Using Innovative Instructions to Create Trustworthy Software Solutions

Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Carlos Rozas, Vinay Phegade, Juan del Cuvillo
Intel Corporation

{matthew.hoekstra, reshma.lal, pradeep.m.pappachan, carlos.v.rozas, vinay.phegade, juan.b.del.cuvillo}@intel.com

Abstract

Software developers face a number of challenges when creating applications that attempt to keep important data confidential. Even diligent use of correct software design and implementation practices, can allow secrets to be exposed through a single flaw in any of the privileged code on the platform, code which may have been written by thousands of developers from hundreds of organizations throughout the world. Intel is developing innovative security technology that allows software developers control of the security of sensitive code and data by creating trusted domains within applications to protect critical information during execution and at rest. This paper will show how protection of private information, including enterprise rights management, video chat, trusted financial transactions, among others, has been demonstrated using this technology. Examples will include both protection of local processing and the establishment of secure communication with cloud services. It will illustrate useful software design patterns that can be followed to create many additional types of trusted software solutions.

1 Introduction

1.1 Secure Software Challenges

Digital devices are integral to the lives of millions of people today. Applications on these devices are used for everything from sharing pictures with family and friends, to working with top secret enterprise intellectual property, with hundreds of new applications and cloud services becoming available every day.

As a result, applications are responsible for protecting increasing amounts of sensitive information including financial account data, passwords, personal calendars, medical information, as well as confidential enterprise data, etc.. Recognizing their responsibility to ensure that private data is only used as intended, application developers are becoming increasingly security conscious. Secure development practices are standardized in many organizations and security testing is often a key part of software validation cycles. But even so, protection of critical data depends upon the correctness of a significant amount of software on the platform which has been written by others and which grants privileges to look into an application's data space. Application developers must also trust the users of client platforms and system administrators of cloud service platforms to follow security best practices that keep application-managed data safe from malicious software.

Anti-virus products play a critical role in limiting opportunities for the introduction of malware, but history shows that users will continue to browse, download interesting applications, and fail to follow security best practices when these practices become inconvenient. As a result, users fall victim to bad actors and put not only their personal information at risk, but potentially critical enterprise intellectual property as well.

There is a clear need for technological solutions that will help software developers ensure that even naïve users, with little understanding of digital security threats, can safely manage their personal, financial, and enterprise IP without encumbering their user experience, or limiting their personal control. This need is exhibited across hundreds of applications that manage important information, with more created every day.

1.2 Overview

The paper Innovative Instructions and Software Model for Isolated Execution [1] provides a description of Intel® Software Guard Extensions (Intel® SGX), a new set of CPU instructions that give application and service providers a safe place to stand when managing the use of the data they consume and collect. Sensitive data is protected within applications even when the platforms on which they run are infected with more privileged malicious software or if the platform falls into the physical control of a person wishing to gain unauthorized access to the data.

The remainder of this paper provides a review of the programming model for SGX. It describes the design steps taken by developers wishing to take advantage of these instructions, and then reviews three example secure solutions that have been developed to take advantage of these new instructions.

2 SGX Programming Model

In this review of SGX and its programming model, we use the following terminology:

- **Enclave** – isolated region of code and data within an application's address space. Only code executing within the enclave can access data within the same enclave.
- **Measurement** – a cryptographic hash of the code and data in an enclave at the time it is initialized.
- **Attestation** – the mechanism by which an enclave on one platform proves to a remote entity, that it was instantiated correctly.

Using SGX, an application can start the enclave creation process by executing the following sequence of instructions:

- ECREATE – Allocates a region of virtual memory within the application for hosting the secure code and data.
- EADD - Critical code and data pages are added to the enclave using EADD.
- EEXTEND – Updates the **measurement** of the enclave to include the code or data added in EADD.

- EINIT – Locks down the contents of the enclave and ensures that only the code within the enclave has access to the data regions in the same enclave.

Once the enclave has been created and initialized with EINIT, attempted accesses to the enclave's memory from unauthorized software, even software such as virtual machine monitors, BIOS, or operating system functions operating at a higher privilege level, are prevented.

From a physical point of view, while enclave data is resident within registers, caches, or other logic blocks inside the processor package, unauthorized access via software is prevented by CPU logic. Whenever enclave data leaves the on-package caches to be written to platform memory, the data is automatically encrypted and integrity protected. This prevents malicious individuals from using memory probes or other techniques to view, modify, or replay data or code contained within the enclave.

