Trustworthy and Portable Emulation Platform for Digital Preservation

Zahra Tarkhani
Computer Science Department
Indiana University Bloomington
ztarkhan@indiana.edu

Geoffrey Brown
Computer Science Department
Indiana University Bloomington
geobrown@indiana.edu

Steven Myers
Computer Science Department
Indiana University Bloomington
samyers@indiana.edu

ABSTRACT

We present an architecture for a trustworthy and portable emulation platform designed to protect the confidentiality and integrity of sensitive born-digital content when executed on a fundamentally untrustworthy platform. In evidence, we present a modified GameBoy emulator which is executable on a remote user platform while simultaneously protecting the contents of game ROM files. In a more general application, an archive or a library might use such an emulation architecture to control access to restricted material on more sophisticated computer emulators.

Our solution relies on Intel's Software Guard Extensions (SGX) technology for implementation of the trusted emulation environment. Access to sensitive data is protected by server controlled encryption keys accessible only within the protected execution environment. This enables secure caching of encrypted data on the untrusted user platform for use by the emulator and hence limits the potential performance issues originated from remote execution over Internet connections.

KEYWORDS

secure digital preservation, trustworthy emulation, malicious operating systems

1 INTRODUCTION

This paper provides a technological solution to a fundamental problem faced by libraries and archives with respect to digital preservation — how to allow patrons remote access to digital materials while limiting the risk of unauthorized copying. The solution we present allows patrons to execute trusted software on an untrusted platform; the example we explore is a game emulator which provides a convenient prototype to consider many fundamental issues.

Examples of born-digital content include commercial and academic publications, such as interactive CDROMs as well as the archived works of writers and statesmen. For example, Emory University provides access to the processed portions of Salman Rushdie's digital archive, and an interactive simulation of Rushdie's original computing environment[16]. Access is only possible via an emulator located on a dedicated workstation in the reading room that limits access to the content.

While emulation is the main preservation strategy for legacy applications that require obsolete software for access [10, 28–30, 36, 37, 43], the approach we propose can also be used to secure access born-digital contents that are not being preserved by emulation platforms. For example, consider the case of PDF documents: while PDF encryption is widely supported [44] accessing such documents requires distributing keys; the approach we propose might be used to execute a trusted PDF viewer on an untrusted platform that is able to verify the untrusted platform and protect keys/documents.

One fundamental limitation of our approach is that anything the user is allowed to view or hear could be captured through screenshots or audio recordings. That is, our solution is susceptible to attacks on the output devices, just as all traditional digital-rights management systems are. However, this is an issue with current solutions even within the confines of a library or archive, unless patrons are strictly monitored (e.g. by impounding cell phones). Our solution does enable a new use case for born-digital sharing, even if tight controls are necessary — archives with restricted reading rooms might use our approach to safely share materials with other archives that provide similar physical security. The architecture can also be used to protect the content against unauthorized access from servers that provide web/cloud based emulation services.

Figure 1 illustrates the major components of the type of system we explore: an emulator, executing on the client side, accesses secure storage containing both the emulation system software stack (e.g. a disk image containing an operating system) and any digital content to be accessed. Because we cannot guarantee the security of the client operating system or file storage, we require a trusted emulator that has controlled access to any secured materials. Our proposed architecture is based upon several key assumptions:

- The remote platform is under full control of the user, and the institution does not trust any of its software or hardware components. Therefore, establishing trust between the server and client is essential. Trust establishment allows the server to look at the execution platform and verify

key properties that are required for the trusted application to protect its secrets on the remote platform. For example, in our solution, the server needs to verify that the remote platform uses a real SGX enabled CPU.

- The emulator provided by the source institution runs on the remote platform and relies on its resources. This means that the emulator is managed by the untrusted client-side operating system, which itself may have been subverted by malware.
- All sensitive materials on the server-side storage are encrypted, and there is a secure protocol to transmit the encrypted files and keys from the server to the emulation client after trust establishment.

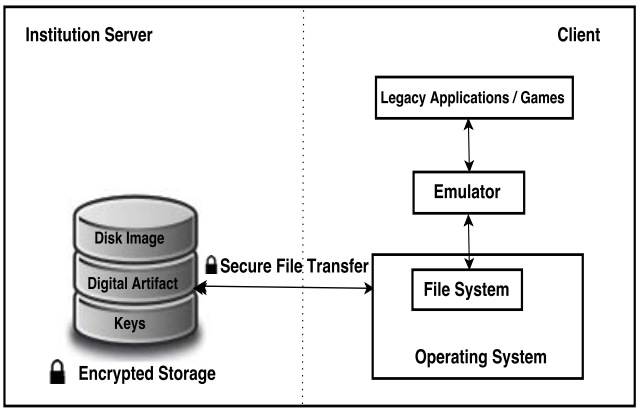

Figure 1: Generic system architecture

Delivering sensitive content to an untrustworthy remote platform is very challenging. Solutions that rely only on data encryption and access control mechanisms are vulnerable to several software and hardware attacks. For example, a compromised OS may steal encryption keys and disk image contents from a running emulator. This paper will address the following security problems:

- **Trustworthy Emulator**
  The emulator core, which has access to keys and unencrypted security sensitive data, is meant to be kept private from the user on the client side. Therefore, such data and keys must be secured by the emulator. The emulator should also support on-the-fly decryption/encryption of data stored on the local file system (see Figure 2).
- **Trust Establishment**
  The server should be able to authenticate the user and verify his/her platform before exchanging keys and encrypted files (see Arrows (1) and (2) in Figure 2). The server should also be able to ensure that the client's trusted emulator behaves correctly. That means the client should be able to **attest** to the fact that all requirements (e.g., running a right version of the emulator) for the emulator to behave as expected (e.g., protecting the restricted data) are satisfied.

