The Impact of Attestation on Deniable Communications

Ricardo Iván Vieitez Parra

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Supervisors

Prof. N. Asokan

Prof. Danilo Gligoroski

Advisor

Dr. Lachlan Gunn

Aalto University
School of Science

NTNU
Department of Information Security and Communication Technology

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Ricardo Iván Vieitez Parra

Title The Impact of Attestation on Deniable Communications

Degree programme Security and Mobile Computing (NordSecMob)

Major Security and Mobile Computing **Code of major** T3011

Supervisors Prof. N. Asokan, Prof. Danilo Gligoroski

Advisor Dr. Lachlan Gunn

Date 3/7/2018 **Number of pages** 58+13 **Language** English

Abstract

Deniable protocols allow their participants to plausibly deny their participation or what was being said. A variety of Internet and messaging protocols are designed with the goal of providing deniability as a means of enhancing privacy. At the same time, commodity hardware often has the ability to provide Trusted Execution Environments (TEEs) with remote attestation capabilities, which can provide credible proof to an external verifier of some state or property existing or having existed on a device. We show that the potential availability of remote attestation implies that unconditionally deniable protocols are impossible if they also provide message authentication. We then present an practical attack that can be used to compromise deniability in the Signal messaging protocol, and introduce some possible countermeasures that restore deniability.

Keywords deniability, deniable protocols, deniable authentication, remote attestation, repudiation, Signal protocol, X3DH, Trusted Execution Environment, TEE, Intel SGX, trusted hardware, zero-knowledge proofs

Aalto University
School of Science

Aalto-universitetet, PB 11000, 00076
AALTO
www.aalto.fi
sammandrag av diplomarbetet

Författare Ricardo Iván Vieitez Parra

Titel Påverkan Av Attestering i Förnekbar Kommunikation

Utbildningsprogram Säkerhet samt Mobil Kommunikation (NordSecMob)

Huvudämne Säkerhet samt Mobil Kommunikation **Huvudämnets kod** T3011

Övervakare Prof. N. Asokan, Prof. Danilo Gligoroski

Handledare FD Lachlan Gunn

Datum 3.7.2018 **Sidantal** 58+13 **Språk** Engelska

Sammandrag

Förnekbara protokoll är de där deltagarna kan plausibelt förneka sin delaktighet eller vad som sades. En mängd olika Internet- och meddelandeprotokoll är utformade med målet att tillhandahålla förnekbarhet som ett sätt att förbättra integritet. Samtidigt har konsumentvaror ofta förmågan att tillhandahålla pålitliga exekveringsmiljöer (TEE) med möjlighet till fjärrattestering, som kan ge trovärdiga bevis åt en extern bekräftare om att vissa tillstånd eller egenskaper som existerar eller har existerat på en enhet. Den möjliga tillgängligheten av fjärrattestering innebär att ovillkorligt förnekbar protokoll är omöjliga om de också erbjuder meddelandeautentisering. En praktisk attack presenteras som kan resultera i kompromiss av förnekbarhet i meddelandeprotokollet Signal, och efteråt några möjliga motåtgärder som återställer förnekbarhet.

Nyckelord förnekbarhet, förnekbara protokoll, förnekbar autentisering, förkastande, Signal protokoll, X3DH, pålitliga exekveringsmiljöer, TEE, Intel SGX, pålitliga datorhårdvara, nollkunskapsbevis

Preface

The thesis before you presents how Trusted Execution Environments (TEEs), available in a variety of off-the-shelf consumer hardware, may be used to compromise the deniability of communications. It was written to fulfil the requirements for the degree of master of science as part of my studies in the master's programme in security and mobile computing that I pursued at Aalto University and the Norwegian University of Science and Technology.

I was engaged in conducting research and writing this thesis at the Secure Systems Group at Aalto University between January and June 2018. The topic of my research was arrived to together with my supervisor at the Secure Systems Group, Prof. N. Asokan, and my advisor, Dr. Lachlan Gunn, and this work partly contributed to the paper *On The Use of Remote Attestation to Break and Repair Deniability*.

During these six months, I devoted most of my work to implementing the attack presented in chapter 4 and its corresponding defence, shown in chapter 5. This was at times a daunting task, especially in finding appropriate documentation of the components used and deciding on a sound design for this implementation. I have learnt much as a result, particularly about TEEs and the functioning of deniable communication protocols like Signal.

I find this topic particularly important and relevant for today's communications; messaging applications are increasingly popular as a way of staying in touch and many promise privacy —that what we say stays between us and the people we are talking to— because we use them to share intimate or embarrassing details. Deniable protocols enhance our ability to speak freely and confidently by giving additional assurances that third parties will not be able to verify what was said, or even that a conversation took place.

I would like to thank my supervisors and my advisor for their guidance and support throughout this time, and without whom this work would not be possible. I also wish to express my gratitude to my colleagues at the SSG, Fritz Alder, Arseny Kurnikov and Koen Tange for their readiness to assist with the many technical challenges that I faced as well as the rest of my coworkers for their support and motivation. Likewise, I thank my parents for their guidance and support throughout my studies.

I wish you pleasant reading,

Espoo, 3/7/2018

Ricardo Iván Vieitez Parra

Contents

Symbols and abbreviations

Symbols

$\mathbb{N}$	the set of natural numbers, i.e., $\{1,2,3,\ldots\}$
$\mathbb{Z}$	the set of integers, i.e., $\{0\} \cup \{1,2,3,\ldots\} \cup \{-1,-2,-3,\ldots\}$
$\mathbb{Z}_0^+$	the set of all non-negative integers, i.e., $\{0,1,\ldots\}$
$\mathbb{Z}_n$	the set of all non-negative integers smaller than n, i.e., $\{0,1,\ldots,n-1\}$
$\mathbb{B}$	the binary set, i.e., $\{0,1\}$
π	a protocol
π^*	a protocol that simulates a protocol π
τ	a protocol executed in a TEE
$\mathcal{M}$	a message sent using some protocol
$\mathcal{T}$	a transcript from executing some protocol, i.e., the tuple of all its inputs and outputs

Operators

$\varphi(n)$	Euler's totient function for n
$\lvert S \rvert$	the cardinality of the set S
$\operatorname{ord}(\mathcal{G})$	the order of the group $\mathcal{G}$
g^X	the corresponding public key of a secret key X in an asymmetric cryptosystem
$\left(g^X\right)^Y$	the Diffie-Hellman agreement or a similar operation in an asymmetric cryptosystem between a secret key Y and a public key g^X
$\Pr(X)$	the probability of an event X occurring
$\Pr(X\|Y)$	the conditional probability of an event X occurring given that Y is true
$\bar{x}$	the sample mean of a set of samples x, i.e., $\sum_{i \in x} \frac{i}{\lvert x \rvert}$
s_x	the sample standard deviation of a set of samples x, i.e., $\sqrt{\sum_{i \in x} \frac{(i-\bar{x})^2}{\lvert x \rvert - 1}}$
$X \to Y$	a mapping from the set X to the set Y
$X \twoheadrightarrow Y$	a surjection from the set X onto the set Y

Other Notation

K_A	the key of type K corresponding to a party A; for example, if I denotes an identity key, I_A is A's identity key
🔒K	a sealed copy of the key K
`X`	the processor instruction `X`; for example, `MOV`

Acronyms and Abbreviations

AE	authenticated encryption
AEAD	authenticated encryption with additional data
AES	Advanced Encryption Standard, a cipher
AEX	asymmetric execution event, a type of event in SGX that may interrupt an enclave execution
CMAC	cipher-based message authentication code, a subfamily of MAC algorithms based on the use of a block cipher
CPU	central processing unit
EE	execution environment
EPID	Intel Enhanced Privacy ID, a group signature cryptosystem
ETH	a unit of the Ether cryptocurrency
HMAC	hash-based message authentication code, a subfamily of MAC algorithms based on the use of a hash function
HTTP	HyperText Transfer Protocol, an Internet protocol
HTTPS	HTTP Secure, an extension of HTTP that uses a TLS channel
IKE	Internet Key Exchange, an Internet protocol used in IPSec
MAC	message authentication code, a family of symmetric authentication algorithms
OTR	Off-the-Record, a messaging protocol
QE	quoting enclave, a component of remote attestations in SGX
REE	rich execution environment
RPC	remote procedure call
RSA	Rivest–Shamir–Adleman, a public-key cryptosystem used for encryption and digital signatures
SEV	AMD Secure Encrypted Virtualization
SHA	Secure Hash Algorithms, a family of cryptographic hash functions
SIGMA	from 'SIGn-and-MAc', a family of protocols for authenticated Diffie-Hellman agreements
SGX	Intel Software Guard Extensions
TEE	trusted execution environment
TLS	Transport Layer Security, a cryptographic protocol for establishing a secure channel
TPM	Trusted Platform Module (ISO/IEC 11889), a standard for a secure cryptoprocessor
USD	United States dollar
X3DH	Extended Triple Diffie-Hellman, a key agreement protocol used in the Signal protocol

1

Introduction

Communication protocols abound in our everyday life, both in their analogue and digital forms. When we complete a tax return form, we are filling in a document that reports our taxable positions according to the laws and regulations of a certain jurisdiction. Similarly, when we make an online transfer, we are communicating in real time with our bank by following certain conventions and ultimately instructing the bank to effect a transfer. Communication protocols can similarly model lower-level interactions, such as a person-to-person conversation, epistolary communication, a conversation over the telephone or a conversation taking place over e-mail or a chat application.

A protocol in this sense can be described as a series of messages that are exchanged for successful communication to take place. The nature, order and content of these steps define the resulting protocol properties, one of which is whether it is (non-)repudiable. Repudiation refers to the act of a party to disavow their part in a communication [2]. A (plausibly) repudiable protocol is then one in which parties can (plausibly) dispute their involvement, and, by contrast, a non-repudiable protocol is one in which this is not feasible. For example, a communication protocol consisting of writing messages (such as 'Lunch at noon') on a chalkboard in a public room is generally repudiable because forgeries are trivial —anyone may enter the room and write new messages or change existing ones. By contrast, a protocol based on archived notarised documents for messages can be considered non-repudiable; in digital communications, digital signatures provide non-repudiation by linking a signer to a message.