As each page of data is loaded into the enclave, using the EADD and EEXTEND instructions, internal data structures within the CPU are updated to include cryptographic **measurements** of the code and data added to the enclave. The ability of a remote party to verify that enclave **measurements** and platform settings are configured as expected is referred to as attestation. SGX uses two related credentials for **attestation**: reports and quotes. A report is used to verify the correctness of enclaves on the local platform, and a quote can be used to reflect platform and enclave state to entities outside of the platform. Once the **attestation** has been successfully completed, a trusted channel can be established between a server and the enclave enabling secrets to be securely transmitted. More details regarding **attestation** are discussed in [2].

3 Example Applications

Using the software model described in the overview, a number of interesting trustworthy applications can be created through the use of the SGX technology. Each of the applications described in the remainder of this document have been successfully built and executed on a prototype hardware implementation of the SGX technology. Design and prototyping of the Enterprise Rights Management (ERM) and Secure Video Conferencing (SVC) applications in particular were sponsored by the United States Department of Homeland Security and the United States Air Force Academy.[i]

3.1 One-time Password (OTP)

A simple application that can be created with SGX is a generator of one-time passwords.

3.1.1 Overview

OTP is an authentication technology often used as a second factor to authenticate a user. As suggested by the name, the password is valid only for one authentication and is often used to authorize online financial transactions.

One example of a one-time password solution is RSA SecurID®. With this solution, a pre-shared key is established between a server component and a hardware token, the RSA SecurID® Hardware Authenticator, which can be in the form of a keychain or a credit card. Periodically an algorithm is used to cryptographically combine the pre-shared key and the time to produce a code which becomes the one-time password.

When logging into a service that uses a one-time password, in addition to the traditional username and password, a user will also enter the code that is displayed on the token (computed from the pre-shared key and the time). In more advanced services the user may need to use a keypad on the token to enter a value generated by the server and displayed on the login page (challenge-response) to prevent phishing attacks. When verifying the user's identity, the server component can also generate a code given the pre-shared key, the time, and an optional challenge code.

3.1.2 Design Goals

For such a one-time password solution to work, hardware tokens must be created and distributed to users, resulting in increased cost for organizations that wish to deploy such a solution. Software versions of one-time password solutions exist. However, depending upon the value of the service or transaction being protected, the solution may be too vulnerable to malware targeted at the solution.

It would be desirable then to create a software-based one-time password solution which utilizes the SGX technology to prevent attacks from malware. A prototype was developed to evaluate such a solution.

3.1.3 Secure One-time Password Architecture

Like the hardware token based solution, there are two primary components to the architecture: the OTP server and the OTP client. In the case of the prototype developed for this work, the OTP client side component was implemented as a browser plug-in, but it could be instantiated in a number of different ways as well, including a stand-alone application or a background service. Within the OTP client software, the algorithms that interact directly with the OTP secrets were placed in an enclave. Figure 1 shows the overall architecture.

An example set of steps for installing the OTP plugin on the client device and generating a shared key is described below:

1. The user requests a new account at a financial institution (FI).
2. The FI asks the user to install a new browser plugin.
3. The user chooses to install the plugin.
4. The plugin launches and instantiates the OTP enclave.
5. The plugin contacts the FI server and asks for a new pre-shared key.
6. The plugin authenticates the FI server using a protocol such as TLS.
7. The FI server ensures that it is communicating with a valid OTP enclave using mechanisms described in [2]. It can then establish a trusted channel that terminates in the enclave.
8. To prove that a new account is being established for a valid user, the user can be provided an authentication code through an out-of-band channel such as a text message or phone call.

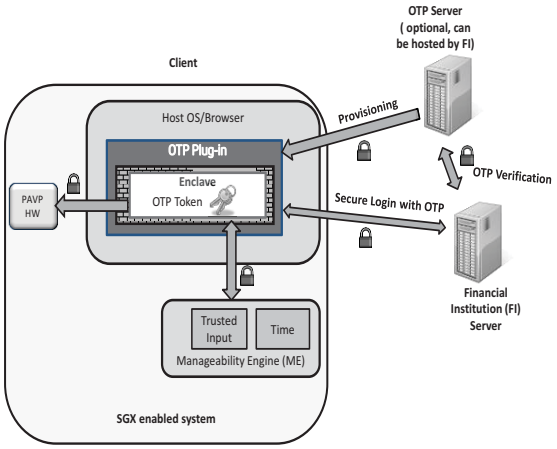

Figure 1: OTP within an Enclave.