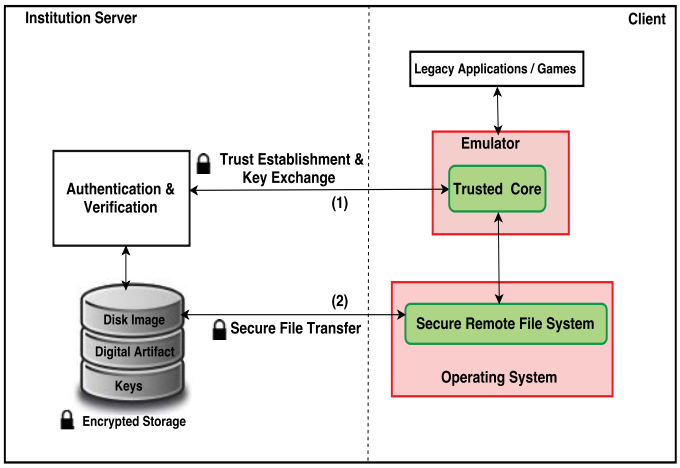

Figure 2: Generic security model

Many different methods for solving both of these problems have been proposed [5, 6, 8, 12, 19, 20, 25, 26, 33, 34], but mostly they are based on modified OS, hypervisor, or compiler. Our approach relies only on Intel SGX hardware support that provides protected execution environments called enclaves. The enclaves allow execution of software within them such that others, including the owner or administrator of the machine, cannot peer into or modify–it is as if the code executes in an impenetrable black box. SGX further provides a hardware-based mechanism to attest to the software running in the enclave. All Intel Skylake processors support SGX, and thus it should be widely available in the future. At the time of writing this paper, The firmware(BIOS) and the drivers required for SGX are available for Windows and Linux operating systems. Applications that use SGX are currently most easily developed in the C/C++ languages, as Intel has released development tools for them.

The direct interface through assembly is both documented and possible, suggesting that a large number of development tools will be developed in due time. In section 3.1 we summarize SGX functionality related to our work. Although our solution relies on SGX, one may consider using other trusted execution environment (TEE) solutions such as ARM TrustZone technology[1] to implement our architecture.

As a concrete example of the proposed system architecture, we developed a prototype of our solution using GearBoy, an open source emulator for GameBoy. The GearBoy architecture is described in Section 3.3. While developing this system, our focus was to protect materials on the client side and we assumed a trusted server. However, the proposed techniques

can easily be applied to protecting resources on server-side and dedicated workstations as well.

The remainder of this paper is organized as follows. We begin with a review of related work in Section 2. For background, in Section 3 we explain required security concepts and the basic architecture of GearBoy, the GameBoy emulator. Next, in Section 4 we describe the architecture of our trustworthy emulator which is utilizing trusted hardware based enclaves. Finally, implementation details and results are provided in Section 5.

2 RELATED WORK

We survey related work in two areas: emulation as a digital preservation technique and systems that execute on and protect applications from an untrusted OS.

Emulation as a digital preservation technique
Emulation has been successfully tested to provide access to i) obsolete software and hardware systems [4, 7, 27, 32, 35], ii) preserved digital artifacts [3, 22, 31], and iii) console games [11, 23]. However, there are fundamental problems that must be resolved when providing remote access to sensitive preserved content.

One approach to provide remote access to born-digital materials is via a remote desktop or web-based access to emulation environments. The bwFLA project [27] implements a distributed framework and a networked approach, where the emulators run in a well-controlled environment. GRATE [38] provides a web-based remote emulation. The GRATE server is responsible for session management (managing VNC sessions, executing emulators, etc.) and transporting uploaded digital objects into the emulated environments. Although these systems can reduce technical hurdles on the client's side, they provide limited security and are not practical solutions for highly sensitive collections. For example, it is possible to steal the digital objects from the servers through RDP/VNC vulnerabilities [39].

The KEEP project[4] transfers a complex service stack (it consists of a core application, a software archive, and emulator archive) to the remote user, but doesn't address the security issues considered in this paper.

Systems to protect apps from an untrusted OS

Considerable effort in the research community have been devoted to protecting security-sensitive applications from a compromised OS (or other privileged system code) running on an untrustworthy platform[2, 5, 6, 8, 12, 18–20, 47]. While these works have resolved many of the issues, they have many drawbacks that need to be addressed.

Many of the solutions rely on trusted hypervisors to protect applications from a malicious OS [5, 12, 34, 46, 47]; however, they do not offer any protection against a compromised hypervisor. Our solution does not need a trusted hypervisor but does require SGX support, which has a smaller and more securely deployable code-base in comparison (small code bases developed by high-quality teams, e.g. Intel, are more amenable to the application of development tools and techniques). Among the proposed secure systems, some incur huge performance loss due to the costly encryption/decryption operations [5, 12, 20], or they require significant modifications to legacy applications [19, 20]. Another major approach is isolating applications in a dedicated VM [8, 34], which dramatically increases the size of trusted computing base (TCB) and therefore weakens their security strength. Systems like Haven[2] shields applications from an untrusted OS using a secure library OS, based on Drawbridge [24], which works inside an enclave. A library OS is a lightweight OS that runs in the address space of an application and only supports the minimal OS features which the application depends on. The approach taken by Haven requires building a customized libraryOS to support legacy emulator/application dependencies and also has a large TCB size.

3 BACKGROUND

In this section, we briefly describe SGX and its features used in developing our trusted emulator. We also review the trust establishment problem and explain how SGX's attestation capability can be used to solve this problem. Finally, we briefly describe the GameBoy emulator architecture that we modified to build our trusted emulator. It is important to note that while our work builds upon Intel-specific features, it is reasonable to assume that this or equivalent features will be commonplace in the near future.

