The Million Dollar Handshake: Secure and Attested Communications in the Cloud

Nikolaos Chalkiadakis, Dimitris Deyannis*, Dimitris Karnikis*, Giorgos Vasiliadis
FORTH-ICS
Heraklion, Crete, Greece
{*nikoschalk, deyannis, dkarnikis, gvasil*}*@ics.forth.gr*
**Also with the Department of Computer Science, University of Crete, Greece*

Sotiris Ioannidis†
Technical University of Crete
Chania, Crete, Greece
sotiris@ece.tuc.gr
†*Also with the Institute of Computer Science, Foundation for Research & Technology Hellas*

***Abstract*—The number of applications and services that are hosted on cloud platforms is constantly increasing. Nowadays, more and more applications are hosted as services on cloud platforms, co-existing with other services in a mutually untrusted environment. Facilities such as virtual machines, containers and encrypted communication channels aim to offer isolation between the various applications and protect sensitive user data. However, such techniques are not always able to provide a secure execution environment for sensitive applications nor they offer guarantees that data are not monitored by an honest but curious provider once they reach the cloud infrastructure.**

The recent advancements of trusted execution environments within commodity processors, such as Intel SGX, provide a secure reverse sandbox, where code and data are isolated even from the underlying operating system. Moreover, Intel SGX provides a remote attestation mechanism, allowing the communicating parties to verify their identity as well as prove that code is executed on hardware-assisted software enclaves. Many approaches try to ensure code and data integrity, as well as enforce channel encryption schemes such as TLS, however, these techniques are not enough to achieve complete isolation and secure communications without hardware assistance or are not efficient in terms of performance.

In this work, we design and implement a practical attestation system that allows the service provider to offer a seamless attestation service between the hosted applications and the end clients. Furthermore, we implement a novel caching system that is capable to eliminate the latencies introduced by the remote attestation process. Our approach allows the parties to attest one another before each communication attempt, with improved performance when compared to a standard TLS handshake.

***Keywords*-Remote Attestation; Secure Communication; Secure Enclaves;**

I. Introduction

The widespread use of cloud infrastructure has led a plethora of applications and services to relocate from private, on-premise, servers to cloud platforms. These platforms typically host a wide variety of software, thus, it is becoming more and more common for user-end applications to communicate with a remote cloud-based server or in a peer-to-peer fashion. Such applications range from data storage and processing solutions [1], [2], [3], chat services [4], medical and financial applications [5], to productivity software, anti-malware systems [6] and IoT oriented applications controlling business or household appliances [7]. This trend leads security and privacy sensitive applications co-existing with other software on the same environment, communicating with the user applications via standard encryption schemes, such as TLS.

In such environments, where various types of applications are co-hosted together, the use of virtual machines and compartmentalization are the most fundamental means of isolating software. However, such facilities do not provide guaranties for the integrity of software and data nor they shield software against adversaries controlling one of the communicating ends. Moreover, standard communication channel encryption schemes, such as TLS, do not secure the channel from an honest but curious service provider that wishes to monitor the offloaded user data.

In order to address these issues, recent research work has made use of the Intel Software Guard Extensions (SGX), in order to design and implement systems that offer a means of software isolation for applications hosted on such environments [8], [9]. Intel SGX was introduced as an ISA extension with the 6th generation of Intel processors and provides hardware-assisted software enclaves that operate as reverse sandboxes, protecting code and data from other applications, debugging utilities and even the operating system kernel. However, the majority of these works do not provide a service that will allow the establishment of a secure communication channel between enclaves of different entities.

In this work, we design and implement a system that can provide a secure communication channel establishment for SGX-enabled applications where at least one of the commu-

nicating ends may reside on a potentially untrusted remote location, such as a cloud platform. For this purpose, we build our system based on Intel's Remote Attestation process in order to enable secure key exchange and authorization of the communicating ends. Our system exposes a simple API with which, two communicating entities can verify and attest each other, identifying if signed SGX-enclaves are used at both ends – running with hardware support, and create an SGX-to-SGX encrypted communication channel. Moreover, we implement a caching system for SGX Remote Attestation responses which greatly reduces the latency of new connections and benefits applications that require multiple short-term connections.

