EURASIP Journal on Wireless Communications and Networking*, 75, 2011.

[21] M. Ongtang, K. R. B. Butler, and P. D. McDaniel. Porscha: policy oriented secure content handling in Android. In *26 Annual Computer Security Applications Conference (ACSAC)*, pages 221–230. ACM, 2010.

[22] M. Ongtang, S. E. McLaughlin, W. Enck, and P. D. McDaniel. Semantically rich application-centric security in Android. In *25 Annual Computer Security Applications Conference (ACSAC)*, pages 340–349. IEEE Computer Society, 2009.

[23] A. Oprea, D. Balfanz, G. Durfee, and D. K. Smetters. Securing a remote terminal application with a mobile trusted device. In *20th Annual Computer Security Applications Conference (ACSAC)*, pages 438–447. IEEE Computer Society, 2004.

[24] P. Samarati and S. De Capitani di Vimercati. Data protection in outsourcing scenarios: Issues and directions, April 2010.

[25] A. Shabtai, Y. Fledel, and Y. Elovici. Securing Android-powered mobile devices using SELinux. *IEEE Security and Privacy*, 8(3):36–44, 2010.

[26] R. Sharp, A. Madhavapeddy, R. Want, and T. Pering. Enhancing web browsing security on public terminals using mobile composition. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 94–105. ACM, 2008.

[27] G. Starnberger, L. Froihofer, and K. M. Göschka. QRTAN: Secure mobile transaction authentication. In *Proceedings of the The Forth International Conference on Availability, Reliability and Security (ARES)*, pages 578–583. IEEE Computer Society, 2009.

[28] J. Zic and S. Nepal. Implementing a portable trusted environment. In *Future of Trust in Computing*, pages 17–29, 2009.