3.1 Software Guard Extensions (SGX)

The key technology that we exploit in implementing our system is Intel Software Guard Extensions (SGX)[21] that is a new technology added to the Intel Skylake family of processors to aid in trusted computation on remote platforms. SGX provides a set of new instructions and new memory access protections for the Intel x86 computing architecture. Collectively these extensions enable the creation of so-called "enclaves". These are figurative black boxes in which secure code can be loaded and executed; even the operating system — the ultimate arbiter and controller of the machine— cannot peer into the box to inspect computation or its memory accesses. Through an attestation process, remote entities can confirm that particular code is running in the enclave. In the most recent application development tools, enclaves are introduced to developers through a metaphor of trusted shared libraries.

Because enclaves isolate sensitive code and associated data from the host environment, they can securely handle secrets

without fear of release to the host. In particular, enclaves are useful for managing cryptographic keys.

The mechanisms for how SGX protects the integrity and privacy of programs executing within an enclave are quite technical, but at a high-level, the CPU ensures that processes outside of the enclave do have access to read or manipulate the data within an enclave. This is done using physical capabilities provided by CPU (i.e., the CPU's architecture and microcode), and by ensuring that all memory reads and writes are encrypted and authenticated using cryptographic methods. This is implemented using a secret-key that is created on the CPU, and is only accessible to it—that is, it is not even directly accessible within the enclave to programs operating within it.

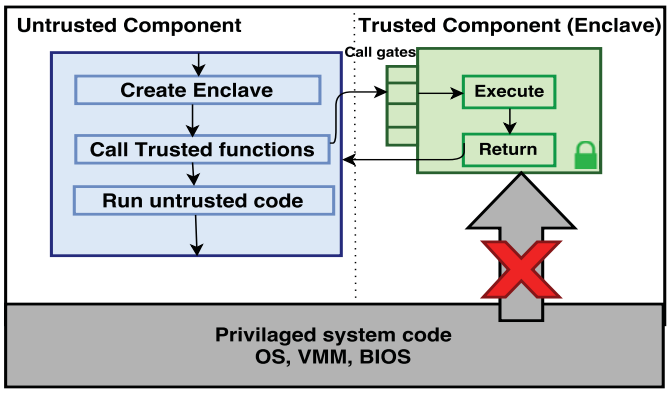

Figure 3: SGX-enabled application model

An SGX-enabled trustworthy application is formed by two classes of trusted and untrusted components (see Figure 3).

- **Trusted component**
  The application's trusted code (i.e., the code that has access to and processes the application's secrets) which works inside an enclave. A trusted application can have several trusted components that reside in one or more enclaves. If necessary, multiple enclaves can arrange to communicate amongst themselves securely.
- **Untrusted component**
  The application's code that doesn't process sensitive data and doesn't need to be protected inside an enclave.

Untrusted code creates and runs the enclave which is placed in the protected memory. When a trusted function is called, only the code running inside the enclave sees the sensitive data in a non-encrypted form, and any external access to the data is denied. An enclave only returns trusted function results to the untrusted component and the enclave data remains in the trusted memory.

3.2 Trust Establishment

This section includes technical details about how SGX can be used for trust establishment. Consider the scenario in which a trusted application (in our case an emulator) behaves correctly on a real SGX enabled platform and can protect sensitive data while being processed. When the application is running on an untrusted platform, the application's provider needs mechanisms to verify the integrity of the remote platform and assure that it has not been tampered with. For example, the provider may need to verify that it is communicating with a real SGX-enabled platform and not with a malicious SGX emulator. It also needs to be able to ensure that the application behaves correctly and as expected, and hence, will be able to protect the critical information. After this verification step, it will be safe for the provider to send its encrypted data/ key, via a secure channel (established by traditional public-key cryptographic mechanisms and with different keying data than those needed for the emulator), to the remote platform to be processed by the trusted application.

Providing secrecy of computation is important for our goals, but without a mechanism for ensuring that the code one believes is running in an enclave is the *actual* code running, secrecy is meaningless. For example, if the enclave can be loaded with malicious code that would "export" secret data outside of an enclave to a waiting malicious machine, then secrecy inside the enclave has achieved little. Unsurprisingly, SGX provides mechanisms to demonstrate that the trusted components of an application have been properly instantiated and are securely running within enclaves on a valid Intel SGX-enabled platform. These attestation mechanisms use two measurement registers provided by SGX to uniquely distinguish each enclave based on its code, data, and author. The measurements are created by using cryptographic hash functions which provide what are essentially unique "fingerprint" values. These values are recorded when the enclave is built, and are finalized before enclave execution starts. The CPU then uses a signing key that is i) unique to the CPU; ii) secret and known to no others; and iii) physically burned into the CPU upon its creation. It performs something similar to a digital signature[1] of the unique fingerprint that was created from the measured code in the enclave [14, 15]. Intuitively, this signature allows an initiating party to verify that the correct code has been loaded into an enclave. The signature is verified by confirming from Intel that the signature is indeed valid, and thus that the loaded code is as declared.

[1]Due to the goal of providing privacy, an actual digital signature is not used, but a digital signature is a good first approximation.