The contributions of this paper are:

- A system that provides remote attestation of communicating ends executing inside SGX enclaves
- A framework that provides seamless establishment of enclave-to-enclave encrypted network communication between two or more parties
- A caching system for SGX remote attestation responses that reduces the latency of consecutive connections, rendering them substantially faster than a standard TLS handshake
- A comparison of our secure communication method against commonly used methods such as SSL/TLS

II. Background

A. Intel SGX

Intel SGX (Software Guard eXtensions) [10] is an ISA extension to the Intel architecture that provides secure software *enclaves*, backed by hardware support, that aim to offer protection for sensitive code segments and data against disclosure and modifications. SGX enclaves operate as reverse sandboxes, meaning that other system processes, debugging utilities and the operating system do not have access to their code or data while they are being executed. This enables SGX-assisted applications to securely operate even when executed on an untrusted or even malicious platform with a tampered operating system, firmware or hypervisor. Such protection is made possible as enclave memory pages reside in a protected physical memory space called Enclave Page Cache (EPC). When data from the protected memory region are moved to DRAM, they are automatically encrypted by an on-chip memory encryption engine called MME.

Enclave memory is also protected against memory modifications and rollbacks via integrity checking. The EPC size ranges between 64MB and 128MB and the Intel SGX platform provides a secure paging mechanism for swapping EPC pages to the untrusted DRAM by utilizing the MME. Software residing out of the enclave is not able to access the EPC [11] while enclave code can directly access the untrusted DRAM contents. Moreover, data moved from the enclave to the file system can be sealed and later be decrypted and checked for their integrity, upon usage.

An SGX enabled application is divided into two parts: (i) the untrusted code, residing in DRAM and (ii) the trusted enclave, developed in C/C++, residing in the protected memory space. Application code can be placed into an enclave only at its developing phase by special APIs and software, made available by the Intel SGX SDK. Moreover, enclaves can be signed during compilation and later be attested locally or by a remote party. The untrusted segment of the application can communicate with the secure enclave through specific interfaces, defined by the developer and any other attempts to access the enclave result in segmentation violations.

Enclaves can be created using the `ECREATE` instruction, which initializes an SGX enclave control structure (SECS) in the EPC. The `EADD` instruction adds pages to the enclave, which are further tracked and protected by the SGX (i.e., the virtual address and its permissions). The `EINIT` instruction creates a cryptographic measurement, after the loading of all enclave pages. The cryptographic measurement can be used by remote parties for attestation. After the enclave has been initialized, enclave code can be executed through the `EENTER` instruction, which switches the CPU to enclave mode and jumps to a predefined enclave offset. The `EEXIT` instruction causes execution to leave the enclave.

B. SGX Remote Attestation

Remote attestation is the process of verifying the authenticity of a software component, running inside an isolated container, to some remote party. In the case of SGX, the software being attested is a secure enclave created by the trusted CPU hardware. For the remote attestation procedure, the CPU generates a measurement for the attested enclave which uniquely identifies it. This information is then signed by the privileged Quoting Enclave, resulting in an attestation signature (QUOTE). The Quoting Enclave is a special trusted enclave software, which is issued by Intel and has access to the SGX hardware attestation key that signs the measurement. The attestation signature is generated using the EPID group signature scheme [12] in order to preserve privacy. The communication between the two enclaves must also be done in a secure way. This is achieved by performing a local attestation between the two communicating enclaves as a means to establish a secure channel. The attestation signature can then be sent to the remote party, who will relay this information to the Intel Attestation Service (IAS) in order to verify its validity. Thus, the remote party can be aware if the enclave has been tampered or if the attested software is not running within a genuine hardware-assisted SGX enclave. This information is critical as it verifies that the SGX enclave is executed on SGX enabled hardware and not in simulation mode, which makes the enclave accessible by debugging utilities. The SGX Remote Attestation process