The main focus of this thesis is on the effect that execution environments have on plausible deniability. Deniability is a concept intimately related to repudiation; we say that a protocol offers deniability if its execution can be plausibly repudiated: its execution does not result in evidence of the involvement of its executors nor of the particular contents of any messages exchanged. An execution environment, in the context of a protocol, is the set of external factors that condition the protocol result.

Some messages are highly non-repudiable by design, such as notarised documents or those documenting court proceedings; as a result, they offer little to no deniability. However, other messages that may be plausibly repudiated in some environments can be non-repudiable in other environments. For example, messages written on a chalkboard protocol are repudiable

under the assumption that any person could walk into the room and change the messages in any way they wish, including writing new messages, changing existing messages or even erasing them, and that therefore it is impossible to determine the authenticity or authorship of any messages. However, if we mention that a trusted third party (such as a notary or justice of the peace) is present in the room at all times and intently records what is being written on the chalkboard and by whom, any deniability offered in the messages-on-a-chalkboard protocol is compromised because those records can be considered non-repudiable.

The concept of data authentication is closely related to the property of deniability. Data authentication refers to associating an *authentication tag* to a piece of data that can later be verified to ascertain *integrity*, that is, that data were not tampered with, and *provenance*, that is, who produced the data. In the case of a tax return, integrity may be established by observing that its envelope is immaculate whereas provenance may be established by assessing the presence of a signature and successfully comparing it to a known authorised signature for the person allegedly filing them. The successful verification of an authentication tag that can establish both integrity and provenance compromises deniability by providing evidence of involvement and the contents of the messages.

In the realm of digital communications, we can generally divide data into three categories based on how they are authenticated:

unauthenticated data As the name implies, these data do not have an authentication tag associated with them and can be readily tampered with; therefore, repudiation claims are plausible and offer strong deniability. They are comparable to messages on a public chalkboard.

asymmetrically authenticated data These data are protected against tampering by an authentication scheme that makes use of an asymmetric identifier with two components, a secret and a public one. An authentication tag is computed based on the secret identifier, and it can later be verified using only the public identifier. If the public identifier can be associated with a certain entity, and in so far as the secret remains protected, such a scheme is highly non-repudiable. These schemes are also known as digital signatures and offer little deniability if it is possible to determine the entity with access to the secret, such as with the aid of a public key infrastructure [3].

symmetrically authenticated data These data are protected against tampering by an authentication scheme that makes use of a secret to compute *and* verify data integrity. Because verification requires knowledge of a secret, and this secret can be used to generate forgeries, such a scheme is repudiable when a verifier cannot be trusted not to make forgeries. These schemes are also known as message authentication codes (MACs) and are generally building blocks for protocols provide both deniability and authentication, a combination of properties sometimes called *deniable authentication*.

Whether a verifier is trusted not to produce forged messages is a property of the execution environment the verifier resides in. If a verifier can be presumed to be incapable of forging messages, it is possible to alter the plausibility of repudiation claims by the sender of a message. Trusted Execution Environments, presented in § 2.2, can provide the conditions for building such verifiers: a verifier that, while in possession of all the information needed to produce

forgeries, can be trusted not to produce them. Remote attestation (q.v. § 2.2), a feature of TEEs, can further produce convincing evidence that a TEE was present and which may be verified by anyone. As a result, TEEs and remote attestation can be leveraged to compromise deniable authentication by turning them into non-repudiable authentication.

1.1 Motivation

Messaging applications are increasingly popular; the most popular instant messaging application, *WhatsApp*, has over 1.5 billion monthly active users (as of April 2018) [4]. *WhatsApp* implements a variant of the Signal protocol, which is designed to provide deniability.

Deniability does not require confidentiality; anonymous feedback forms are an example of a communication protocol that offers deniability but not confidentiality, and which allows their respondents to include information they may be unwilling to include if they were not anonymous. However, deniability can be a critical property of confidential communications because it complements privacy (i.e., that unintended recipients are unable to deduce the contents of a communication) by offering protection to a message sender against unintended information leakage, even if the unintended recipient learns the communication contents. Thus, deniability allows for having honest conversations without being concerned about the implications that the conversation may have if it is disclosed to an unintended recipient, and exchanging potentially sensitive or incriminating messages without deniability is dangerous.

As a contemporary example, consider the Podesta email dumps [5]. Many of the messages contained digital signatures, which were embedded as part of the anti-spam measures used in email. These signatures enabled third parties, who were not intended recipients, to verify that the emails and were indeed authentic. The contents of the emails were highly sensitive and might have played a role in the outcome of the US 2016 presidential elections [6]. Had this same information been exchanged using deniable protocols, it would have been impossible to verify their authenticity and perhaps their impact would have been reduced.

Deniable authentication seems like a paradoxical combination of two desirable properties. On the one hand, we want assurances that we are talking to our intended recipient; on the other hand, we may not want to produce evidence of the communication that third parties can verify. Achieving both requires producing evidence that is only useful to our intended recipient, in other words, that this evidence be non-transferrable. Intuitively, non-transferrability is not always possible, especially if we introduce a trusted third party that attests to what is being said.

Hardware capable of providing TEEs is available in commodity hardware, including a majority of mobile devices (with technologies such as ARM TrustZone) and desktop computers (with technologies such as AMD SEV, Intel SGX and TPMs), and can be used to fulfil the role of this trusted third party highly efficiently. While both deniable authentication and TEE and remote attestation applications are well-researched areas, the combination of the two does not appear to be.

1.2 Goals

The work in this thesis aims, as a primary goal, to explore how the availability of remote attestation affects deniability in communication protocols. This goal is considered both from the perspective of an adversary, wishing to compromise deniability, and protocol users, wishing to preserve it. For the adversarial goals, we first examine how an adversary may be able to use remote attestation to compromise deniability, followed by an analysis of practical aspects of such an attack against real-world protocols. For the user goals, we explore the countermeasures that may be implemented against attacks on deniability that leverage remote attestation and consider how remote attestation may be incorporated to provide deniability.

1.3 Structure

The remainder of this thesis is structured as follows:

- Chapter 2 presents background information on the concepts and technologies used and discussed in the rest of this document and chapter 3 presents a formal treatment of how deniability is impacted in trusted execution environments.
- Chapter 4 presents a practical attack on deniability in the Signal protocol.
- Chapter 5 discusses some possible defence mechanism against the attack in chapter 4 and similar attacks.
- Chapter 6 reviews the treatment of deniability and similar attacks in the literature, and chapter 7 discusses the implications of the availability of remote attestation for deniable communications.
- Finally, overall conclusions and suggestions for future work are presented in chapter 8.
- Appendices A and B document the TEE interface used in the attack (chapter 4) and defence (chapter 5) implementations, respectively.

2

Background

2.1 Authenticated Encryption

In cryptography, encryption refers to a process that provides *confidentiality*, meaning that the data are injectively scrambled in a way that the result is unintelligible; the specific map of the domain to the image is determined by a secret. Similarly, authentication provides *integrity*, meaning that modifications to the scrambled data can be detected. Authenticated encryption is hence encryption that provides both integrity and confidentiality [7]. Authenticated decryption refers to the unscrambling of authenticatedly encrypted data. The Galois-Counter mode of operation (GCM) [8] is an example of a cryptosystem providing authenticated encryption (AE). An extension of authenticated encryption is authenticated encryption with associated data (AEAD), which refers to the authentication being performed not only on the ciphertext, but also on some external, and not necessarily secret, data.

2.2 Execution Environments

An execution environment refers to the combination of hardware and software components that can be used to execute and support applications [9]. A typical execution environment consists of a processing unit, memory, input and output (I/O) ports and an operating system. Because application execution requires an execution environment, applications are ultimately limited by any constraints placed onto them by their execution environment. This means, for example, that applications can use only as much memory as their execution environment allows, and that they can only perform tasks supported and allowed by their execution environment.

Rich Execution Environments (REEs)

Traditionally, computing is effected in execution environments that not only permit the loading and execution of arbitrary programs but may also themselves be manipulated in arbitrary ways, such as by attaching a hardware debugger to observe and change the internal

state. As it is impossible for such environments to make any verifiable assertions as to their state, they are inherently untrustworthy. The term Rich Execution Environments (REEs) is sometimes used to refer to this type of execution environments, such as by GlobalPlatform in [10].

To understand the difficulty of establishing trust in REEs, consider the example of a messaging application that requires uploading a person's contacts to its developers' systems for the purpose of allowing that person to discover contacts already using said application. While this can be a valuable feature, the main disadvantage of such a system is that the users do not have a means of auditing or controlling how the information about their contacts will be used. While honest developers might use this information only for the stated purpose of contact discovery, unscrupulous developers might instead collect this information, aggregate it and then surreptitiously sell social graphs in the open market. Because at this point a sceptical user has only the word of the application developers that contact information is used for the stated purpose and only for the stated purpose, convincing prospective sceptical customers to upload their contacts requires demonstrating that their execution environment is such that the contacts uploaded cannot possibly be misused. However, providing such proof is impossible with an REE because a malicious developer could alter the execution environment in any number of ways such that contact information is exfiltrated.

For example, suppose that the application developers hire an independent, objective and trusted auditor to inspect their systems for compliance to their stated policy and that not only does the auditor find the systems compliant, but also inspects the systems on a regular basis to ensure ongoing compliance. If the developers had malicious intent, they could attempt to fool the auditor (and their users) in various ways. A crude approach could be changing their server configuration files after each audit, but more sophisticated techniques are possible, such as modifying the hardware in their servers to exfiltrate the desired information.

Furthermore, even if the developers *were* honest and trustworthy, the nature of REEs results in the contact-use policy still being difficult to enforce by technical means because other actors involved might still interfere in ways that result in the misuse of contact information, such as the developers' employees and suppliers. Some of these breaches could include an employee reading the server's memory directly, a malicious operating system transmitting data on user activity, or even an unauthorised third-party accessing the server and modifying the software being executed.