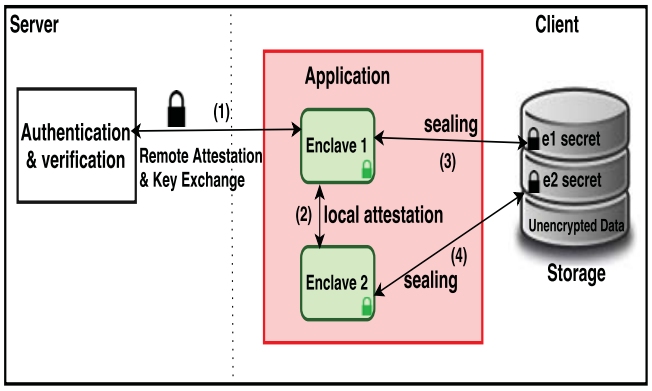

Figure 4: SGX-based attestation and sealing

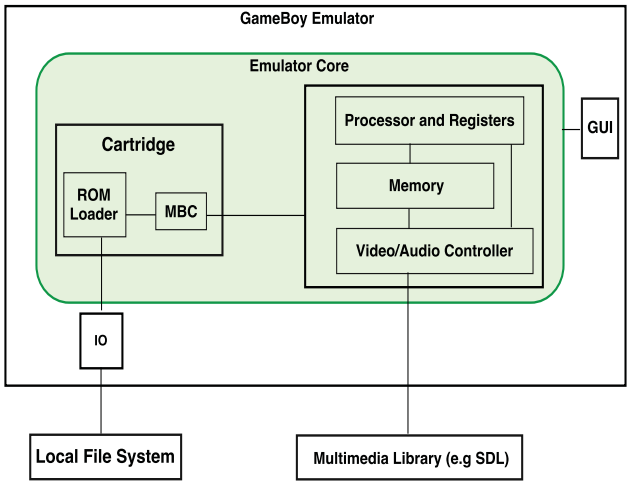

Figure 5: GameBoy emulator architecture

A service provider (institutions, libraries, and archives) may utilize attestation to establish secure communication with a remote platform and provision sensitive materials such as encryption keys to the enclave through a secure channel.

The following example contains technical details that non-technical readers may safely skip.

Consider the scenario, illustrated in Figure 4, in which the application contains two trusted modules, Enclave 1 and Enclave 2 that are running on the same platform. For example, in our trusted GameBoy implementation, one enclave is responsible for trust establishment and key exchange between the service provider and the user. The other enclave contains the secured emulator modules that are responsible for protecting ROM files. The two enclaves can verify and authenticate with one another before exchanging information (see Arrow (2) in Figure 4). This process is called local attestation and is different, but related to the previous attestation methods discussed. After the two enclaves verify that their counterpart is trustworthy, they can exchange information on a secure channel, which provides confidentiality and integrity protection. The local attestation and protected channel establishment use the Diffie-Hellman Key Exchange protocol. After local attestation, the two enclaves trust each other, but the server still needs to verify the trustworthiness of the platform/application before exchanging any secrets.

The server only needs to perform remote attestation with one of the enclaves, Enclave 1 in our example (see Arrow (1) in Figure 4). After the attestation phases, each enclave receives secrets via a secure channel. Next, each enclave can securely encrypt (using a hardware-based encryption key) and store its sensitive data outside the enclave (e.g., on a disk) such that the data can be retrieved only by itself or a trusted enclave (sealing phase, see Arrows (3) and (4) in Figure 4).

3.3 Generic GameBoy Emulator Architecture

In this section, we describe a generic GameBoy emulator architecture and how our trusted emulator architecture can be implemented on it.

The key modules we describe here are part of the trusted or untrusted parts of our trusted emulator. Game Boy is a video game console, originally developed and manufactured by Nintendo in 1989 [40]. There are different GameBoy models. Figure 5 shows the essential components of a GameBoy emulator.

The emulator core contains the major emulated modules of a real GameBoy console. The core includes one or more emulated CPU models. Usually, the CPU is based on a subset of the Zilog Z80 microprocessor and has instructions and registers similar to the Intel 8080 and Intel 8085. It has eight 8-bit registers and two 16-bit registers (SP and PC) [17]. The core contains different memory modules such as video RAM or graphics memory. The amount of such memory is very small (e.g., less than 1MB). GameBoy has no operating system. Therefore, it operates by storing all system functions inside a game ROM cartridge which contains the game's program and data. The cartridge may use memory bank controllers (MBCs) to switch between memory banks and control backup RAM. An emulated cartridge loads a video game's ROM file and maps it to the emulator memory.

Video and audio controllers generate the video/audio timings and read video/audio data from memory in order to output it to screen. The drivers may depend on cross-platform multimedia libraries like SDL[42] to display video/audio.

The core contains all modules that directly access and process ROM files —the secrets we want to protect,— so they should be protected and secured in any emulation. There is

a narrow interface to connect the emulator core to the rest of the emulator. The non-core parts of the emulator implement the graphical user interface and IO operations. These portions don't need to be secured in our model. Through the interface, the trusted and untrusted components communicate in a way that no secret from the trusted code can leak out to the untrusted part.

4 TRUSTWORTHY EMULATOR ARCHITECTURE

In this section, we explain the following key components of our secure emulation architecture, and how these components operate to provide the security level needed for remote accessing to restricted materials (see Figure 6). We begin with a description of a threat model for our system.

4.1 Threat Model

Our goal is to allow possessors of private digital-born media to be able to share this over an untrusted Internet to a foreign client location so that it can be viewed on an emulator at a foreign site. The emulator will provide the appropriate output (audio and video), and those channels are not secured by our system. The assumption is that the foreign client may have malicious software, and be operated by malicious operators. The goal is that the system keeps any data otherwise private. We do not protect against physical attacks which are able to decompose the physical CPU and retrieve physical information that is embedded in the device or otherwise physically alter the execution of the CPU via some physical attack. We assume that the possessor has some way of authenticating the foreign-site to ensure that they are talking to the correct individual; this is traditionally done through password or public-key infrastructure. We also do not provide security against software attacks running on the machine that makes use of memory-page lookups, or timing and cache side-channels. Such attacks are known to be plausible[45], but they require software which explicitly targets the emulator in question and the software running under emulation. These attacks are expensive, as they must be specifically targeted at the emulator in question, and require more technical sophistication than traditional software exploits on the part of the authors. Therefore, such exemptions in the threat model really should provide substantial improvements in real-world security. Further, countermeasures are actively being developed for these attacks which can be incorporated into our model at a future date.