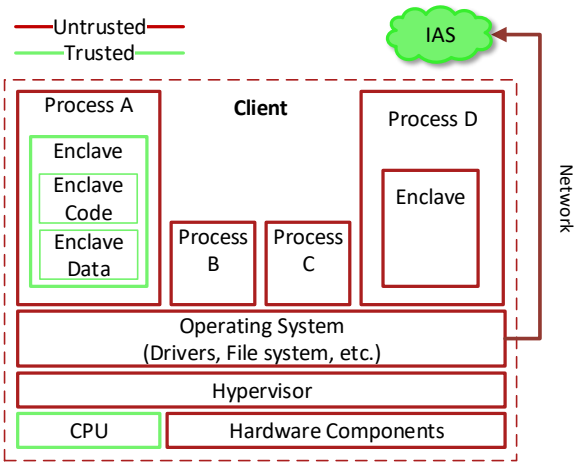

Figure 1: Assumed TCB and possibly malicious components

utilizes a modified SIGMA [13] protocol, therefore at the end of the process the remote party and the enclave establish a shared secret for secure communication. Contrary to Trusted Platform Modules (TPMs), SGX Remote Attestation has the benefit that attested software runs within the CPU thus, having better performance. Moreover, SGX utilizes an EPID group signature scheme and attested enclaves cannot be uniquely linked back to a specific CPU through their attestation signature.

III. Threat Model and Assumptions

In this work, we assume a set of client nodes that may be distributed over different area networks or even different geographical areas. These nodes are connected over a public, not necessarily trusted, network, over which they can transmit and exchange data. We also assume that the client nodes may be compromised by a powerful adversary with full-privileged access or even access to the physical hardware, except the CPU, that is further equipped with SGX (or equivalent technology). All the enclaves that can run within SGX in each client are considered part of our TCB, as well as the Service Provider, which acts as a directory server responsible for resource and users discovery, and the Intel SGX Attestation Service that verifies that the SGX hardware is genuine; any other components, such as the host application and the underlying OS are not. The entities that are part of our TCB are the Intel SGX enclaves that can run in each client (excluding the container application and the hosting OS), the Service Provider and the IAS. An overview of our TCB and possibly malicious components is presented in Figure 1.

Overall, our primary aim is to defend against adversaries that can: **(a)** control and tamper applications and the clients' operating system, **(b)** observe and tamper data transmitted over the network channel, **(c)** conduct man-in-the-middle attacks between either a client and the SP or between two client enclaves, and, **(d)** perform replay attacks that utilize information from previous sessions.

We note that our threat model excludes Denial-of-Service (DoS) attacks on enclaves since the life cycle of the process containing the enclave can be controlled by a malicious operating system or super-user. While adversaries can prevent or abort the execution of enclaves, they are not able to obtain any valuable information by doing so. Furthermore, side-channel attacks that exploit page faults or timing information are excluded from this work, even though we assume a small and well audited TCB. Finally, we assume that the design and implementation of the Intel SGX framework are free of vulnerabilities.

IV. Design Overview

A. Involved Parties

Our protocol involves three parties in total, namely the *clients*, the *service provider*, and the *attestation service*.

Clients. The remote parties that want to communicate and support Intel SGX enclaves. For simplicity, in the rest of this paper, we assume only two clients, Alice and Bob, where Alice wants to communicate with Bob via a secure communication channel, using Intel SGX. We notice though that our design can scale by nature and operate independent of the number of connected peers.

Service Provider (SP). The application's vendor who signs and ships the clients' SGX enclaves. Signing verifies that the code can not be changed, tampered, or altered. Service Providers must register themselves with the Intel Attestation Service in order to make use of the provided services. To do so, the Service Providers must fulfill a set of standard requirements in order to submit their attestation results to the attestation service and verify that the system is sound. Consequently, this process assigns a TLS certificate to the Service Provider ID (SPID), thus granting access to the attestation services.

Intel Attestation Service (IAS). The IAS is used in order to verify the attestation evidence that the Quoting Enclave generates and report them back to the Service Provider. Quoting Enclave (QE) is provided by Intel and its goal is to receive and verify reports from other enclaves, which converts and signs using the Intel EPID Key, which is a device specific asymmetric key. The attestation flow is the following:

1) The application receives an attestation request (REPORT) from a third party and forwards it to its enclave.
2) The enclave sends the local attestation request to the Quoting Enclave.
3) The Quoting Enclave validates the request and transforms it int a remote attestation request.
4) The request is returned to the application which forwards it to the challenger.