Trusted Execution Environments (TEEs)

TEEs address the inherent untrustworthiness of REEs by providing an isolated environment for user applications, the properties of which are well-known. Continuing the previous example of contact discovery, consider messaging application developers willing to go to great lengths to substantiate their contact use policy. They may accomplish this by limiting the execution environment to prevent arbitrary state manipulation and by providing a means of establishing a verifiable chain of trust. One of the ways they can accomplish this is to configure the contact discovery service and then hand it over to a trusted third party[1]. The trusted third party then proceeds to thoroughly audit the entire execution environment (including both hardware and

[1]A trusted third party in this context means an entity that is trusted to be honest by the application users.

software components) and publish the relevant findings. Then, the entire machine is secured[2] so that the execution environment cannot be modified further. When a user uploads their contact information to the contact discovery service, they also consult the trusted third party as to whether the service they are connecting to corresponds to the audited system.

Intuitively, this scheme provides a high degree of confidence that the system a user is connecting to corresponds to one that behaves as per the published audit findings providing the user trusts the auditor. However, the example described is not practical for several reasons. Firstly, the increasing complexity of computer software and hardware and the interactions between them make it extremely difficult for an auditor to be decisively confident that a system behaves in a certain way, as doing so would require evaluating each of the components that comprise the system. Secondly, the example relies on an idealised notion of securing a machine that is not practical; physical access to the machine may be needed to effect repairs and other maintenance tasks, logical access may be needed to perform software updates. Thirdly, because of the restrictions placed on the entire system, any modifications would require going over the impractical processes of a full audit and securing once again.

A more practical approach would be to isolate certain critical system components (such as software for contact discovery) from the rest of the system, and then audit these critical components. Hardware manufacturers are in a unique position to provide such isolated execution environments that allow users to restrict the interactions between sensitive components from the rest of the system, and that facilitate auditing by providing just enough functionality for implementing these critical operations [11]. In GlobalPlaform terminology, these environments are known as Trusted Execution Environments (TEEs) [10] and are defined as follows in [12]:

> [A] Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored on a persistent memory. In addition, it shall be able to provide a remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting backdoor security flaws are not possible.

TEEs have numerous privacy-enhancing applications that may benefit users. One of them is, as discussed earlier, private contact discovery; the *Signal* application uses a contact discovery service enhanced using Intel SGX, a TEE technology, to protect its users' privacy [13]. A similar application of TEEs is performing malware analysis in a remote cloud service, so that the service may not identify users by the contents of their devices, such as the applications they have installed [14], especially important as 98.93 % of users may be uniquely identified by the list of applications they have installed [15].

[2] Securing the machine refers in this case to applying a combination of physical and technical tamper-evident seals such that any modifications to the execution environment be noticed if not completely prevented. As an example, administrator-level software access could be disabled, the entire software stack could be placed in read-only memory and the entire machine could be encased in concrete and then shipped to a secure location.

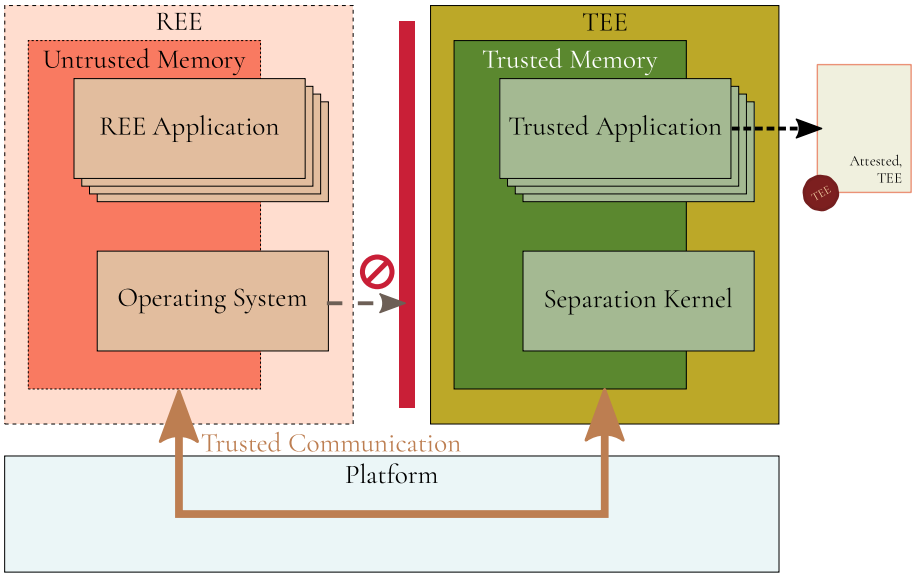

Figure 2.1: An overview of some of the components of a trusted platform. On the left, REE applications are executing on top of an untrusted operating system and sharing untrusted memory. Any privileged process in a TEE is able to manipulate the state and memory of REE applications. On the right, trusted applications are being executed in a trusted memory region. The platform and the isolation kernel protect access to the trusted memory and state of the trusted applications, which is protected from unauthorised access, even by privileged processes such as the untrusted operating system; any communication between the trusted and the untrusted parts is executed through a designated communication policy enforced by the platform. Lastly, trusted applications may produce attestations to prove their identity and trustworthiness.

Sealing and Unsealing

In the context of TEEs, sealing and unsealing refer to cryptographic primitives providing authenticated encryption and decryption, respectively, with a secret that exists *only* within the TEE to securely store some of the TEE state in untrusted storage. Some TEE platforms, such as Intel SGX, provide a mechanism for deriving sealing secrets deterministically based on the TEE identity, which allow for unsealing information in different instances of the same TEE. Sealing can be useful to reduce the state information that is needed within a TEE, such as when the TEE is memory-constrained or for persistence when the TEE state may be lost (such as due to a power failure). A high-level overview of the sealing primitive is shown in fig. 2.2.

Attestation

A crucial component of TEEs is their providing a protocol to provide *attestations*, verifiable assertions about the state or properties of the TEE; in particular, they can prove the presence of a particular TEE. Note that attestations are indispensable for TEEs, as without them it is impossible for an observer, including local observers with full access to a system, to differentiate between a TEE and another execution environment, whether it be another TEE

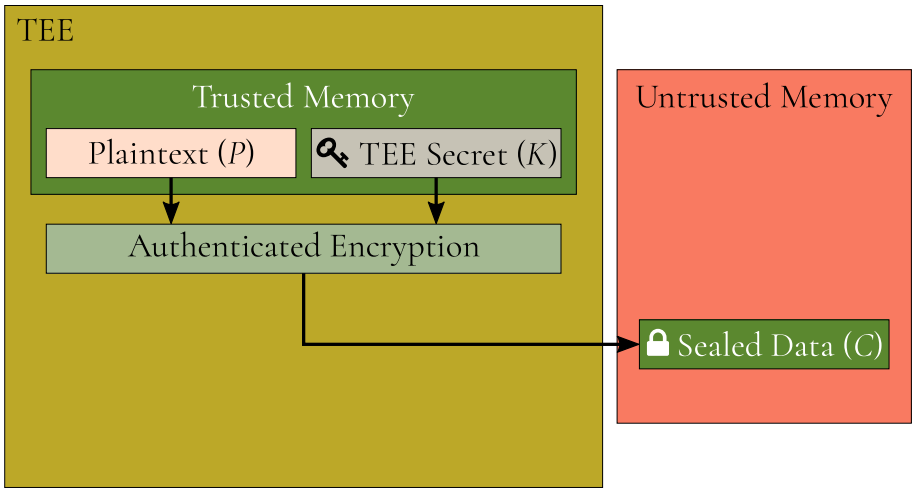

Figure 2.2: High-level overview of the concept of *sealing*. A plaintext P is authenticatedly encrypted using a secret K that exists only within the TEE. This produces a sealed datum C, which can be safely stored in untrusted memory.

or an REE. Hence, although it is indeed possible to use TEEs without using attestations (for example, for their isolation properties), they provide weaker security guarantees.

The trusted part of a TEE stems from the concept of a *root of trust*, that is, the notion that TEEs are trusted because some entity makes assertions about the properties of the TEE, and that these assertions are believed because the entities verifying the assertion trust the one making them. Trust in attestations entails both trusting the processing environment used in the TEE to meet certain specifications (a static component) and the specific state or application in the TEE (a semi-dynamic component) [12].

Attestation Example Although various attestation schemes are possible, it may be helpful to consider a simple attestation model to understand the different parts that make an attestation. A TEE manufacturer produces TEE hardware in which each device δ is assigned a unique per-device asymmetric key pair $\left(\delta_{pk}, \delta_{sk}\right)$ stored in the device. The public part of this key, δ_{pk}, is signed with the device manufacturer's key pair $\left(M_{pk}, M_{sk}\right)$ to produce a signature σ_δ. When a TEE is initialised, the secret key (δ_{sk}) is used to sign the TEE state S[3] to produce a signature σ_S. An attestation could consist of the tuple $\left(M_{pk}, \sigma_\delta, \delta_{pk}, \sigma_S, S\right)$. This would allow anyone to verify that the state S was present in the TEE δ. In particular, σ_δ establishes trust on the device δ because the manufacturer is trusted; then, the trust on the presence of state S is possible because the device δ is trusted; lastly, if the state S corresponds to a well-known and application, the entire TEE can be trusted to execute this application according to the device specifications.

[3]For the purposes of this example, the state S may be assumed to contain a nonce, a timestamp or some other similar measure that assures freshness.

2.3 Intel Software Guard Extensions

Intel Software Guard Extensions (SGX) is an extension to the Intel architecture, available since the Skylake microarchitecture, that provides a TEE for user software in the form of isolated software containers known as *enclaves* [16]. SGX has the stated goal of reducing application attack surface [17] and protecting applications from potentially malicious privileged code, such as the operating system and a hypervisor [18]. An SGX enclave is a protected region within the memory space of an application, which after initialisation can only be accessed by software resident in the enclave. This section provides background information on the operation of SGX enclaves based on [16] except where noted.

Enclave Signing Each SGX enclave is required to be signed with a 3072 bit RSA key, along with including identifiers of the enclave program and its version. The public RSA key is used to identify the enclave developer, and a 256 bit identifier is derived from *measuring* this public key, which is called MRSIGNER. By signing several enclaves with the same key, some convenience features are available, such as the ability to derive common sealing keys for exchanging information between enclaves made by the same developer.