4.2 Key Components of Emulation Architecture

The key components of our emulation architecture are explained in this section.

Trusted Emulator: The emulator contains a trusted component which is hosted inside a secure enclave and it is responsible for protecting the restricted data from the untrusted platform.

Authentication and Verification (AV): The server needs to setup an AV module on both the client and server to authenticate the user and verify untrusted platform. These AV modules are responsible for trust establishment, and to perform a key exchange between the server and client.

Secure Remote File Access: The server needs to provide access to the encrypted files, i.e. sensitive materials we want to protect, via a secure remote file system.

In this architecture, the restricted content provided by the server is encrypted, and the encrypted files and keys will be provisioned to the client after the establishment of trust. On the client side, only the trusted modules, which are working inside SGX enclaves, have access to the keys and unencrypted data. We now elaborate on each component.

4.3 Trusted Emulator

The trusted emulator should be able to protect the sensitive data on the client platform by processing the data inside secured enclaves. In our architecture, only the trusted component executes inside an enclave and thus has access to restricted content. The trusted component includes the core, the trusted interfaces, and the security module (see Figure 6).

The core, which is responsible for processing ROM files, contains the emulated cartridge, processor, registers, memory, and video/audio controllers (see Figure 5). The core communicates outside the enclave through narrow trusted interfaces. The untrusted component is responsible for creating the core enclave and passing the encrypted data to the core to be processed, and thus no enclave protection is necessary. It also includes the emulator graphical user interface (GUI).

Since the restricted materials are encrypted by the service provider, we add a security module to the trusted component which is responsible for on-the-fly encryption/decryption of the data inside the enclave, allowing the core to access the decrypted data, but never making it available outside an enclave. The enclave re-encrypts any data that it needs to store on the local file system. It seals keys and encrypted files before shutting the emulator down (destroying the enclave). Sealing is the process of encrypting enclave secrets to disk for long term storage. Encryption of these keys makes use of long-term secret keys embedded in the CPU, created randomly and burned into it at manufacturing time.

The security module gains access to the keys after ensuring the AV enclave is trusted and running in the same SGX enabled platform through a local attestation during the emulator setup phase. In the local attestation process, two

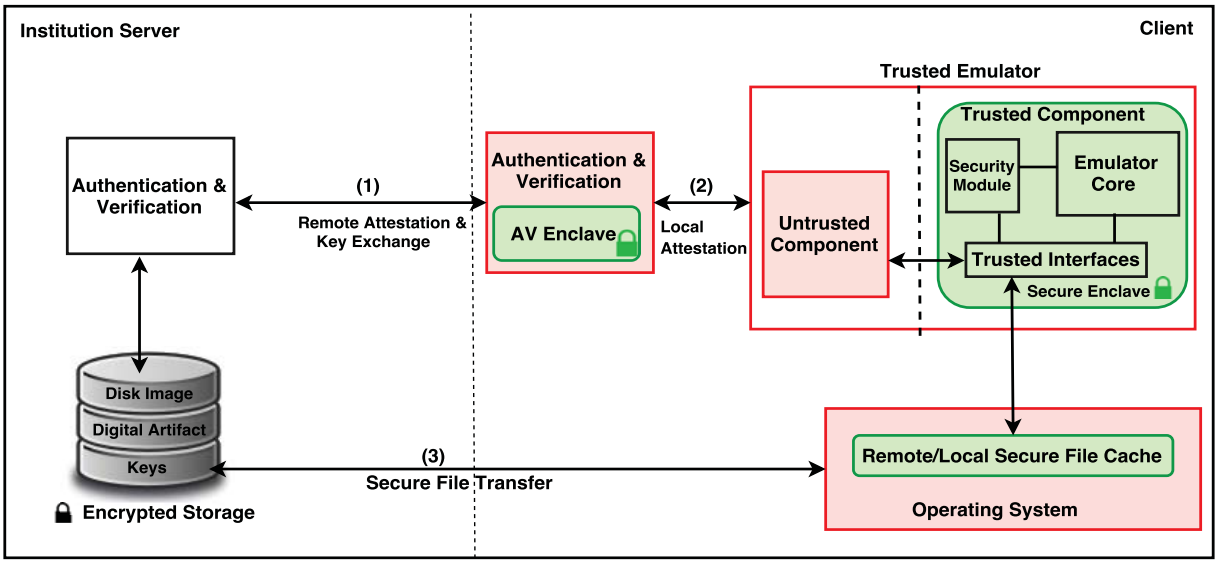

Figure 6: Trustworthy emulator architecture

enclaves executing on the same platform can verify and authenticate with one another before exchanging information (see arrow (2) in Figure 4).

4.3.1 Authentication and Verification. The AV modules (on the server and client side) are responsible for establishing trust between the server and remote platform. The client AV module has a trusted component, called the AV enclave, which is responsible for secure session establishment, attestations and key exchange/store in the remote platform. First, the client AV launches the trusted emulator and ensures the trusted emulator's enclave is instantiated correctly through a local attestation– refers to two enclaves on the same platform authenticating to each other, (see Arrow (2) in Figure 6). After the local attestation with the emulator's enclave, the client AV enclave generates a verifiable report of the client identity and the emulator's enclave. This report is generated and signed by an architectural enclave, called a quoting enclave, and bound to the client platform by the CPU. The server uses this report to verify the platform/emulator on the client side through a remote attestation (see Arrow (1) in Figure 6). In this process, the server verifies the remote attestation report generated by the AV enclave to ensure that is communicating with a real SGX-enabled machine and the user is authorized to access the materials on the specified platform.