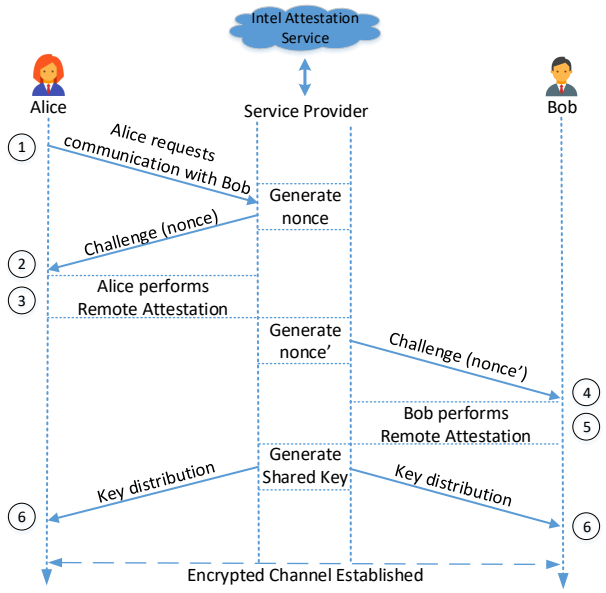

Figure 2: Attestation protocol

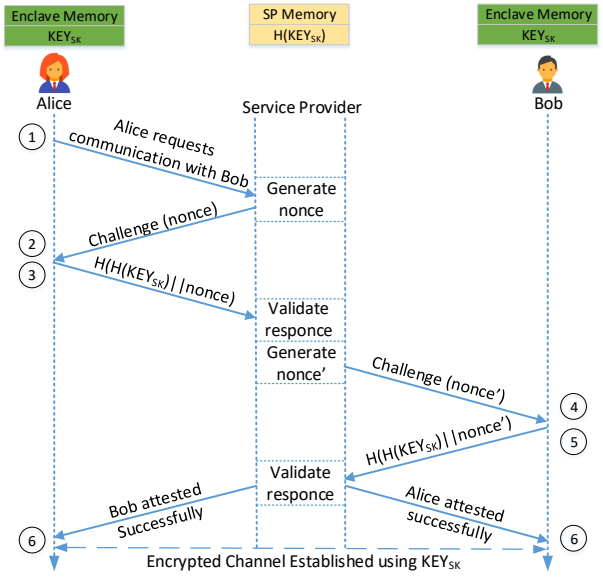

Figure 3: Attestation protocol with response caching

5) The challenger uses the Attestation Verification service to perform the verification.

B. Protocol Design

In this section, we present our protocol in detail, describing the messages exchanged between two clients, namely Alice and Bob. In our setup, we assume that Alice wants to initiate the communication. A graphical representation of the message flow is depicted in Figure 2. More specifically, the actions performed by the two parties are the following:

1) Alice requests communication with Bob from the Service Provider.
2) The Service Provider challenges Alice with a message containing a *nonce*.
3) Alice attests to the Service Provider by sending her generated *Quote*. The Service Provider sends the Quote to the IAS afterwards and checks the response. By doing so, the Service Provider verifies that Alice utilizes an untampered enclave, running on a genuine Intel SGX hardware. In this step, the Service Provider also checks the key that has been used to sign the Enclave (MRSIGNER), the hash value of the Enclave (MRENCLAVE), the software version of the Enclave (ISV_SVN) and the Enclave's product ID (ISV_ PROD_ID) which allows multiple enclave instances to be distinguishable. Intel [14], Hile Vill [15] and others [11] have thoroughly described this remote attestation process.
4) If the attestation is successful, the Service Provider challenges Bob.
5) Bob performs the same attestation process as Alice.
6) If the attestation is successful, the Service Provider notifies Bob that Alice wants to communicate with him. Then, the Service Provider generates a shared secret for the establishment of the secure enclave-to-enclave encrypted channel between the two clients. For each client, the secret is distributed via a secure channel that has its client-side endpoint within the enclave and has been established during the IAS attestation step.

After completing the aforementioned steps, the enclave-to-enclave channel between the two clients is established and Alice can communicate securely with Bob. The shared secret is used to guarantee the confidentiality, integrity and authenticity between the two enclaves and is never exposed outside of the SGX environment.

The protocol follows a serial approach and we leave its parallelization as part of our future work. Additionally, in our model we assume that Alice does not know how to directly communicate with Bob and that Bob does not trust anyone that has not been verified by the Service Provider. This introduces a level of centralization which we discuss in Section VII-A.