Debug and Production Modes SGX enclaves exist in two flavours, which are determined at build time: debug and production enclaves. These two behave identically, except that external processes may use the `EDBGRD` and `EDBGWR` instructions for reading from and writing to debug enclave memory, and may similarly set breakpoints and traps within debug enclaves. As a result, the state of debug enclaves can be arbitrarily compromised and they are *not* TEEs. While any SGX-capable processor may instantiate debug enclaves, only enclaves signed with a key whitelisted by Intel may be executed in production mode [19].

Enclave Identity The initial enclave state is *measured* upon initialisation by applying a hash function on every memory assigned to the enclave, which includes its code and data. This providing an enclave identity called MRENCLAVE, which is represented as a 256 bit value. This identity, along with MRSIGNER, represents the enclave identity. Both measurements may be used to derive enclave- or signer-specific keys (for example, using the `EGETKEY` instruction) that depend on a processor-unique secret.

Execution Model

An enclave may be accessed by using the `EENTER` instruction, which must point to a well-known entry point in the enclave, each of which is called an *ecall* [20]. Once inside an enclave, the enclave may transfer control back to the calling process by using the `EEXIT` instruction or its execution may be interrupted by an Asynchronous Exit Event (AEX), caused by external events (such as interrupts) that occur while an enclave is being executed. If an enclave execution is executed as a result of an AEX, the enclave execution may be resumed using the `ERESUME` instruction. An enclave may call `EEXIT` to return once it has finished its current task, or to perform an *ocall* [20], that is, to execute code outside the enclave, such as a system call. In the latter case, the function performing the *ocall* may not have finished executing and may expect the called code to return by following some convention to call back into the enclave.

Memory Model

The processor manages access to any memory pages assigned to SGX enclaves (referred to as *trusted memory*) and triggers page faults for unauthorised accesses, and any memory pages swapped to disk are sealed to provide confidentiality and integrity, even from privileged processes. By contrast, memory pages not assigned to enclaves (referred to as *untrusted memory*), do not offer these protections any may be arbitrarily manipulated by the operating system or other processes.

Attestation Model

SGX provides *local* attestations, which depend upon a processor-unique secret and can only be verified by enclaves executing on the same processor, and *remote* attestations, which can be verified by anyone, and do not require an enclave for verification.

Local Attestations

Any SGX enclave may produce a report that can be verified by other enclaves executed on the same processor, referred to as *local attestations*. These attestations enable secure information exchange between enclaves, and may be used both for synchronous communication (e.g., for establishing a secure channel between enclaves) and asynchronous communication (e.g., for storing information that may be later used by another enclave). The local attestation process is summarised in fig. 2.3.

Generation Local attestations are generated using the `EREPORT` instruction, which takes two inputs, TARGETINFO and REPORTDATA, and produces a *report* as an output. TARGETINFO identifies the enclave intended to verify the generated report and includes the target enclave MRENCLAVE as well as other attributes, such as whether it is a debug enclave. REPORTDATA is a 512 bit data structure that may contain arbitrary information intended to be verified by the target enclave. The generated report is authenticated with a symmetric key derived from the provided target MRENCLAVE and contains REPORTDATA and information about the attesting enclave, including MRENCLAVE, MRSIGNER and other attributes, such as whether it was a debug enclave.

Verification Local attestations can only be verified by the target enclave, which must be executed in the same processor that was used for generating the report. The verification process consists of deriving a report key using the `EGETKEY` instruction and verifying that the MAC attached to the report is valid.

Remote Attestations

In addition to local attestations, SGX supports the generation of *remote attestations*. Unlike local attestations, verifying remote attestations does not require the use of an enclave nor is it limited to the same processor.

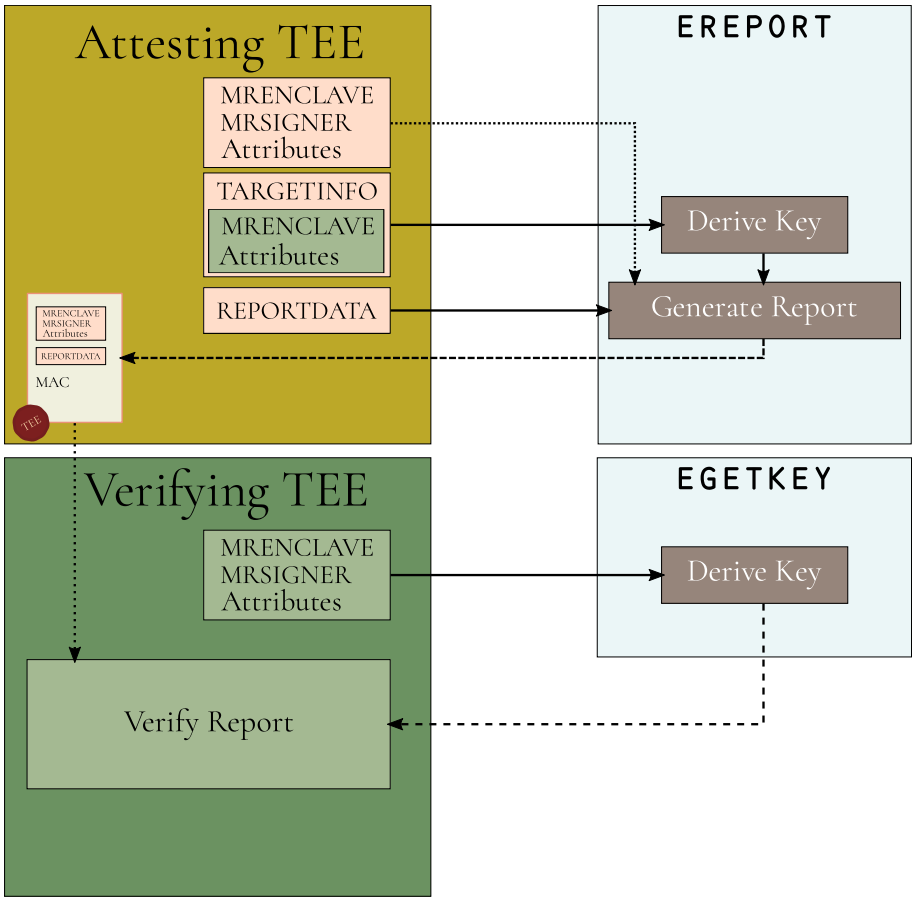

Figure 2.3: Overview of the SGX local attestation generation and verification processes. An attesting enclave generates a report for a verifying enclave that contains some REPORTDATA using the `EREPORT` instruction. The verifying enclave generates a report verification key using the `EGETKEY` instruction and proceeds to verify the report. The report includes REPORTDATA and information about the attesting enclave.

Generation The process for generating a remote attestation is similar to that for local attestations but involves some additional steps. A remote attestation begins with a report made for an Intel-provisioned quoting enclave (QE) [18]. The quoting enclave then receives and verifies the report, and converts the report into a *quote*, which is signed with the QE private key [21].

The quoting enclave (QE) is provisioned by Intel as part of the Intel SGX Platform Software (PSW) [22]. The QE is tasked with verifying a local attestation report (described earlier) and using its public key to sign the report, which can be verified later. The signature uses a group signature scheme called Intel Enhanced Privacy ID (EPID) to ensure that a signature cannot be linked to a particular signer, but just to a member of the group [21].

Verification Remote attestation quotes may be verified by validating whether the signature they contain is valid, which can be done by using either an EPID public key certificate or an attestation verification service [21], such as the Intel Attestation Service (IAS) [23].

2.4 Plausible Deniability

The ISO non-repudiation framework states that the goal of non-repudiation is 'to collect, maintain, make available and validate irrefutable evidence concerning a claimed event or action in order to resolve disputes about the occurrence or non-occurrence of the event or action' ([24]).

Plausible deniability is a concept closely related to *non-repudiation* in that it has the opposite stated goal: it is desired that any evidence regarding a disputed event be plausibly refutable. In particular, a protocol executed between two entities, Alice and Bob, and observed by an external observer Eve is plausibly deniable if its execution does not generate any evidence that can be used to irrefutably demonstrate Alice's or Bob's involvement [25]. Note that this definition permits mutual authentication of Alice and Bob insofar as this mutual authentication is non-transferrable (i.e., it has no meaning for a third party).

Following [26], [27], we can model deniability in the paradigm of simulation. For this, we assume an arbitrary protocol π in which there is a sender A, a receiver B, an informant E who observes the protocol execution, a misinformer M who does not observe the protocol run but executes a simulated protocol π^* and a judge J who will rule at the end of the protocol whether communication took place between A and B. The protocol π is deniable if J is not able to distinguish between E and M.

Authentication

Authentication was presented earlier in the context of authenticated encryption as a cryptographic process conferring *integrity*. In the context of communications, authentication has a broader meaning that encompasses the entire communication, including *provenance*. Generally speaking, for a message $\mathcal{M}$, the authenticator $\mathcal{A}$ allows a verifier V to assess not only whether the message $\mathcal{M}$ was tampered with but also whether it originated from the expected party, that is that $\mathcal{M}$ was sent by some party A and not from an unauthorised party C. Although authentication is useful to verify that messages are genuine, the way that $\mathcal{M}$ and $\mathcal{A}$ are tied together, possibly to a sender A, may reveal information to parties external to the protocol as to A's involvement.

Working in the deniability framework set forth above, authentication is deniable if it does not give the informant E any advantage over the misinformant M in convincing the judge. We will examine three different authentication schemes and see how deniability is affected.

Scenario 1: No authentication In this case, the protocol π provides no authentication. Note that it is impossible for the judge to tell which messages are genuine (meaning that the misinformant is able to forge messages that are indistinguishable from any real messages sent by the sender). This protocol is therefore deniable.

Scenario 2: Digital signatures In this case, the protocol π uses digital signatures for authentication. The sender has a key pair $\left(I_A, g^{I_A}\right)$, where g^{I_A} is used for verification and is public and I_A is used for generating an authenticator and is a secret known only by the sender. Assuming that generating a valid authenticator without knowledge of I_A is infeasible, the

judge can now distinguish the informant from the misinformant because the latter is unable to furnish messages that validate correctly.