9. The user is prompted for the authentication code, which he enters into the OTP plugin, which passes the code to the enclave, which in turn passes the authentication code over the trusted channel to the OTP server.
10. The OTP server has now verified that it is communicating with a correct version of the OTP enclave, and that an authorized user has requested a new account.
11. The OTP server can now generate a random OTP pre-shared key. It will store this key locally in a secure fashion, and associate the key with the user's account.
12. This pre-shared key can now be sent to the OTP enclave via the trusted channel that was established during the attestation process. The OTP server and the OTP enclave now possess the same pre-shared key.
13. The OTP enclave can now store the pre-shared key using the sealing process described in [2]. As described in this paper, the OTP pre-shared key can be encrypted and stored with a key that is only known to the OTP enclave executing on the same platform that initiated the OTP provisioning process.
14. At this point both the OTP server and the OTP client enclave in the OTP plugin can securely access the same pre-shared OTP key.

Once the pre-shared key has been established between the OTP enclave and the OTP server, secure solutions can now use the SGX enabled OTP as a second authentication factor following the set of steps below:

1. A user begins the login process with her financial institution using her browser with the installed OTP plugin.
2. The web page received by the browser contains an OTP element which signals the OTP plugin to become active.
3. The OTP enclave and OTP server establish a secure channel as described in the provisioning step previously.
4. The OTP server generates a random challenge value and sends this to the OTP enclave over the secure channel.
5. The OTP enclave generates the OTP value by cryptographically combining the challenge, the current time, and the pre-shard key to produce the one-time password.
6. As the user completes the login procedure, the browser plugin adds the computed one-time password to the set of data that is sent to the FI.
7. The FI can now validate that the correct one-time password was used during the login process.

3.1.4 Results

The OTP prototype described here prevents malicious software from gaining access to the OTP pre-shared key, including targeted attacks from malicious software with higher privilege. This results in higher confidence that the authenticating service is indeed communicating with a platform that was provisioned by an authorized user.

To better ensure that the web service is indeed communicating with a human user, additional techniques could be used. Instead of sending the challenge directly to the OTP enclave, the remote service could generate a bitmap containing a visual representation of the challenge code, and encrypt this before sending to the OTP enclave. The combination of the OTP enclave and the OTP plugin could use Intel's Protected Audio Video Path (PAVP) technology [3] to securely render the bitmap to the screen in a way that cannot be observed by software. The user would then enter the challenge code when prompted by the OTP plugin, thereby ensuring that a person is requesting the transaction from an authorized system.

To harden the solution even further, a trusted input path could be established between an input device and Intel's Manageability Engine (ME), part of the Intel® Active Management Technology [4]. The ME could then establish a secure channel to the OTP enclave, ensuring that no software could observe the input entry. This solution could use a PIN known only to the user, and cryptographically combined with the pre-shared key, time, and challenge to compute the one-time password.

3.2 Enterprise Rights Management

3.2.1 Overview

Enterprise Rights Management (ERM) is a technology that aims to secure crucial elements of access and distribution of sensitive documents, such as confidentiality, access control, usage policies, and logging of user activities. While most existing solutions focus on the protection of enterprise data, the need to enforce the authorized use and dissemination of personal content such as pictures and videos is becoming increasingly apparent. The same technologies could be used for this purpose as well.

ERM protected applications typically run on off-the-shelf client platforms and operating systems. Malware, including viruses and rootkits, could compromise the ability of an ERM application to protect its secrets and enforce its policies, potentially resulting in the transparent loss of digital assets which may remain undetected for a significant amount of time. One example of such an attack is operation Aurora which affected many large corporations [5]. Current solutions, which attempt to protect ERM assets using encryption and access control mechanisms, are vulnerable to several attacks. For example, malware might steal document content and/or encryption keys from application memory where it is processed; copy display content from video frame buffers; or violate use policy (e.g., alter time on the client machine to extend an expired document lease). In a more advanced attack, if an attacker has physical possession of a platform, he may be able to use memory snooping or cold boot style attacks [6] to acquire the keying material for a valid ERM solution. This would permit the attacker to create malware which could use those stolen keys to effectively impersonate a valid ERM client. Finally, an authorized consumer of enterprise data may, in rare circumstances, wish to maliciously copy large amounts of sensitive digital information, and could directly modify the ERM solution to avoid logging and other forms of detection. In the following sections, we describe an SGX technology-based ERM architecture, focused on document distribution, access control and viewing, that addresses these critical vulnerabilities found in today's systems.

3.2.2 Design Goals

The key design goals of our ERM architecture were to protect the system against the following threats:

1. Document content and encryption key theft.
2. Platform and application identity spoofing.
3. Use policy and activity log tampering.

We focus on protecting the client in the ERM system, since client applications tend to run on platforms that may not have the same degree of control and security as enterprise servers. However, the techniques described here can also be extended to protecting resources on ERM servers.