C. Attestation Caching

The remote attestation procedure, as shown in Figure 2, consists of multiple stages across different entities and, as a consequence, it may require a significant amount of time to complete. These times are accumulated in cases where applications require to communicate with multiple enclaves at the same time, since they would have to attest to the Service Provider for each individual session. To

make matters worse, the overhead of session establishment becomes even more notable for short-term sessions, in which the time required for the initiation of the session can be significantly larger than the overall communication itself.

In order to reduce the remote attestation overheads, we extend our basic protocol that has been described in Section IV-B, by further introducing a cache layer mechanism at the Service Provider. The mechanism is responsible for caching the remote attestation results that are produced by IAS, aiming to increase the protocol's performance. Our approach also allows the fine-tuning of the caching period, based on the security requirements of the enclave that the Service Provider is attesting. During protocol initiation, the enclaves inform the Service Provider about their intent to use cached attestation results. However, the Service Provider is the final arbiter of whether or not a full remote attestation should take place.

We implement two caching approaches:

1. TTL-based ephemeral session keys. In this approach, the Service Provider and the client enclaves store the session keys in memory for future use. This is a safe operation since the enclave's memory is encrypted, checked for integrity violations and is also inaccessible from the untrusted environment. The Service Provider can define a Time-To-Live (TTL) for which the keys are assumed to be valid before requiring the client enclave to perform a complete remote attestation again. In each attestation attempt, the Service Provider challenges the enclave with a $nonce$ and expects the correct $HMAC$ value as a response.

2. Session key hashes. During the first protocol instantiation with the client enclaves, the Service Provider uses a secure hash function H to store the hash value of a session key, $H(KEY_{\text{SK}})$, as opposed to storing actual keys in the previous approach. Afterwards, each time the client's enclave attests to the service provider, the Service Provider issues a $nonce$ to the enclave and expects it to respond with the correct $H(H(KEY_{\text{SK}})||nonce)$ value. This method is similar to remote user authentication using passwords [16] but since the keys are ephemeral there is no need for salt. The stored hash values can be associated with a TTL field or a fixed number of maximum N sessions to be initiated using the same ephemeral key.

Both approaches can be applied by the Service Provider, meaning that different enclaves that attest to the Service Provider can follow different approaches. Additionally for the implementation of the caching mechanism, the Service Provider can decide on which key will be used as caching key. This can be achieved in two different ways:

1. Enclave session key caching. The service provider uses as caching key the session key established with the attested enclave during the remote attestation with the IAS in order to implement caching for that particular enclave.

2. Enclave-to-Enclave channel shared secret caching. The service provider uses as caching key the shared secret generated by the Service Provider during the establishment of the secure communication channel between the two enclaves in order to implement caching for that pair of enclaves.

It is important to note that the Service Provider can force the enclaves to perform a complete remote attestation at any protocol instance or even opt-out of caching depending on the security requirements and access control policies of the enclaves. Furthermore, when attestation caching is used, the IAS is not involved. This reduces the privacy footprint of an enclave regarding how frequently it attests to Intel. A graphical representation of the protocol using remote attestation caching, using *Session key hashes* and *Enclave-to-Enclave channel shared secret caching*, is presented in Figure 3.

V. Implementation

We implement the clients and the SP utilizing our proposed protocol using C/C++ and the Intel SGX SDK 2.6. Also, we use OpenSSL 1.1.0h, build on top of the SGX framework in order to perform all the required cryptographic operations. The overall system consists of 7290 lines of code. Three different machines were used in order to evaluate and test the implementation of our protocol, involving two clients and one SP. The two clients run Arch Linux with LTS Kernel 4.19 and utilize an Intel Core i7-8700K processor clocked at 3.70GHz along with 32GB of DDR4 memory. The SP is hosted by an Ubuntu 18.04 system, running Linux Kernel 5.0.0-36-generic, utilizing an Intel Core i7-8565U CPU clocked at 1.80GHz along with 16GB of DDR4 memory. All the enclaves and the SGX related binaries are built and executed in SGX Hardware Mode in order to take advantage of the SGX capabilities.

VI. Evaluation

For the evaluation of our protocol, we measure the time required for establishing a secure enclave-to-enclave communication channel between two clients, using our custom Remote Attestation approach. Then, we compare the results to the time required to perform a standard TLS handshake with a valid certificate signed by an Intermediate Certificate Authority (CA).