Remote attestation requires establishing a secure communication session between the client and server to securely exchange the secrets. The secure session establishment is done via traditional public-key cryptography (analogous to how the SSL handshake includes both authentication and session establishment).

After a successful remote attestation, the server provides access to encrypted digital objects and provisions the keys to the AV enclave. The keys would be securely stored and only accessible by the AV enclave and emulator's enclave using the SGX sealing mechanism (e.g., sealing by signing identity). Since the keys are securely stored, there is no need to repeat the attestation and secret provisioning steps every time the emulator is restarted on the platform. However, if one wants to insert policies on use, it is certainly possible to change the architecture so that there could be provisioning from the provider server on each initiation, or on a regular time period.

4.3.2 Secure Remote File Access. After a successful remote attestation, the server provides secure access to the encrypted files via mounting of a secure remote file system on the local platform (see Arrow (3) in Figure 6). For example, in our implementation, the trusted emulator uses SSHFS (SSH Filesystem) that is a filesystem to mount and interact with files located on a remote server via the SSH file transfer protocol (SFTP), a network protocol providing secure file transfer and a secure remote file system. SSHFS authenticates and encrypts connections. Thus, only those who should have access to remote directories can mount them. Files are encrypted inside such secure remote file system, and only the enclaves (the AV and emulator's enclave) have access to the keys and can decrypt the data.

5 IMPLEMENTATION AND RESULTS

We implemented a prototype of the system on a platform with Skylake processor and SGX-enabled BIOS support. We

started with an open-source GameBoy emulator, called Gearboy [9], and added the trusted component and the AV module to it.

Designing the trusted GameBoy using SGX requires considering fundamental decisions that affect both (1) the security properties of the system, such as the TCB size and the exposed interface to the enclave's outside world, and (2) the performance, mostly due to the restrictions of SGX such as unsupported system calls. Therefore, the most important step in developing the system is refactoring the emulator into the trusted and untrusted components and designing the trusted interfaces to connect them. Besides the system design trades-off, refactoring process can also be challenging because of strong dependency between the core and untrusted modules.

As Figure 5 demonstrates, the trusted core includes the cartridge, processor, registers, memory, and video/audio controllers. In our work, we assume no secrets are displayed to the user. However, since the Video/Audio drivers have direct access to the emulated processor, memory and registers, we added the Video/Audio controllers and the main frame-buffers to the trusted core to decrease security risks against the core and improve performance by decreasing the interaction between the core and the untrusted world. Having minimal interfaces is also important for step-by-step debugging the system and ensure its correctness. The untrusted component includes the graphical user interface, IO modules, and SDL library which are not hosted inside the enclave.

The untrusted part is responsible for creating enclave pages using the ECREATE instruction, loading the trusted code and data into the enclave using the EADD instruction, updating the enclave's measurement for the attestation phase using the EEXTEND instruction, and initializing the enclave using the EINIT instruction. Then a process can execute the enclave's code using the EENTER instruction and leave it using the EEXIT instruction. Because SGX does not support many legacy (frequently insecure) standard C library functions, system-calls, and some of the user-mode instructions [13, 21], one of the main challenges in our work was re-implementing a large part of the core to use the Intel's provided trusted libraries or providing trusted interfaces for accessing the unsupported system-calls or instructions. For Example, SGX does not support IO operations inside an enclave; so we developed a trusted interface for the basic file operations used by the core. Following the same approach, we provided trusted interfaces for accessing SDL library and Qt framework [41] (that are not trusted) from the core enclave. The emulator depends on SDL and Qt for providing the GUI, displaying video/audio, and keyboard/event handling.

To ensure no secret from the trusted core can leak out to the untrusted part, the trusted interfaces are designed in a manner that requires carefully validating all the inputs and outputs of the trusted interfaces within the enclave. To detect unauthorized input from outside into the enclave, we check the correctness of the input based on its type. As a simple example, in the case of numerical input, we can verify its range, length boundaries, and value before it is processed by the trusted code. A similar approach is considered for validating outputs from enclave to the untrusted domain.

The emulated cartridge loads encrypted ROMs from the secure remote file system through the trusted IO interface. The security module has access to the keys and is responsible for decrypting ROMs after being loaded by the ROM loader and encrypting them before writing to the local file cache. Therefore ROMs would be unencrypted only inside the enclave and encrypted before storing outside the enclave. The security module uses Intel's trusted OpenSSL library (topenssl) for cryptographic operations inside the enclave.

Refactoring of the emulator to fit with the trusted/untrusted partitioning and provide the needed SGX support increased the lines of code (LOC) by $\approx 15\%$. Less than 10k LOC were added to enclaves, so while an enclave's trusted storage is limited (128MB at most) it is more than enough for many similar emulators.

For performance evaluation, we first compare running time of the original emulator with the trusted emulator, both executing the same (not encrypted) ROM file, without enabling the on-the-fly encryption/decryption support for the trusted emulator. This evaluation shows the running time of our trusted emulator increased by $8.6X$. In the second evaluation, we compared the two emulators, but this time we enabled the trusted emulator on-the-fly encryption/decryption support. In this case, the trusted emulator performance decreased by $9.4X$. Our analysis shows that the main performance cost is because of the IO and cryptography operations from inside enclaves during the startup and shutdown phases.

Finally, we developed the client AV module using the Intel Elliptical Curve Diffie-Hellman (ECDH) key exchange library to establish a trusted channel between the two enclaves (the security module is responsible for the local attestation) and attest the trusted emulator locally. For remote attestation, it uses the Intel's key exchange and remote attestation library.