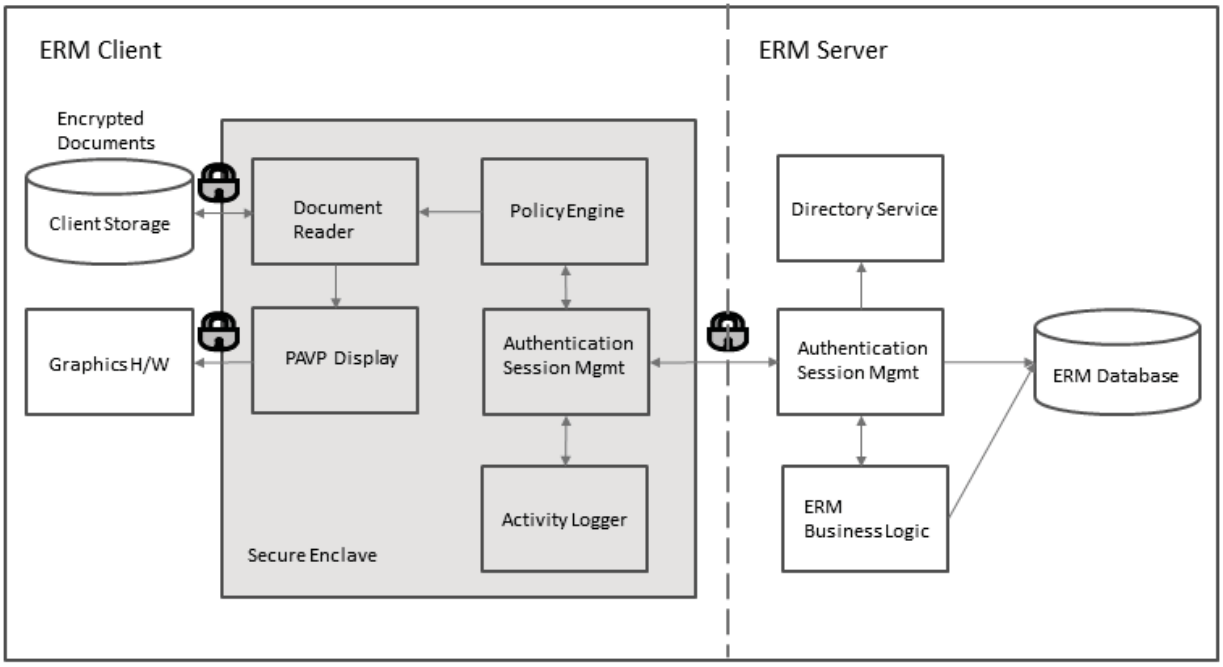

Figure 2: Key Components of Client-Server ERM Architecture

3.2.3 Secure ERM Architecture

In this section, we describe the key components of our client-server ERM architecture (see Figure 2). The trusted parts of the client, which are hosted inside of an SGX protected enclave, are responsible for operating on the assets of the application that need protection. In the discussion that follows, we describe the key components of our architecture and how they operate to provide the level of security needed for various use cases in the ERM domain.

The client authentication and session management module authenticates the client platform and user to its counterpart in the ERM server. Using SGX attestation capabilities, this module generates a verifiable report of the client's identity that is bound to the platform by the CPU. The report also includes information about the user running the ERM session. The server verifies the report to ensure that it is communicating with an SGX-enabled machine and validates (using a directory service such as Active Directory) that the user is part of an organizational domain that is authorized to access the ERM system on the specified platform. The client and server engage in a one-time provisioning protocol that results in application secrets being securely sealed to the client platform, using SGX's sealing capabilities. These secrets, which can only be unsealed by the application that sealed it, are used to establish secure sessions with the server in the future, without the need for constantly proving the identity of the client platform.

In our architecture, document use policy and encryption keys are stored in the ERM database on the server. The document owner specifies use policy and access control using the policy engine that runs inside the enclave.

The policy is then uploaded to the ERM database through the secure communication channel between the client and server. The protected document is encrypted within the enclave with a randomly generated key which is stored on the server, and later distributed to authorized Intel® SGX protected document viewers. The encrypted documents themselves need not be stored in the server database and can be disseminated to intended recipients by various means (document repositories, email etc.).

An authorized user, upon receipt of an encrypted document, can view it using the secure document reader component of the client application, running inside an enclave. The policy engine, after validating that the use policy (downloaded securely from the server into the enclave) of the document is compatible with the user operation (e.g., viewing), also gets the document decryption key and transfers control to the document reader component. The reader decrypts the document inside the enclave, parses the content and generates page bitmaps for rendering on the display device. Since the path between application memory and the video frame buffer that holds the bitmap before rendering is insecure, we use PAVP technology to encrypt bitmaps using a symmetric key shared with the GPU on the platform. The encrypted bitmaps are transferred to the graphics hardware for rendering via standard graphics drivers.
Finally, the secure activity logger, also running inside the enclave, records every user activity related to document viewing and transmits it to the server where it is stored. This capability enables features such as auditing of document access and non-repudiation of user actions.