We perform 200 TLS handshakes and measure the time required for the client to establish the TLS connection with the server. Also, we perform an additional 400 secure channel establishments using our Remote Attestation protocol, out of which half are performed without caching while the other half are performed with *Session key hashes* and *Enclave-to-Enclave channel shared secret caching*.

As seen in Figures 4 and 5, our protocol, when no caching schemes are utilized, is approximately 3.5 times slower compared to the TLS handshake. We also observe that the standard deviation for the TLS handshake is 62.15ms while for our protocol it is 138.03ms. This is due to the

Table I: Time Statistics (in milliseconds)

	TLS Handshake	Remote Attestation without caching	Remote Attestation with caching
Average	974.93	3269.66	4.20
Standard Deviation	62.15	138.03	0.27
Minimum	733.07	3113.54	3.43
Maximum	1088.67	3940.94	5.06

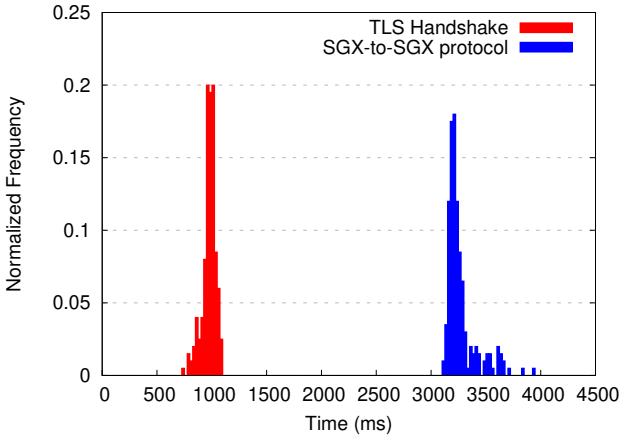

Figure 4: Execution time normalized frequencies.

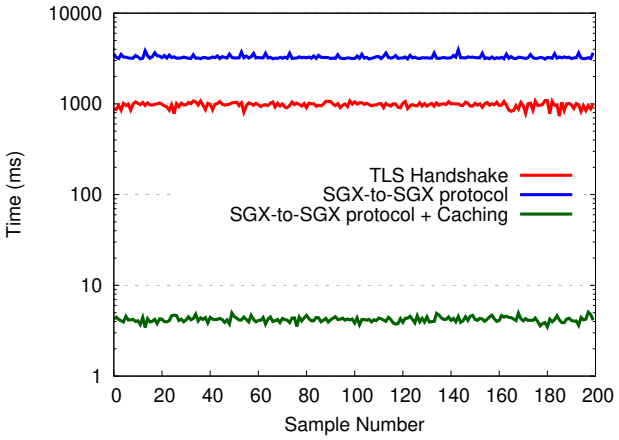

Figure 5: Execution time comparison.

fact that our protocol relies on the IAS, meaning that the time required for communicating with the IAS over the network and performing the remote attestation is also measured. However, as presented in Figure 5, we observe that our approach, when caching is deployed, is three orders of magnitude faster than the TLS handshake, with mean establishment time of 4.20ms and standard deviation of 0.27ms. These time statistics are more thoroughly presented in Table I.

VII. Discussion

A. Protocol Centralization

During our research of the Enclave-to-Enclave communication problem, we came across the option of either including an intermediary remote party, namely the Service Provider, or implement a completely decentralized approach. We chose to include the Service Provider because it offers the following benefits:

(1) Remote Attestation is a complex procedure involving trust policies which can vary based on the required security and strictness of the application. These policies become even harder to define when there are multiple and different enclaves involved. The Service Provider can dynamically change those policies without any modification to the enclave code or redeploying them.

(2) Different policies for different enclaves can be established. The Service Provider can allow different types of enclaves communicating with each other and then control the various trust levels between them.

(3) Only the Service Provider knows the communicating parties (Alice and Bob) and the protocol does not depend on other peer discovery primitives which can have privacy implications.

(4) We can assume that an entity with the role of the Service Provider exists as enclaves are signed before being deployed in a production environment. Any enclave that is to be provisioned secrets from the network should always be attested, thus raising the need for such a remote party. Moreover, since enclaves cannot be tampered, techniques like certificate pinning in combination with Enclave Signature Revocation Lists [15] can enhance the security even further.