Scenario 3: Symmetric authentication In this case, the protocol π uses symmetric authentication, such as that provided by AES-GCM. The sender and the receiver agree on a key K and the sender transmits messages encrypted and authenticated under this key, which are verified and decrypted by the receiver using the same key. Note that because K is a shared secret, the receiver (or anyone who knows K) can forge messages that are indistinguishable from genuine messages from the sender. Because forgeries are possible, the judge cannot distinguish the informant from the misinformant.

2.5 Signal

Signal is a free instant messaging application made by Open Whisper Systems that provides end-to-end encrypted messaging [28]. The Signal application uses the Signal *protocol* which is based on the Curve25519 [29], AES-256 [30] and HMAC-SHA-256 [31], [32] primitives and features 'enhanced deniability properties' [33] as a result of the X3DH handshake protocol [34], [35]. The Signal protocol is based on four sub-protocols: XEdDSA and VXEdDSA (elliptic curve digital signature schemes), X3DH [36] (key agreement), Double Ratchet [34] (message exchange) and Sesame (session management) [37].

Even though the Signal protocol provides end-to-end messaging, the *Signal* application does not use peer-to-peer technology to enable decentralised or direct communication amongst users. Instead, it acts as a client to the Signal service, which routes messages from and to users and provides some other necessary infrastructure, such as registration, key distribution, contact discovery and asynchronous delivery.

Deniability in Signal

The Signal protocol claims 'deniability properties', which are explained as 'X3DH doesn't give either Alice or Bob a publishable cryptographic proof of the contents of their communication or the fact that they communicated' [36]. This notion of deniability is consistent with the one discussed in § 2.4, but does not apply to the entire protocol; doing so requires expanding the notion of deniability thus: '*The Signal protocol* doesn't give either Alice or Bob a publishable proof of the contents of their communication or the fact that they communicated.'. The adversarial goal of breaking deniability is to make the protocol non-repudiable; that is, to obtain publishable convincing proof of either communication contents or communication occurrence. Achieving this goal requires an authentication tag that can be cryptographically verified.

Understanding why Signal claims to offer deniability requires understanding first how conversation messages are authenticated, which involves the X3DH handshake protocol and the Double Ratchet message exchange. The Double Ratchet protocol uses unique per-message symmetric keys derived from two 'ratchet' algorithms: a symmetric ratchet, which derives keys from previous ratchet keys and a message counter, and an asymmetric Diffie-Hellman ratchet, which derives keys based on the Diffie-Hellman agreement [38] using ephemeral asymmetric

keys[4]. The messages themselves are encrypted with authenticated encryption using the derived per-message keys. In addition, the Double Ratchet is initialised using a shared secret, which in the case of the Signal protocol is the result of an X3DH exchange. Establishing deniability hence requires that the protocol be deniable in every possible state, namely: at initialisation, after performing a symmetric ratchet and after performing an asymmetric ratchet. Intuitively, if the X3DH stage is assumed to be deniable, the entire protocol is deniable because only ephemeral keys, which cannot be linked to any particular party, are used in the Double Ratchet protocol.

X3DH

The X3DH protocol is a key agreement protocol based on the Diffie-Hellman agreement, which consists of three or four Diffie-Hellman handshakes. For some party X, let I_X denote their long-term identity key, which is well-known and can be linked to X; E_X, their ephemeral key, freshly generated for each protocol run; S_X, their signed pre-key, a medium-term key that is signed by the long-term identity key I_X; and P_X their one-time pre-key, a key that is generated in advance but used at most once[5].

Furthermore, let $g^K \in \mathbb{F}$ represent the public component of a key K, $\left(g^K\right)^L \in \mathbb{F}$, the Diffie-Hellman agreement between keys K and L in a field $\mathbb{F}$. A key-derivation function $\mathbf{H} : \mathbb{F}^n \to \mathfrak{H}$ takes n Diffie-Hellman key agreements and produces a derived key in a space $\mathfrak{H}$.

Suppose that Alice (A) wishes to communicate with Bob (B) using a post-X3DH protocol π. The X3DH protocol is then executed as follows:

1. Alice computes $K = \mathbf{H}\left(\left(g^{S_B}\right)^{I_A}, \left(g^{I_B}\right)^{E_A}, \left(g^{S_B}\right)^{E_A}, \left(g^{P_B}\right)^{E_A}\right)$ [6].
2. Alice sends Bob a message $\mathcal{M}$ containing g^{I_A}, g^{E_A}, g^{S_B}, g^{P_B} and an initial message $\mathcal{M}_0$ encrypted using an authenticated encryption scheme and with a key derived from K.
3. Bob uses $\mathcal{M}$ to reconstruct K'.
4. Bob attempts to decrypt and verify $\mathcal{M}_0$ using key K'. If successful, $K = K'$ and communication between Alice and Bob may continue under protocol π and key K.

The X3DH handshake provides perfect forward secrecy (i.e., the compromise of long-term keys (viz., I_A and I_B) does not compromise the confidentiality of past communications [39, p. 496]) from the use of one-time ephemeral keys (the $\left(g^{P_B}\right)^{E_A}$ part of the handshake) and the medium-term signed pre-key S_B (the $\left(g^{S_B}\right)^{E_A}$ part of the handshake). It also provides authentication (i.e., assurance that Alice and Bob are talking to each other) by deriving a secret that requires oracle access to the identity keys I_A and I_B, as incorporated in the $\left(g^{S_B}\right)^{I_A}$ and $\left(g^{I_B}\right)^{E_A}$ handshakes.

[4] In this context, ephemeral keys are freshly generated asymmetric key pairs which are discarded after use.

[5] Both the one-time pre-key and the ephemeral are used at most in one protocol run. They differ in that the one-time pre-key is generated before the protocol is run, thus facilitating non-interactive session initiation.

[6] Or alternatively, $K = \mathbf{H}\left(\left(g^{S_B}\right)^{I_A}, \left(g^{I_B}\right)^{E_A}, \left(g^{S_B}\right)^{E_A}\right)$. This variant results in weaker perfect forward secrecy.

Double Ratchet

The Double Ratchet algorithm is a message exchange protocol in which each message is encrypted and authenticated using unique keys derived from a series of chains. Without taking into account the handling of out-of-order or skipped messages, the protocol can be summarised as follows:

1. Alice initialises the protocol with a shared secret K from executing the X3DH protocol.
2. Alice generates a fresh key pair $\left(R_A, g^{R_A}\right)$.
3. Alice initialises her root key RK_A to the value of K and then computes a root key $RK_A = \mathbf{H}_1\left(RK_A, \left(\left(g^{R_B}\right)^{R_A}\right)\right)$, where g^{R_B} is Bob's public ratchet key and $\mathbf{H}_1$ is some key derivation function.
4. Alice derives a sending chain key $CKs_A = \mathbf{H}_2\left(RK_A\right)$, where $\mathbf{H}_2$ is some key derivation function.
5. Alice derives a message key $MK = \mathbf{H}_3\left(CKs_A, \mathcal{C}\right)$, where $\mathcal{C}$ is a message counter and $\mathbf{H}_3$ and some key derivation function.
6. Alice sends the first message $\mathcal{M}_A$ to Bob. The message includes g^{R_A}.
7. Subsequent messages $\mathcal{M}_A$ sent by Alice use keys derived from CKs_A.
8. When Alice receives a message $\mathcal{M}_B$ from Bob, she checks whether the public key used by Bob matches the current value of R_B. If it does not, she updates this value and then re-calculates RK_A, computes a receiving key chain CKr_A similarly to how CKs_A was computed and then generates a fresh key pair $\left(R_A, g^{R_A}\right)$ and re-calculates CKs_A.
9. Alice derives a message key from CKr_A and a message counter and uses it to decrypt $\mathcal{M}_B$.
10. The steps from 4 on are repeated with each message in either direction, recomputing the root key and chain keys.

The Double Ratchet protocol provides *post-compromise security* [40], [41], sometimes also called *future secrecy* or *self-healing*, which means that two parties executing the protocol have security guarantees even if the other party protocol secret state has been compromised. In Signal in particular, this means that if a passive attacker that is able to compromise a session at some point in time will not compromise future states of the protocol because Alice's and Bob's ratchet keys (R_A and R_A, respectively) are regularly changing.

2.6 Zero-Knowledge Proofs

Zero-knowledge proofs can be informally described as schemes in which a prover P proves to a verifier V that they know some information I without revealing any additional information about I. For example, consider the case in which P wishes to prove to V that they know the solution to the discrete logarithm problem in a finite cyclic group $\mathcal{G}$; that is, V knows the exponent x such that $x = \log_\alpha \beta$ for some $\alpha, \beta \in \mathcal{G}$, and wishes to prove this knowledge to P without revealing x [42]. This may be done as follows, where φ denotes Euler's totient function:

1. P shares α and β with V.
2. P picks a random $r \in \mathbb{Z}_{\varphi(\mathrm{ord}(\mathcal{G}))}$, computes $\gamma = \alpha^r$ and sends γ to V.
3. V picks a random challenge $b \in \mathbb{B}$ and sends b to P.
4. P computes $y \equiv r + bx \pmod{\varphi(\mathrm{ord}(\mathcal{G}))}$ and sends y to P.
5. V checks whether $\alpha^y = \gamma\beta^b$.
6. Steps 2-5 are repeated N times.

In this scheme, it is possible at any iteration for P to lie about knowing x. If P believes (or guesses) V will provide $b = 0$, P behaves exactly as described and reveals $y \equiv r \pmod{\varphi(\mathrm{ord}(\mathcal{G}))}$ in step 4; V verifies that $\alpha^r = \gamma$. Alternatively, if P believes (or guesses) V will provide $b = 1$ in step 2, P can pick $r' \in \mathbb{Z}_{\varphi(\mathrm{ord}(\mathcal{G}))}$, compute $\gamma' = \alpha^{r'}\beta^{-1}$ and provide γ' in lieu of γ and then r' in lieu of y; V verifies that $\alpha^{r'} = \alpha^{r'}\beta^{-1}\beta$. Assuming that V chooses values for b uniformly at random, at any iteration P has a 1/2 chance of guessing V's challenge correctly and succeeding at lying about knowing x. Hence, after N iterations, the chance of P not knowing x but correctly responding to all of V's challenges is $1/2^N$, a less than 1-in-10 000 chance with 14 iterations.