The untrusted part of the application, consisting of the GUI and libraries used to avail of kernel services (e.g., file I/O, thread management etc.), interface with the trusted part through well-defined entry-points. The interface was designed to ensure that no secrets from the trusted part are allowed to leak out to the untrusted part, and the hardware/software protections of SGX ensure that the secrecy and integrity of the data and code inside the enclave-resident trusted part is maintained at all times.

The ERM server, which we assume to be secure in this work, consists of the authentication and session management module, a directory service for maintaining user and platform information, and the database that stores information about whitelisted client platforms; client-server session state; document policies and keys; and user activity logs. All communication between the server and authenticated clients is encrypted and also offers integrity and replay protection to provide end-to-end security for various use cases.

3.2.4 Implementation and Validation

We implemented a prototype ERM solution, starting with an open source software library (MuPDF) for document viewing and rendering, and adding modules for provisioning, authentication and session management, policy enforcement, activity logging and secure display for the client; the ERM server was completely implemented from scratch. We implemented several attack scenarios such as key stealing, policy modification, video frame buffer scraping, and spoofing of application identity to attack the provisioning protocol. Our experiments showed how SGX technology enabled the application to withstand these attacks and satisfy all its security objectives. The prototype successfully passed two rounds of penetration testing by security experts in the enterprise and government sectors.

We learned several lessons from our experience in building the ERM application using SGX technology. Since we started with existing open source software in our implementation, we gained insight into performing a security analysis of the application's assets. This led to a refactoring of the application to fit the untrusted/trusted partition that was required for SGX enabling. Our work influenced the validation and enhancement of the libraries and tools that facilitate SGX application development. By carefully designing various modules, we were able to develop several reusable security modules (provisioning, authentication and session management) that were subsequently used by other projects.

In Figure 3 below, we show a high level view of the Secure Video Chat application which contains two SVC clients and a server that are protected using SGX technology. The figure illustrates the trusted components that are hosted inside the enclave and the trusted channel between the SVC enclave and I/O devices (e.g. camera, microphone, speaker and display).

3.3 Secure Video Conferencing

We will now examine how the security of a video conferencing application can be hardened using SGX technology. With the widespread availability of high network bandwidth and inexpensive hardware for capturing video and audio on client platforms, use of video chat, video conferencing and web conferencing applications has become increasingly popular for real time information sharing. This creates an opportunity for the unauthorized capture and distribution of a video conferencing stream by malicious individuals, or theft of valuable IP or sensitive information in enterprise and government sectors. An unauthorized entity or malware can intercept and steal the Audio/Video (AV) stream during a video conferencing session. Today's secure video conferencing solutions provide strong protection of sensitive content on the network through the use of cryptographic methods. But with the migration of threats from the network onto the computing platform, this level of security is no longer sufficient to protect the AV stream as it is being processed on the computing device. SGX allows a video conferencing application to protect its assets on the platform and enables strong participant authentication, thus mitigating a broad range of threats that could compromise the secrecy and integrity of the AV stream. Further, if the platforms contain input and output (I/O) devices that present a strong identity and have processing capability, a secure channel can be created between the VoIP enclave and the I/O device. Thus we can protect the AV input and output against an attack that compromises the integrity of the I/O stack in order to steal the input from the camera and microphone or the output to speaker and display devices.

3.3.1 Threat Model for Video Conferencing

To understand the protection of a video conferencing application using SGX, let us consider a two person Secure Video Chat (SVC) application. The users first establish a call session using a call initiation protocol such as Session Initiation Protocol (SIP), after which the locally captured AV streams are transmitted using a transport protocol such as Real Time Transport Protocol (RTP). In a secure video chat, participant authentication occurs prior to session setup with the help of a SVC server that checks the identity of the participants. The AV streams are encrypted for secure transmission using a protocol such as Secure Real Time Transport Protocol (SRTP). The limitation of such a secure video chat is that it is vulnerable to attacks such as the following:

1. Keys used for SRTP encryption can be stolen from the application's memory during processing.
2. The cleartext AV stream can be stolen from the application's memory during media stream processing or SRTP decryption.
3. User identities can be spoofed, compromising participant authentication.
4. Any policy, such as logging the call events, or not allowing recording, etc., can be violated through modification of application code that enforces such policies.