6 CONCLUSION

This paper introduces a secure, trustworthy and portable emulation architecture for digital preservation to provide a secure remote access to restricted born-digital materials while protecting the confidentiality and integrity the sensitive data. Our solution is built on top of Intel SGX which is the state-of-the-art technology in trustworthy computing.

We developed a trusted GameBoy emulator as a proof of concept. However, our solution can be used to build more

general-purpose trustworthy digital preservation platforms with reasonable performance overheads. For example, we have been working on developing a trusted classic MacOS emulator by modifying Basilisk emulator (a popular classic MacOS emulator). The trusted Basilisk is designed to protect confidentiality and integrity of restricted disk images while processing them on an untrusted platform. Similar to the trusted GameBoy architecture, our trusted Basilisk architecture has a trusted core and an untrusted component which are connected by narrow trusted interfaces. The trusted Basilisk core, which works inside an enclave, contains the 68k CPU engine (including the interpreter, CPU/FPU emulator, and memory management), the ROM and resource patches, and some of the drivers (including the VHD disk, CD-ROM, and video drivers). The untrusted component includes the remained modules. As an example, Emory University could use the trusted Basilisk to provide remote access to the processed portions of Salman Rushdie's digital archive[16], while protecting the digital archive from unauthorized copying. To protect confidentiality and integrity of the restricted disk images, the trusted Basilisk core loads encrypted disk images from a secure file system and decrypt them inside the emulator core enclave and encrypts the data while writing the modified images back to a local cache.

An institution can use such trustworthy emulation platforms to provide a secure remote access to restricted born-digital content while protecting the confidentiality and integrity of the sensitive data while being processed on an untrusted platform.

Our solution can also be used for providing secure remote access to born-digital documents that do not depend on emulation platforms. For example, a service provider can use a similar architecture to develop a trusted PDF viewer to protect confidentiality and integrity of sensitive documents while being processed on an untrusted platform. Basically, PDF is a container that includes information about how to layout and organize contents to display. The core functionality of a trusted PDF viewer that needs to be protected inside and enclave is the process of rendering (encoding image information) the contents of a PDF file into a display format; Usually, PDF viewers depend on PDF rendering libraries or SDKs to render PDF contents. The PDF viewer needs to provide trusted interfaces to access such untrusted libraries from the enclave.

ACKNOWLEDGMENT

This material is based in part upon work supported by the National Science Foundation under Grant Numbers IIS-1529415 and CSN-1111149. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] ARM Limited. 2009. ARM Security Technology: Building a Secure System using TrustZone Technology. (2009).

[2] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 8.

[3] bbcdeclassify 2005. Readers 'declassify' US document. (2005). http://news.bbc.co.uk/1/hi/world/europe/4506517.stm

[4] Winfried Bergmeyer. 2011. The Keep Emulation Framework.. In *SDA*. Citeseer, 8–22.

[5] Xiaoxin Chen, Tal Garfinkel, E Christopher Lewis, Pratap Subrahmanyam, Carl A Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan RK Ports. 2008. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *ACM SIGARCH Computer Architecture News*, Vol. 36. ACM, 2–13.

[6] John Criswell, Nathan Dautenhahn, and Vikram Adve. 2014. Virtual Ghost: Protecting Applications from Hostile Operating Systems. *SIGPLAN Not.* 49, 4 (Feb. 2014), 81–96. DOI: http://dx.doi.org/10.1145/2644865.2541986

[7] dosemu 2006. DOSEMU. (2006). http://www.dosemu.org.

[8] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. 2003. Terra: A virtual machine-based platform for trusted computing. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM, 193–206.

[9] Ignacio Sanchez Gines. 2012. Gearboy - Nintendo Game Boy Emulator. https://github.com/drhelius/Gearboy. (2012).

[10] Stewart Granger. 2000. Emulation as a Digital Preservation Strategy. *D-Lib Magazine* 6, 10 (October 2000).

[11] Mark Guttenbrunner, Christoph Becker, Andreas Rauber, and Carmen Kehrberg. 2008. Evaluating strategies for the preservation of console video s. *Proceedings of the Fifth Annual Conference on Preservation of Digital Objects* (2008).

[12] Owen S. Hofmann, Sangman Kim, Alan M. Dunn, Michael Z. Lee, and Emmett Witchel. 2013. InkTag: Secure Applications on an Untrusted Operating System. *SIGPLAN Not.* 48, 4 (March 2013), 265–278. DOI: http://dx.doi.org/10.1145/2499368.2451146

[13] INTELCORP. 2016. Intel Software Guard Extensions (Intel SGX) SDK. (2016). https://software.intel.com/sgx-sdk

[14] Simon P Johnson Vincent R Scarlata Ittai Anati, Shay Gueron. 2013. Innovative Technology for CPU Based Attestation and Sealing. https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing

[15] Simon Johnson. 2013. Attestation and Sealing with Software Guard Extensions. https://software.intel.com/en-us/attestation-sealing-with-software-guard-extensions

[16] Emory Libaries. 2014. (2014). Retrieved December, 2014 from http://marbl.library.emory.edu/using/reading-room/digital-archives-access.html

[17] Paul Robson Martin Korth Marat Fayzullin, Pascal Felber. 1998. Everything You Always Wanted To Know About GAMEBOY. (1998). http://www.emulatronia.com/doctec/consolas/gameboy/gameboy.txt

[18] Lorenzo Martignoni, Pongsin Poosankam, Matei Zaharia, Jun Han, Stephen McCamant, Dawn Song, Vern Paxson, Adrian Perrig, Scott Shenker, and Ion Stoica. 2012. Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. USENIX Association, Berkeley, CA, USA, 14–14. http://dl.acm.org/citation.cfm?id=2342821.2342835