The protocol described above is a zero-knowledge proof because, at any iteration, the only thing that V learns is that it is more likely that P does indeed know x, but otherwise V does not learn additional information about x. Furthermore, the knowledge that V gains about P knowing x is non-transferrable. An observer E to the interaction between P and V cannot determine whether V and P have colluded by agreeing to a certain sequence of values for b. An external verifier who did not observe the exchange does not know whether a transcript of the exchange is forged.

3

Effect of Execution Environments on Deniability

3.1 Introduction

This chapter describes how different execution environments can affect protocol deniability by constructing undeniable protocols by composition and then showing how a defence against such undeniable protocols may be achieved. The main implication is that achieving deniability against a TEE-capable adversary requires that an attestation not reveal any additional information about the protocol being executed to the judge. We claim that, in general, deniable key exchanges and deniable authentication, such as the schemes presented in [27] are impossible in the presence of arbitrary TEEs, but that deniable key exchanges and deniable authentication are possible if the presence of a TEE is known *a priori*. For a more rigorous treatment in the universal composability framework [43], refer to [1].

3.2 Compromising Deniability

Definition 1. *A protocol can be modelled as a Mealy machine [44], [45], [46] with a predefined initial state that, given its current state and a tuple of inputs produces a tuple of outputs and advances its state. The notation* $\pi : \mathfrak{S} \times \mathfrak{X} \times \mathfrak{I} \rightarrow \mathfrak{S} \times \mathfrak{O}$ *is used to denote a protocol* π *that, for a space of protocol states* $\mathfrak{S}$*, an input space* $(\mathfrak{X}, \mathfrak{I})$ *and an output space* $\mathfrak{O}$*, produces an output in* $O \in \mathfrak{O}$ *from applying an output function* $\mathbf{O} : \mathfrak{S} \times \mathfrak{X} \times \mathfrak{I} \rightarrow \mathfrak{O}$ *and a state* $S' \in \mathfrak{S}$ *from applying a transition function* $\mathbf{T} : \mathfrak{S} \times \mathfrak{X} \times \mathfrak{I} \rightarrow \mathfrak{S}$*.*

Definition 2. *A protocol operation, denoted as* $X \in \mathfrak{X}$*, is a high-level concept that describes the nature of the transformation applied by the output function* $\mathbf{O}$ *to the operation arguments* $I \in \mathfrak{I}$*. In a communication protocol, some of these operations may be* `send`*, to send a message, and* `recv`*, to receive a message.*

To understand how execution environments affect deniability, consider a protocol $\pi : \mathfrak{S} \times \mathfrak{X} \times \mathfrak{I} \to \mathfrak{S} \times \mathfrak{O}$, where $\mathfrak{S}$ is the space of all possible protocol states, $\mathfrak{X}$ is the space of all valid operations, $\mathfrak{I}$ is the space of all possible operation arguments and $\mathfrak{O}$ is the space of all possible operation outputs. If π is a communication protocol used for sending messages between Alice and Bob, only the protocol outputs in $\mathfrak{O}$ are exchanged between them, and the states in $\mathfrak{S}$ are kept internally by each party. For an operation $\texttt{send} \in \mathfrak{X}$, which takes a message $I_A \in \mathfrak{I}$ to be delivered from Alice to Bob as an input and produces an output $O_A \in \mathfrak{O}$, there is a corresponding operation $\texttt{recv} \in \mathfrak{X}$, executed by Bob, which takes an input $I_B \in \mathfrak{I}$ based O_A and produces a corresponding output $O_B \in \mathfrak{O}$ which includes information about the message I_A.

Let $\mathcal{T}_E = ((S, X, I), (S', O))$ be the resulting transcript from executing π. If π is a deniable protocol, a judge cannot distinguish $\mathcal{T}$ from some other transcript $\mathcal{T}^* \neq \mathcal{T}$ generated by a misinformant who did not witness this particular execution of π but executes a simulated protocol π^*; that is, a judge cannot learn any more information from $\mathcal{T}_E$ than she would from $\mathcal{T}_M$.

Consider now the protocol $\tau_A : \mathfrak{S}_{\tau_A} \times \mathfrak{X} \times \mathfrak{I} \to \mathfrak{S}_{\tau_A} \times \mathfrak{O} \times \mathfrak{V} \times \Sigma$, where $\mathfrak{S}_{\tau_A}$ is the space of all possible states for τ_A and Σ is the space of all possible signatures. Additionally, $\mathbf{M} : \mathfrak{S}_{\tau_A} \twoheadrightarrow \mathfrak{S}$ maps $\mathfrak{S}_{\tau_A}$ onto $\mathfrak{S}$ and $\mathfrak{V}$ is the space of all the information that may be verified by executing π. τ_A executes in a TEE, and as a result, it can produce remote attestations and the states in $\mathfrak{S}_{\tau_A}$ cannot be arbitrarily manipulated. It behaves as follows given X, S_{τ_A}, I:

1. Compute $(S', O) = \pi\left(X, \mathbf{M}\left(S_{\tau_A}\right), I\right)$.
2. Compute $S'_{\tau_A} \in \mathfrak{S}_{\tau_A}$.
3. Compute V from (X, S, I, S', O).
4. Produce an attestation $\sigma_A \in \Sigma$ to (τ_A, X, I, O, V).
5. Return $\left(S'_{\tau_A}, O, V, \sigma_A\right)$.

A transcript $\mathcal{T}_{\tau_A}$ from executing τ_A would contain $((X, I), (O, V, \sigma_A))$. If a protocol offers deniable authentication, when Alice executes π to communicate with Bob, Alice would verify Bob's identity, and therefore V in $\mathcal{T}_{\tau_A}$ contains this fact. A judge can therefore learn from $\mathcal{T}^*_{\tau_A}$ that it resulted from a genuine execution of τ_A, which executes π honestly, and that a conversation took place between Alice and Bob, that operation X was performed with input I and output O, and that this information can be trusted by verifying σ_A. A transcript from a misinformant who did not witness the execution of τ_A is unable to produce a valid transcript with all of this information if σ_A is assumed to be unforgeable because the attestation cannot be simulated by executing a protocol $\tau_A{}^*$. The protocol τ_A is also indistinguishable from π in a communication session because the output produced is generated by executing π as a subprotocol, and is therefore indistinguishable from a regular execution of π.

The attack presented in chapter 4 follows this concept for compromising deniability in Signal. The Signal protocol is executed in a TEE and an attestation is produced to its inputs and outputs as well as the verified public identities of the executors. It is important to note that compromising deniability in this manner relies on two properties of TEEs: isolation and

attestation unforgeability. Isolation is used for protecting the protocol states in $\mathfrak{S}_{\tau_A}$ from external changes and reads, which ensures the honest and correct execution of the protocol and attestation unforgeability is used for allowing external verifiers, such as the judge J, to verify that the transcript produced corresponds to an execution of τ_A and not some other protocol.

3.3 Restoring Deniability

As the approach described in the previous section may be used for any protocol that can be executed within a TEE, it is impossible to develop countermeasures that work against arbitrary TEEs. It is, however, possible to implement effective countermeasures if the adversarial TEE capabilities are known. In particular, we will model TEEs under the assumption that information that originates from outside the TEE does not have any of the isolation guarantees of a TEE.

Consider the protocol $\pi_D : \mathfrak{S}_D \times \mathfrak{X} \times \mathfrak{I} \to \mathfrak{S}_D \times \mathfrak{O} \times \mathfrak{V} \times \Sigma$ which implements TEE-resistant deniability under this model, where $\mathfrak{S}_D$ is the space of possible protocol states for π_D, with the property that it exists in a memory region untrusted by the TEE and that there is a one to one correspondence between $\mathfrak{S}_D$ and $\mathfrak{S}$. The protocol π_D furthermore executes a subprotocol $\tau_D : \mathfrak{S}_D \times \mathfrak{X} \times \mathfrak{I} \times \mathfrak{O} \to \Sigma$.

The protocol τ_D is defined as follows for some (X, S_D, I, O):

1. Compute $(S', O') = \pi(X, M(S_D), I)$.
2. If $O' \neq O$, terminate.
3. Produce an attestation $\sigma_D \in \Sigma$ to (τ_D, X, O).
4. Return σ_D.

The protocol π_D is defined as follows for some (X, S_D, I):

1. Compute $\left(O, S'_D\right) = \pi(X, M(S_D), I)$.
2. Compute $\sigma_D = \tau_D(X, M(S_D), I, O)$.
3. Return $\left(S'_D, O, \sigma_D\right)$.

The attestation produced by π_D ensures that any protocol output was the result of executing the protocol so that the state S_D is available in memory that a TEE cannot trust, and any protocol simulator for π which can produce forged transcripts can therefore trivially simulate π_D by generating incorporating the execution of τ_D in item 2. As a result, a judge who cannot distinguish between $\mathcal{T}$ and $\mathcal{T}^*$ also cannot distinguish between $\mathcal{T}_D$ and $\mathcal{T}_D^*$, where $\mathcal{T}_D$ is a transcript resulting from the execution of π_D and $\mathcal{T}_D^*$ is one resulting from executing a protocol π_D^* that simulates π_D.

Furthermore, it is not possible given the assumptions made on TEEs to similarly adapt τ_A to produce an undeniable transcript, as the approach taken by the protocol τ_A relies on the state $S \in \mathfrak{S}$ of π being in TEE-trusted memory so that it can produce an attestation that the

internal state was not altered nor read. Adapting the τ_A to break the deniability in π_D would require correctly implementing π_D, in particular, to produce σ_D in item 2. Because only an execution of τ_D can yield a valid value for σ_D, and τ_D requires that be state be in untrusted memory, then either τ_A cannot execute π_D, or it has to place the state $S \in \mathfrak{S}$ in untrusted memory for executing τ_D, at which point it can no longer produce an attestation that the state S exists only in TEE-trusted memory.

This countermeasure against TEE-capable adversaries is the basis of the defence presented in chapter 5 for the Signal protocol The underlying concept is the same, but the protocol τ_D is a simplified version that only ensures that all the information necessary to produce a full state S is available in untrusted memory, as opposed to verifying whether the entire state exists in untrusted memory. The effectiveness of this countermeasure depends on the validity of the assumption on there being a region of memory that is untrusted by a TEE, and it is ineffective, for example, against adversaries capable of nesting TEEs in an arbitrary chain, each encompassing both the trusted and the untrusted memory regions of the TEEs at the preceding level, as it becomes impossible to determine whether a memory region is trusted by *some* TEE.