3.3.2 Secure Video Conferencing Architecture

In our lab experiment, we modified an open source video conferencing application to use SGX and created a significantly hardened video chat application. To develop the security architecture, we first identified the adversaries and the threat model for the video chat usage, and created a list of assets that needed protection from the malware. We then analyzed an existing video conferencing stack, and studied its operation to understand the flow of control and secure assets. This enabled us to modify the design of the stack with well-defined trusted and untrusted components. The trusted components contained code and data requiring confidentiality and/or integrity protection such as SRTP keys, media streams, cryptographic operations, policy enforcement logic etc. The untrusted components contained code and data for interfacing with operating system services and drivers. Since the OS components only handled content that was encrypted, they were not required to be in the trusted part of the application. The trusted components could be protected inside one or more enclaves, though in our implementation we used a single enclave to host all sensitive code and data. Using multiple enclaves offers increased isolation amongst the trusted components, especially if the components are developed by multiple ISVs. The elements inside SVC's trusted partition are:

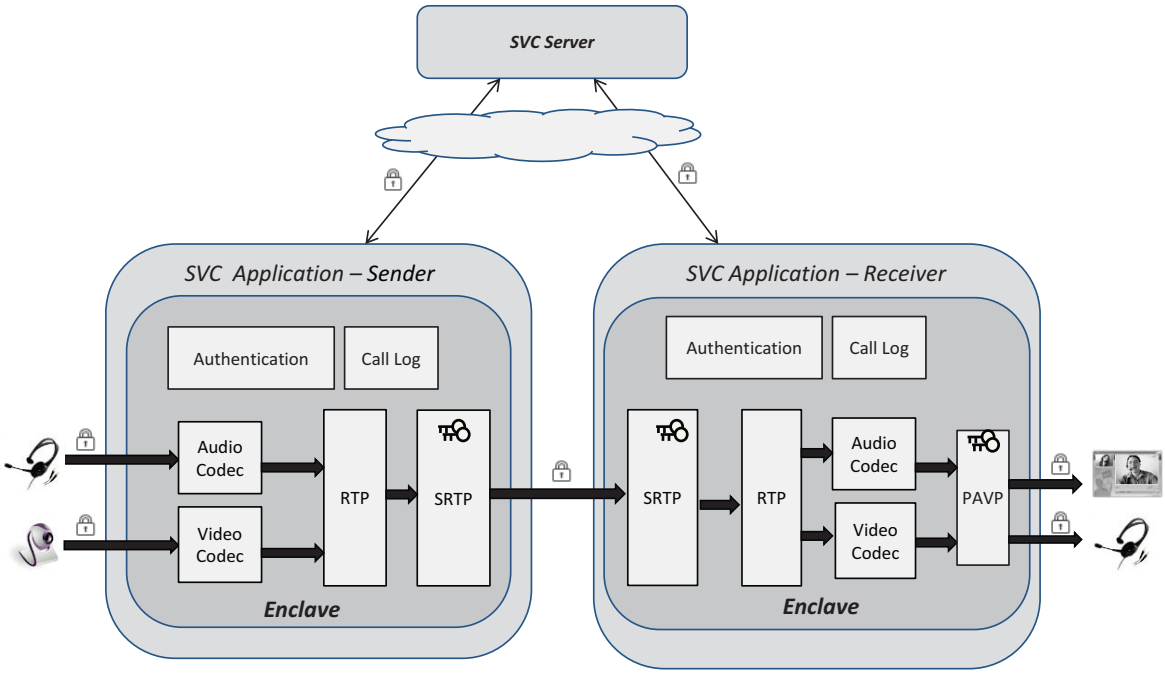

Figure 3: Secure Video Chat Application Structure

1. **Authentication:** Responsible for authenticating the participant to the SVC server, enabling the client to present strong attestation data which is signed using CPU keys. After successful authentication, secrets are provisioned to the SVC enclave and sealed to it for use in subsequent sessions, without requiring lengthy authentication and provisioning each time.
2. **Media Processing:** Performs operations on media such as software encoding/decoding, adaptive de-jittering, lip syncing, and echo cancellation.
3. **SRTP:** Encryption and decryption of AV streams. SRTP keys are protected inside this secure module.
4. **Trusted I/O:** Manages the secure transfer of audio and video streams between the SVC enclave and the AV input and output devices (camera, microphone, speaker, display). This requires the availability of trusted I/O technology for camera, microphone, speaker and display on the computing device. In our experiment, we used Protected Audio and Video Path (PAVP) technology available on Intel platforms to protect audio and video output from the enclave to the graphics device and to the HDMI speakers. Trusted input will be included in future work.
5. **Policy Engine and Call Logging:** The code for enforcing the policy governing the video chat must be hosted inside the enclave to prevent a code modification attacks.