[19] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. 2010. TrustVisor: Efficient TCB Reduction and Attestation. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP '10)*. IEEE Computer Society, Washington, DC, USA, 143–158. DOI: http://dx.doi.org/10.1109/SP.2010.17

[20] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. 2008. Flicker: An Execution Infrastructure for Tcb Minimization. *SIGOPS Oper. Syst. Rev.* 42, 4 (April 2008), 315–328. DOI: http://dx.doi.org/10.1145/1357010.1352625

[21] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '13)*. ACM, New York, NY, USA, Article 10, 1 pages. DOI: http://dx.doi.org/10.1145/2487726.2488368

[22] Pil Mellor. 2003. CaMiLEON: Emulation and BBC Doomsday. *RLG DigiNews* 7, 2 (2003).

[23] Dan Pinchbeck, David Anderson, Janet Delve, Getaneh Alemu, Antonio Ciuffreda, and Andreas Lange. 2009. Emulation as a strategy for the preservation of games: the KEEP project. *DiGRA 2009-Breaking New Ground: Innovation in Games, Play, Practice and Theory* (2009).

[24] Donald E Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C Hunt. 2011. Rethinking the library OS from the top down. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 291–304.

[25] Dan R. K. Ports and Tal Garfinkel. 2008. Towards Application Security on Untrusted Operating Systems. In *Proceedings of the 3rd Conference on Hot Topics in Security (HOTSEC'08)*. USENIX Association, Berkeley, CA, USA, Article 1, 7 pages. http://dl.acm.org/citation.cfm?id=1496671.1496672

[26] Himanshu Raj, David Robinson, Talha Bin Tariq, Paul Engl, Stefan Saroiu, and Alec Wolman. 2011. *Credo: Trusted Computing for Guest VMs with a Commodity Hypervisor.* Technical Report.

[27] Klaus Rechert, Isgandar Valizada, Dirk von Suchodoletz, and Johann Latocha. 2012. bwFLA–a functional approach to digital preservation. *PIK-Praxis der Informationsverarbeitung und Kommunikation* 35, 4 (2012), 259.

[28] Jeff Rothenberg. 1999. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation.* Council on Library & Information Resources.

[29] Jeff Rothenberg. 2000. *An Experiment in Using Emulation to Preserve Digital Publications.* Technical Report. Koninklijke Bibliotheek.

[30] Jeff Rothenberg. 2000. *Using Emulation to Preserve Digital Documents.* Technical Report. Koninklijke Bibliotheek.

[31] seeingdouble 2004. Seeing Double Emulation Theory and Practice. (2004). http://www.variablemedia.net/e/seeingdouble/home.html

[32] Sheepshaver. 2010. The Official SheepShaver Home Page. (2010). http://sheepshaver.cebix.net. Accessed December 2014.

[33] Emin Gün Sirer, Willem de Bruijn, Patrick Reynolds, Alan Shieh, Kevin Walsh, Dan Williams, and Fred B. Schneider. 2011. Logical Attestation: An Authorization Architecture for Trustworthy Computing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 249–264. DOI: http://dx.doi.org/10.1145/2043556.2043580

[34] Richard Ta-Min, Lionel Litty, and David Lie. 2006. Splitting interfaces: Making trust between applications and operating systems configurable. In *Proceedings of the 7th symposium on Operating systems design and implementation.* USENIX Association, 279–292.

[35] Isgandar Valizada, Klaus Rechert, and Dirk von Suchodoletz. 2014. Emulation-as-a-Service–Requirements and Design of Scalable Emulation Services for Digital Preservation. *Hochleistungsrechnen in Baden-Wuerttemberg-Ausgewaehlte Aktivitaeten im bwGRiD 2012: Beitraege zu Anwenderprojekten und Infrastruktur im bwGRiD im Jahr 2012* (2014), 103.

[36] Jeffrey van der Hoeven. 2007. Dioscuri: emulator for digital preservation. *D-Lib Magazine* 13, 11/12 (November/December 2007).

[37] Jeffrey van der Hoeven and Hilde van Wijngaarden. 2005. Modular emulation as a long-term preservation strategy for digital objects. *Proceedings of the International Web Archiving Workshop* (2005).

[38] Dirk Von Suchodoletz and Jeffrey Van der Hoeven. 2009. Emulation: From digital artefact to remotely rendered environments. *International Journal of Digital Curation* 4, 3 (2009), 146–155.

[39] Common Vulnerabilities and Exposures. 2006. (2006). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2369

[40] Wikipedia. 2017. Game Boy — Wikipedia, The Free Encyclopedia. (2017). https://en.wikipedia.org/w/index.php?title=Game_Boy&oldid=790727508

[41] Wikipedia. 2017. Qt (software) — Wikipedia, The Free Encyclopedia. (2017). https://en.wikipedia.org/w/index.php?title=Qt_(software)&oldid=790788169

[42] Wikipedia. 2017. Simple DirectMedia Layer — Wikipedia, The Free Encyclopedia. (2017). https://en.wikipedia.org/w/index.php?title=Simple_DirectMedia_Layer&oldid=788274219

[43] Kam Woods and Geoffrey Brown. 2009. Assisted Emulation for Legacy Executables. In *to appear in 5th International Digital Curation Conference.*

[44] Xpdf 2006. Xpdf. (2006). http://www.foolabs.com/xpdf/. Accessed September, 2006.

[45] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on.* IEEE, 640–656.

[46] Jisoo Yang and Kang G Shin. 2008. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments.* ACM, 71–80.

[47] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. 2011. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles.* ACM, 203–216.