Online Deniability

Online-deniable protocols provide deniability against adversaries who may have compromised a party during protocol execution and are able to interactively execute a protocol. Relying on the non-transferrability of the protocol state to determine the authenticity of a transcript is hence insufficient against such online adversaries because interactive execution of the protocol allows them to rely on the authentication provided by the protocol. This family of protocols therefore requires output forgeability based on protocol inputs alone. If the protocol inputs include long-term secrets, and these secrets are known to have existed in untrusted memory at some point in time, any transcripts from a protocol such as τ_A can be forged by executing a simulator τ_A^*.

4

Attack: Breaking Deniability in Signal with Remote Attestation

4.1 Introduction

As presented in § 2.5, Signal is an instant messaging protocol that provides end-to-end encrypted conversations. The Signal protocol consists of various subprotocols, including the X3DH handshake protocol, used during session initialisation, and the Double Ratchet algorithm, used for message exchange. The Signal protocol claims 'deniability properties', which are derived from the X3DH key exchange and the use of ephemeral keys for message key derivation. The adversarial goal is set to obtain a publishable convincing proof of either communication contents or communication occurrence, a goal that the protocol is explicitly designed to prevent. In this chapter, we will see how a TEE may be used to accomplish both goals.

A working implementation of this attack was built for the Signal protocol using SGX as a TEE platform. Although this type of attack is protocol-agnostic, Signal was chosen to build a demonstration because of the specific deniability properties claimed by the protocol designers, its status as a *de facto* messaging standard protocol [47], [48] which has attracted formal analysis [41], is recommended by cryptographers such as Bruce Schneier [49] and Filippo Valsorda [50] and, to some extent, privacy-focused organisations like the Electronic Frontier Foundation [51] and is approved for staff use in the US Senate [52].

4.2 Attack Setup

Imagine a conversation is to take place between Alice (A) and Bob (B) using the Signal protocol (π). An attacker with access to some TEE, Mallory (M), has compromised Bob[1] by forcing him to execute a protocol τ inside a TEE instead of π and wishes to obtain cryptographic proof that will convince a third party, Judy (J), of both communication occurrence and its contents. The entire (encrypted) conversation between Alice and Bob is observed by Eve (E). This scenario is represented in fig. 4.1.

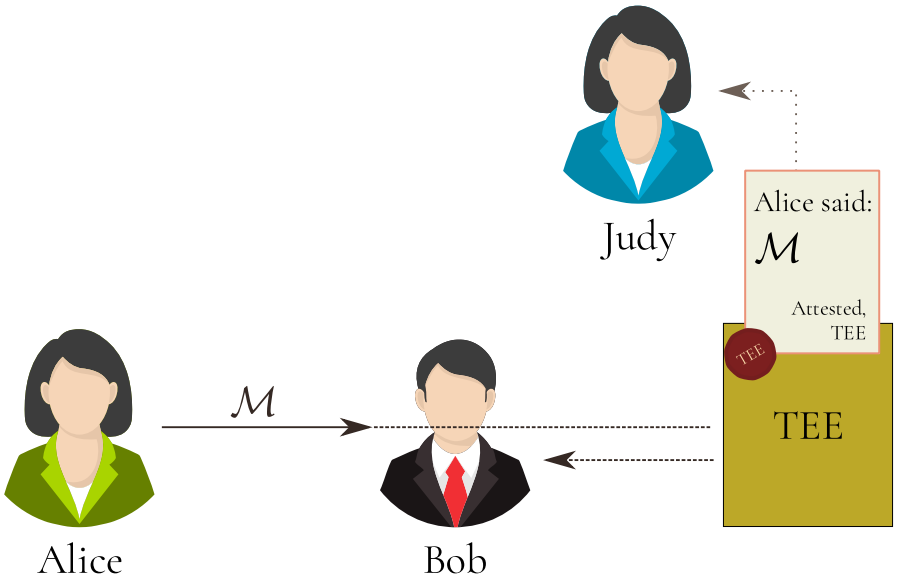

Figure 4.1: Simplified overview of the attack setup. Alice sends a message $\mathcal{M}$ to Bob, and a TEE attests to this fact, which can be verified by Judy.

4.3 Requirements

For the attack to be successful, it is necessary that a conversation actually takes place between Alice and Bob, or else there would be no transcript to generate. This requires that Alice be unaware of the attack at least while it is being executed because otherwise Alice might refuse to communicate, or may decide to limit the information she makes available. Similarly, this also requires that even if Alice is suspicious that this attack may be occurring, she cannot take any steps to minimise or neutralise the attack[2] apart from deciding whether to communicate and the information she chooses to share with Bob. This allows for generating a transcript of the communication.

[1] Bob and Mallory need not be different entities.

[2] For example, if a well-known countermeasure against this sort of attack was that it was ineffective against every third message, Alice may decide to send incriminating information only in every third message and otherwise send innocuous ones.

In addition, Judy needs to be confident that the transcript corresponds to a real conversation between Alice and Bob. This means that the transcript must contain information about its participants, the conversation contents and verifiable information about how the transcript was constructed. This information must convince Judy that the contents of the transcript were not tampered with; for example, it must not be possible for Bob or Mallory to insert messages into the transcript that Alice did not in fact send as part of her conversation with Bob. More generally, for Mallory's transcript to be convincing, she should be incapable of forging a transcript either completely (no conversation took place, but Mallory produced a transcript regardless) or partially (a conversation took place, but Mallory modified the transcript to add information or remove pieces she disliked). It also desired that the transcript be linked to both parts of the communication, Alice and Bob, to ensure that the communication is not taken out of context.

Lastly, although no particular assumptions are made about Mallory's capabilities, it is generally desirable that attacks are easy to launch a deploy. An attack that is difficult to launch is limiting and may reduce its usefulness. For example, if Mallory just learns that Alice and Bob will communicate today, but deploying the attack requires a week (because of complexity, difficulty to procure parts, cost or other reasons), it means that she will miss her chance to launch it.

These requirements are formalised below.

RA1 *Indistinguishability.* π and τ shall be indistinguishable from each other for Alice and Eve. In particular, it shall be impossible for Alice or Eve to determine whether Bob executes π or τ while a conversation is taking place.

RA2 *Verifiability.* τ shall yield an unforgeable transcript $\mathcal{T}$ that Judy can verify to be an authentic execution of τ.

RA3 *Authenticity.* τ shall yield an unforgeable transcript $\mathcal{T}$ unambiguously linked to both Alice. $\mathcal{T}$ should similarly be unambiguously linked to Bob.

RA4 *Inescapability.* Alice should not be able to influence whether Bob executes τ by executing π in a particular way. That is, if T denotes the event 'Bob executes τ instead of π', with probability $\Pr(T)$ and $\mathcal{X}$ is the set of all valid executions of π, then it should be the case that $\forall_{X \in \mathcal{X}} \Pr(T|X) = \Pr(T)$.

RA5 *Rigidity.* τ should yield an unforgeable transcript $\mathcal{T}$ that Mallory is unable to manipulate the contents of. For example, Mallory should not be able to reorder or skip inputs or outputs from τ.

RA6 *Portability.* Mallory can easily deploy an implementation of τ for any specific π implementation in use by Bob.

4.4 Implementation

The attack implementation was constructed using a modified Signal client using an SGX enclave for a TEE. The main part of the implementation is a remote procedure call (RPC)

server acting as a bridge between the Signal client and the SGX enclave, which allows for easily adapting any Signal client to use the TEE by substituting some of the procedures relevant to the Signal protocol with calls to the RPC server. The way these components are related is shown in fig. 4.2.

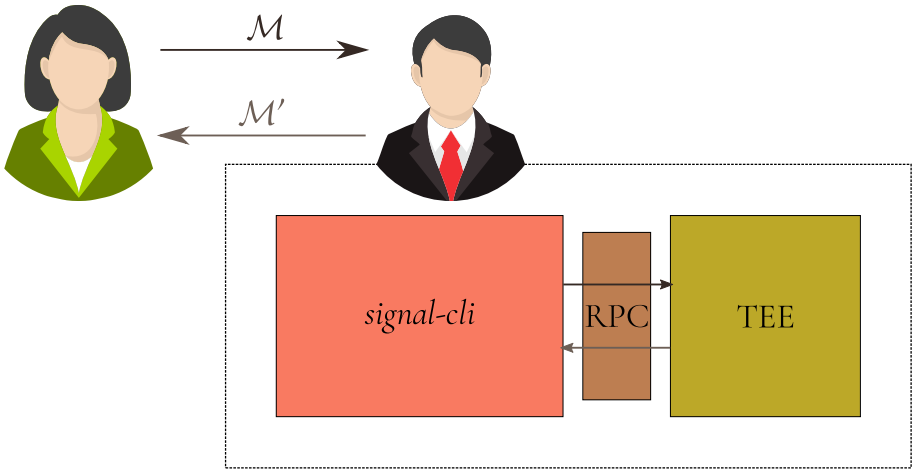

Figure 4.2: Overview of the relationship between the different components. Alice and Bob exchange messages $\mathcal{M}$ and $\mathcal{M}'$ using the Signal protocol. Bob is executing a modified version of the *signal-cli* client that executes the Signal protocol within a TEE. The Signal client and the TEE communicate through an RPC mechanism. Alice may be executing any Signal client.

Architecture

SGX Enclave

The SGX enclave was written in the C programming language making use of the Intel SGX SDK for Linux [53]. The enclave implements an interface for performing the different parts of the Signal protocol in addition to producing a valid attestation and an auxiliary interface to provide the client with necessary session information. The parts relevant to the Signal protocol implementation were written with the support of the *libsignal-protocol-c* library [54], which implements the Signal protocol and uses data structures compatible with those used by various Signal clients.