3.3.3 Results

We were successful in taking an open source video conferencing stack and repartitioning it to use SGX and PAVP to create a video chat application that had significantly increased protection over currently available solutions. We conducted several security penetration tests, where the SGX-enabled application successfully withstood all tested high-privilege malware attacks. In our experiment, we protected the audio and video output as well, through integration of SGX and PAVP technologies, which protects against attacks such as video buffer scraping. Our work demonstrated how we could create secure channels between SGX and trusted I/O devices to protect against attacks that target the I/O stack.

The SVC architecture is applicable to VoIP applications such as multi-party video conferencing, phone (audio) calls, Instant Messaging (IM), texting, whiteboarding and web conferencing. Software developers can map our design to their existing implementations for creating SGX-hardened solutions.

4 Future Work

While we have shown how SGX can be used to mitigate many types of attacks attempting to acquire confidential information during runtime and at rest, there are many additional opportunities.

4.1 Hardening Server Processing

Many security applications such as ERM and SVC rely on the security of backend components that include one or more servers. These servers perform, at a minimum, authentication and distribution of initial secrets and could be involved in other secure transactions such as information distribution policy, storage of audit logs, etc. In our current work, we focused on client security for various usages; however, future work is intended to focus on the use of SGX for server security. This includes interaction with various types of virtualization technologies, workload migration between servers, and secure cloud server provisioning techniques.

4.2 Trusted Input

All three of the use cases discussed in this paper would benefit from improved security when transferring data from local input devices to enclaves on the CPU. This includes input from keyboards, touch screens, microphone, cameras, and others. Future work will focus both on how individual input elements can be extended in order to establish secure paths, as well as opportunities to add secure input aggregation points to the platform. This would continue to allow for low-cost input devices while still maintaining control of confidential information processing.

4.3 Improved tools

In creating the prototypes described in this paper, a robust prototype SDK and set of development tools were used to automatically generate the code required to seamlessly transition between traditional untrusted application code, and SGX protected enclaves. Additional enhancements would also be beneficial, including the ability to see visual representations of

data flows into and out of enclaves to help ensure that the appropriate data is being protected, without accidentally passing sensitive information out of the enclaves. Additional programming language extensions may also make the development of secure software easier.

Future work will evaluate how various enhancements to development tools affect the secure software creation process when utilizing SGX.

5 Related Work and Summary

Several other technologies also serve to protect critical code and data. This section reviews these technologies and describes their relation to SGX.

One related set of technologies include managed runtime environments such as Oracle™ Java and Microsoft™ .NET. A managed runtime environment (MRTE) can enforce security policies pertaining to the managed code, and protect the integrity and confidentiality of any code within the managed environment. However, the MRTE must still trust any code with higher privileges to protect its data. An MRTE can be used effectively with SGX, allowing managed code to interact with trusted services implemented in enclaves, or even potentially hosting the entire MRTE itself within an enclave, but the MRTE alone cannot provide the same confidentiality and integrity benefits.

Another software technique for protecting secrets is utilizing tamper resistant software (TRS) [7]. TRS is a set of techniques which serve to make it difficult to understand and to change the logic flow of critical regions of software, and in many cases help the software keep key secrets safe from observation by unauthorized code. TRS techniques are quite complex for developers to use and are constantly subjected to reverse-engineering by motivated attackers. TRS generally also suffers from significant performance degradation. SGX solves many of the problems that necessitated the creation of TRS, while making reverse engineering of the protection technique ineffective and allowing developers to create trusted code modules using familiar tools and processes.

Another scheme employed to create a more trusted environment is the use of microkernels. A microkernel reduces the total amount of code that executes with the highest privilege and unfettered access to all application code and data. This, in principle, can greatly reduce the attack surface. SGX can achieve the same or better reduction in attack surface, without requiring specific system architecture for the operating system, and can be used in conjunction with microkernels as well.

Type 1 Virtual Machine Monitors can be used to create an application hosting environment in an isolated virtual machine (VM). For example, a secure VM could be created for financial transactions, hosting an operating system which is separated from the general purpose OS used for other activities. Isolation from the general purpose operating system is certainly beneficial but there are still significant challenges for an application that wishes to maintain the confidentiality and integrity of its data. The application must still trust the operating system and the VMM, as well as the BIOS etc. This is a significant amount of code that was most likely not written by the application developer.

A challenge to all software approaches is to be certain that privileged software is trustworthy, For example, it is possible to host one trustworthy microkernel or VMM inside of another VM. To solve this problem, Trusted Platform Modules were introduced. When used properly, the TPM can give a reliable measurement of all of the software components in the stack below the application via local or remote attestation. With TPMs, normally the entire state of the platform is measured.