Finally, the Service Provider does not have to be a single point of failure as many instances can coexist. The remote party acts mostly as a policy enforcing entity and coordinator between communications. As part of our future work, we aim to explore an alternative protocol where we completely remove the Service Provider and allow the enclaves to make their own decisions for the various trust policies.

B. Limitations

Recent research [17], [18], [19], [20], [21] has proven that side-channel attacks are feasible on Intel SGX enclaves. However, protecting SGX enclaves from side-channel attacks that either focus on software or hardware bugs are

orthogonal to our work and thus we do not consider them. On the other hand, any successful attempt to harden the SGX framework [22], [23] has a direct benefit to our work.

Intel SGX manages and controls the behavior, lifetime and execution of a trusted application, hence our framework is based on it. Denial-of-Service (DoS) attacks based on preventing the execution of the process hosting SGX enclaves, initiated by a malicious operating system or super user, are a vulnerability common to all applications that utilize the SGX framework. These attacks are orthogonal to the security provided by enclaves and thus we exclude them from our threat model.

Additionally, it is assumed that the code and data of the communicating parties are free of security bugs and memory leaks, otherwise, side-channel attacks [24] that exploit such vulnerabilities can have a negative impact on the security offered by SGX. These are issues that must be addressed by the developers by maintaining a secure TCB and are not handled by our model.

Lastly, Yogesh P. Swami has shown that [25] SGX's EPID and attestation report do not hold onto their claimed anonymity and privacy guarantees. However, third parties can build their own attestation infrastructure [26] in order to replace the IAS and mitigate this issue as well as solving the single point of failure problem.

VIII. Related Work

Many applications and recent research utilize Intel SGX and it has been integrated in many large-scale projects that have need for increased security. SGX has been used to enhance the Snort IDS [27] and the TOR network [28], while efforts have also been made to move TLS endpoints inside SGX enclaves [29]. Additionally, digital rights management (DRM) content has a direct benefit from the usage of SGX enclaves.

Related to our work, Balfe et al. [30] have shown how TPMs can be used in peer-to-peer networks in order to provide security. In their work, they use the Trusted Computing technology in order to establish pseudonymous identifiers and build secure channels between peers. Additionally, mutual attestation with TPMs has also been used in creating protocols for trusted RFID systems [31]. Moreover Shepherd et al. [32] raise the challenge of secure TEE-to-TEE communication between remote sensing devices.

Furthermore, the Intel SGX technology has been used for Secure Many-Party applications [33] and Secure Multiparty Computation [34], where the enclaves act as Middleboxes [35]. For example, ShieldBox [35] utilizes Intel SGX in order to provide secure middleboxes for high-performance network functions that can be deployed on untrusted servers. Our work can be applied for Middleboxes so as to communicate using mutually trusted channels.

Finally, `SANCTUARY` [36] is a system based on ARM TrustZone that enables the execution of security-sensitive applications within strongly isolated compartments. These compartments are mutually distrusted and reside within ARM's TrustZone world, thus being comparable to SGX's enclaves. Their framework offers attestation service to a third party using a *Proxy Sanctuary Application* which is comparable to Intel's Quoting Enclave. Our work can also be applied to the model offered by `SANCTUARY`.

IX. Conclusion

In this paper, we introduce a protocol leveraging the Remote Attestation feature provided by the Intel SGX framework in order to build mutually trusted secure communication channels between two enclaves, possibly residing in different physical machines. Furthermore, we propose a caching scheme for the remote attestation results that does not compromise the security and privacy of our model and speeds up consecutive remote attestation processes.

Finally, we evaluate our system by comparing the time required for establishing a secure channel between two enclaves using our approach to the time required for a TLS handshake to be completed. Our results show that our system has x3.5 more overhead compared to the TLS protocol when we do not utilize any caching schemes. However, our protocol instantiation has substantially increased performance when attestation caching is applied, rendering it faster than a standard TLS handshake.

Acknowledgments

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. No. 2767). This work was also supported by the projects CONCORDIA, C4IIoT and COLLABS, funded by the European Commission under Grant Agreements No. 830927, No. 833828, and No. 871518. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

[1] "A More Protected Cloud Environment: IBM Announces Cloud Data Guard Featuring Intel SGX," https://itpeernetwork.intel.com/ibm-cloud-data-guard-intel-sgx/.

[2] "Fortanix and Alibaba Cloud Partner to Launch SDKMS Runtime Encryption Key Management on Alibaba Cloud ECS to Protect Sensitive Data," https://fortanix.com/company/news/pr/2018/10/fortanix-and-alibaba-cloud-partner/, 2018.

[3] "Microsoft Azure* Confidential Computing with Intel SGX," https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions/microsoft-confidential-computing-sgx-video.html.

[4] "Signal taps up Intel's SGX to (hopefully) stop contacts falling into hackers, cops hands," https://www.theregister.co.uk/2017/09/27/signal_turns_to_intels_sgx_to_lock_down_contacts_from_spying_eyes/, 2017.

[5] "Blockchain Approaches to Data Privacy in Healthcare," https://medium.com/inside-r3/blockchain-approaches-to-data-privacy-in-healthcare-e6e7f114094c.

[6] Dimitris Deyannis, Eva Papadogiannaki, George Kalivianakis, Giorgos Vasiliadis and Sotiris Ioannidis, "Trustav: Practical and privacy preserving malware analysis in the cloud," in *CODASPY'20*.

[7] "Making Blockchain and IoT Deployments More Secure with Intel SGX," https://itpeernetwork.intel.com/blockchain-intel-sgx/.

[8] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell *et al.*, "SCONE: Secure Linux Containers with Intel SGX," in *12th USENIX OSDI*, 2016.

[9] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX," in *USENIX ATC*, 2017.

[10] "Intel sgx," https://software.intel.com/en-us/sgx.

[11] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, 2016.

[12] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, vol. 1, 2016.

[13] H. Krawczyk, "Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike protocols," in *CRYPTO*, 2003.

[14] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13, 2013.

[15] H. Vill, "Sgx attestation process," 2017.

[16] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer security: principles and practice*, 2012.

[17] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing," in *26th USENIX Security*, 2017.

[18] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium*, 2018.

[19] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:sgx cache attacks are practical," in *11th USENIX WOOT*, 2017.

[20] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th EUROSEC*, 2017.

[21] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How sgx amplifies the power of cache attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2017.

[22] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.

[23] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with déjá vu," in *Proceedings of the 2017 AsiaCCS*, 2017.

[24] Caddy Tom, "Side-channel attacks," https://link.springer.com/referencework/10.1007%2F0-387-23483-7, 2011.

[25] Y. Swami, "Intel sgx remote attestation is not sufficient," *IACR*, 2017.

[26] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, "Supporting third party attestation for intel® sgx with intel® data center attestation primitives," *White Paper*, 2018.

[27] D. Kuvaiskii, S. Chakrabarti, and M. Vij, "Snort intrusion detection system with intel software guard extension (intel sgx)," *arXiv preprint arXiv:1802.00508*, 2018.

[28] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Sgx-tor: A secure and practical tor anonymity network with sgx enclaves," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, 2018.

[29] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, "Integrating remote attestation with transport layer security," *arXiv preprint arXiv:1801.05863*, 2018.

[30] S. Balfe, A. D. Lakhani, and K. G. Paterson, "Trusted computing: Providing security for peer-to-peer networks," in *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, 2005.

[31] M. F. Mubarak, S. Yahya *et al.*, "Mutual attestation using tpm for trusted rfid protocol," in *2010 Second International Conference on Network Applications, Protocols and Services*, 2010.

[32] C. Shepherd, R. N. Akram, and K. Markantonakis, "Establishing mutually trusted channels for remote sensing devices with trusted execution environments," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017.

[33] K. A. Küçük, A. Paverd, A. Martin, N. Asokan, A. Simpson, and R. Ankele, "Exploring the use of intel sgx for secure many-party applications," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*, 2016.

[34] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from sgx," in *International Conference on Financial Cryptography and Data Security*, 2017.

[35] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Shieldbox: Secure middleboxes using shielded execution," in *Proceedings of the Symposium on SDN Research*, 2018.

[36] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Sanctuary: Arming trustzone with user-space enclaves." in *NDSS*, 2019.