Improving upon the TPM, technologies such as the Intel® Trusted Execution Technology (TXT) have been created to allow the platform state to be reset and measured at a particular instant in time, providing a dynamic root of trust measurement, rather than relying software to take action based on measurements that were recorded at some earlier point. This reset-then-measure process allows a system to have a smaller trusted computing base (TCB).

While TXT can limit the amount of platform infrastructure code that needs to be measured, its typical use still includes a full featured VMM and/or operating system as part of an application's trust boundary. The application developer may need to trust a significant amount of external code.

Recognizing the challenges presented by the need to trust a full operating system, CMU's Flicker [8] loads a simple hosting environment, reported to be as little as 250 lines of code that executed with the primary operating system suspended. With Flicker, an application could execute its untrusted functions in the primary operating system and move its trusted functions to the Flicker environment.

The Flicker approach has a few challenges. Firstly, Flicker code runs with a very high privilege level (that of a VMM), so other applications and the primary OS must intrinsically trust this code. Secondly, in order to support long running trusted code, Flicker infrastructure becomes a new type of scheduler, responsible for scheduling the primary operating system and the trusted applets. Thirdly, Flicker has performance issues related to transitioning into and out of trusted code, in part due to communication with the TPM and also the inability to execute untrusted code on some cores while trusted code is executing on others. Finally, using TXT for Flicker makes it unavailable for its typical use: launching a trusted VMM after the platform has been configured. With SGX, trusted and untrusted functions can execute simultaneously on multiple cores. SGX does not require communication with the TPM, and does not interfere with the use of TXT for existing usages.

Overshadow [9] is another research approach to remove the operating system from the TCB of an application. Overshadow uses a virtualization layer to change the memory access semantics to protect applications running in a primary operating system. This approach also shares may design goals with SGX, but requires the application provider to trust the VMM implementation. It is also vulnerable to a hardware based attack, where an adversary with physical control of the platform could attain the encryption keys used by the VMM. SGX protects against many types of physical attacks.

Flicker and Overshadow share many security goals with SGX. One could imagine creating an application that takes advantage of a Flicker or Overshadow type approach on platforms without the SGX technology, and seamlessly makes use of SGX capabilities on platforms which contain the feature.

6 Acknowledgements

The authors of this paper wish to acknowledge the contributions of many people throughout Intel who dedicated countless hours to defining, refining, and implementing hardware and software prototypes of the SGX technology. We also wish to thank software developers from within Intel and partner companies who reviewed and helped refine the SGX software model.

Finally we wish to recognize the support of the United States Department of Homeland Security and the United States Air Force Academy in the design and prototyping of the ERM and Secure Video Conferencing applications.[i]

7 References

[1] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue and U. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," in *Hardware and Architectural Support for Security and Privacy (HASP)*, Tel Aviv, Israel, 2013.

[2] I. Anati, S. Gueron, S. P. Johnson and V. R. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing," in *Hardware and Architectural Support for Security and Privacy (HASP)*, Tel Aviv, Israel, 2013.

[3] Intel Corporation, "Graphics - Blu-Ray Disc* playback with the Intel(R) Graphics (FAQ)," [Online]. Available: http://www.intel.com/support/graphics/sb/CS-029871.htm#whatis. [Accessed 17 June 2013].

[4] Intel Corporation, "Intel(R) Active Management Technology," [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/intel-active-management-technology.html. [Accessed 17 June 2013].

[5] McAfee Labs and McAfee Foundstone Professional Services, "Protecting Your Critical Assets: Lessons Learned from "Operation Aurora"," [Online]. Available: http://www.mcafee.com/us/resources/white-papers/wp-protecting-critical-assets.pdf. [Accessed 17 June 2013].

[6] J. A. Haldermen, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Clalandrino, A. J. Feldman, J. Appelbaum and E. W. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," 2008. [Online]. Available: https://citp.princeton.edu/research/memory/. [Accessed 17 June 2013].

[7] D. Aucsmith, "Tamper Resistant Software: An Implementation," in *Proceedings of the First international Workshop on information Hiding*, London, 1996.

[8] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," *Proceedings of the ACM European Conference on Computer Systems (EuroSys'08),* March 2008.

[9] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin and D. R. Ports, "Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (ASPLOS XIII)," New York, NY, USA, 2008.

---

[i] The Department of Homeland Security (DHS) sponsors the Center of Innovation at the United States Air Force Academy, which conducts research for educational purposes. The United States Air Force Academy and DHS sponsored the production of this material under United States Air Force Academy agreement number FA7000-11-2-0001. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of The United States Air Force Academy or the U.S. Government.