The enclave provides a complete implementation of the cryptographic aspects of the Signal protocol (the component labelled π in fig. 4.3), which includes establishing a session (starting a conversation) and message encryption and decryption. For each of these protocol operations, the enclave receives input parameters and a sealed session record, which contains the Signal protocol state, and returns the operation outputs and an updated sealed session record, as illustrated in fig. 4.3. To ensure correctness, some additional precautions are taken: the enclave also maintains a small internal state with information about each session, such as a monotonic counter to prevent replay attacks by reusing old session records, spinlocks on accesses to this internal enclave state to prevent concurrency attacks and the requirement that both signed and unsigned pre-keys be generated within the enclave and their secret part

be sealed when used as an input parameter. Refer to appendix A for a full reference on the enclave interface.

Signal Protocol State A crucial design decision during enclave design is ensuring that the protocol state maintained within the enclave is sufficient to meet the requirements set forth. An initial approach to this design, following minimalist principles, was providing an enclave interface that provided encryption and decryption primitives without regard to the Signal protocol, that is, the ratcheting steps in the Double Ratchet algorithm would be implemented in a hybrid model in which the logic for key derivation contained TEE and REE parts, and the encryption (for messages originating from Bob) and decryption keys (for messages sent by Alice) would be stored outside the enclave in a sealed form. The enclave would then provide an encryption and decryption interface using these keys. However, this approach was discarded because of the difficulty of ensuring that the keys were used for their stated purpose only, and not for other purposes, such as forging Alice's messages.

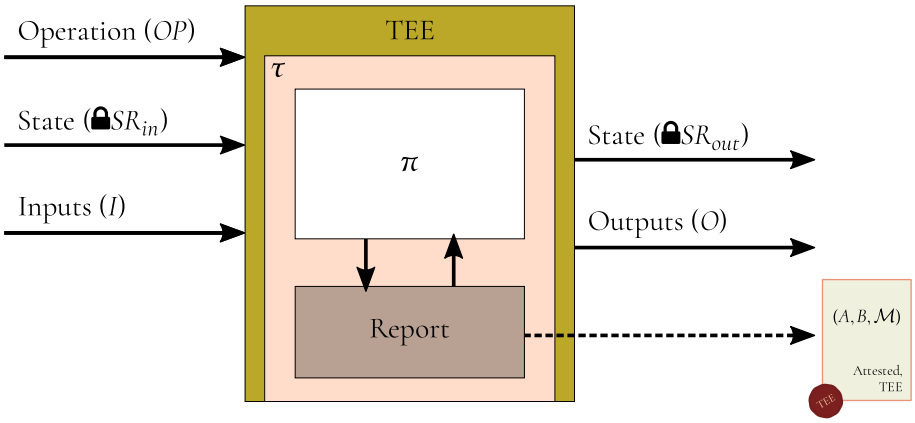

Figure 4.3: Overview of the general interface provided by the enclave. An operation OP takes input arguments $I = (I_0, I_1, \ldots)$ and a sealed state SR_{in}, and provides outputs $O = (O_0, O_1, \ldots)$. The state represented by SR_{in} is advanced, sealed and returned as SR_{out}. Internally, the TEE executes the protocol τ, which consists of the Signal protocol π and an internal state used for producing a report at the end of a session, from which a remote attestation will be obtained. τ is identical to π apart from the generated report.

The goal of the enclave implementation is to produce a verifiable transcript that faithfully represents a messaging session. This is accomplished by maintaining a per-session hash chain state within the enclave, based on the SHA-256 hash function [32]. For a conversation between Alice and Bob, in which Alice is executing the protocol in the enclave, let $\mathbf{H} : \mathbb{B}^L \to \mathbb{B}^{256}, \forall L \in \mathbb{Z}_0^+$ be the SHA-256 hash function, g^{I_A} and g^{I_B} be Alice's and Bob's public identity key, respectively, C_n and M_n represent a sealed message of type T_n[3] and its corresponding unsealed plaintext at step n, respectively, SR_n be the input sealed session record to the TEE at step n, all of which are represented as bit strings. Finally, let $\|$ denote the concatenation

[3]This type can be: Signal message or pre-key Signal message. T_n assumes the value of the one-character string `S` for the former and `P` for the latter.

operation and `"ABC"` refer to the ASCII bit representation of the string `ABC`. The attestation hash state at a step $n \in \mathbb{N}$ is defined as follows:

Initialisation $H_0 = \mathbf{H}\left(\texttt{"SESSION\_START"} \parallel g^{I_A} \parallel g^{I_B}\right)$

Sending (encrypting) $H_n = \mathbf{H}\left(\texttt{"A->B"} \parallel H_{n-1} \parallel \mathbf{H}\left(SR_n\right) \parallel \mathbf{H}\left(P_n\right) \parallel \mathbf{H}\left(C_n\right) \parallel T_n\right)$

Receiving (decrypting) $H_n = \mathbf{H}\left(\texttt{"B->A"} \parallel H_{n-1} \parallel \mathbf{H}\left(SR_n\right) \parallel \mathbf{H}\left(P\right) \parallel \mathbf{H}\left(C\right)\right)$

Finalisation $H = \mathbf{H}\left(\texttt{"SESSION\_END"} \parallel H_{n-1} \parallel \mathbf{H}\left(SR_n\right)\right)$[4]

Upon finalisation, an attestation σ is made on the value of H and returned. Verification consists of reconstructing the hash chain from a transcript T' to obtain H' and verifying whether σ is valid for this value, which implies that $H = H'$. Because SHA-256 is 2^{nd} preimage resistant[5], almost surely $T = T'$. The use of a hash chain allows for reducing the amount of state information that needs to be maintained within the enclave; consequently, σ (or H) alone is not a transcript and it is the responsibility of the enclave caller to maintain enough state information to construct a transcript externally.

The enclave was designed to support various attestation mechanisms, which can be used by themselves or in combination. The demonstration was written with remote attestation in mind, but the enclave can also optionally support SGX local attestations and the Curve25519 signature scheme[6].

Code Analysis Several steps were taken to ensure program correctness. In addition to carefully designing the functionality to be placed in the enclave, a wide selection of warnings were enabled[7] during the build process; in those instances in which a warning occurred, the circumstances were assessed carefully and the code was refactored to avoid logical errors and undefined behaviour. Similarly, the static analysis tools *Clang Static Analyzer* [55] and *Cppcheck* [56] were executed on all source files, and any programming errors they pointed at were promptly addressed. Moreover, extensive test cases were written with the goal of maximising code coverage and ensuring correct, reproducible and expected behaviour of the resulting code. The test cases themselves were executed with the dynamic analysis tools *AddressSanitizer* [57] (a memory error detector) and *UndefinedBehaviorSanitizer* [58] (an undefined behaviour detector) to further identify programming errors.

[4] Optionally, the enclave can release the per-session sealing key used to seal SR at this step. If this option is enabled, this last step is $H = \mathbf{H}\left(\texttt{"SESSION\_END"} \parallel H_{n-1} \parallel \mathbf{H}\left(SR_n\right) \parallel SK\right)$ for a sealing key SK .

[5] 2^{nd} preimage resistance means that given that $Y = \mathbf{H}(X)$, it is computationally infeasible to find $X' \neq X$ such that $Y = H(X')$ [39, p. 323].

[6] In this latter case, the value output is clearly not indicative of the presence of SGX; instead, it is assumed that the public key used to verify the attestation is deemed trustworthy.

[7] For example, most files were built with the following *GCC* flags: `-pedantic -pedantic-errors -Wall -Wextra -Wshadow -Wstrict-overflow=5 -Wunused -Wfloat-equal -Wpointer-arith -Wwrite-strings -Wformat-security -Wmissing-format-attribute -Wundef -Werror -Wmissing-prototypes`.

RPC Server

The RPC server was written in the C++ programming language and is responsible for loading the enclave and acting as a bridge between the enclave and the Signal client (using the gRPC protocol), relaying procedure calls inputs from the Signal client to the enclave and outputs from the enclave to the client. It was designed for simplicity and leanness and is defined in approximately 1200 lines of code (excluding generated sources), most of which are devoted to initialisation and message-passing logic.

The RPC server is also tasked with maintaining the appropriate state that is used to produce an output transcript file. The output file contains σ, which allows for easily verifying the authenticity and integrity of a transcript.

Signal Client

The unofficial Signal client *signal-cli* [59] was modified to execute the X3DH and Double Ratchet subprotocols in the enclave through the RPC server. This was done by building a simple gRPC client (less than 700 lines of Java code excluding generated code) that exposes these protocols using the same interface as the official library *libsignal-protocol-java* and then modifying the relevant parts of the client to use the *gRPC* client instead of *libsignal-protocol-java*, which was accomplished in fewer than 100 lines of trivial code, including build configuration files.

Although integration of this attack with an official client was not attempted, it is expected that the integration process would be similar to that of *signal-cli* for any of the official clients. For the Signal Android client in particular, it is also expected that parts of the work towards integrating with *signal-cli* can be reused because they share the same programming language, Java, and the same underlying libraries for protocol implementation.

4.5 Evaluation

Indistinguishability (RA1) This requirement is fulfilled. The protocol output of externally observable protocol outputs (such as session initialisation and message encryption) is the same as for the original Signal protocol because the Signal protocol itself is being implemented within the enclave. The enclave additionally provides an attestation that may be used to verify a transcript, but this additional functionality is observable neither by Alice nor Eve during protocol execution. While the current implementation may leak some information though side channels (e.g., it may take a different time to encrypt and send a message using the enclave than using the original application), this is of negligible effect because this information is not externally visible as is the nature of asynchronous communication; in the case of timing information, neither Alice nor Eve know when Bob's response would have been visible in both scenarios (viz., using an unmodified client and using a modified client that uses an enclave) to be able to compare.

Verfifiability (RA2) This requirement is fulfilled. It is possible to generate a transcript reflecting the inputs and outputs to the protocol, which is done by the RPC server in this implementation, which is almost always unforgeable upon including the attestation σ. The

unforgeability is derived from the unforgeability of σ and the 2^{nd}-preimage resistance of SHA-256. Furthermore, an attestation means that a known protocol must have been executed within an enclave.

Authenticity (RA3) This requirement is also fulfilled if the secret part of identity keys can be assumed to be secret (i.e., Alice and only Alice knows I_A and similarly with Bob and I_B). It is exceedingly infeasible to determine the correct shared secret of an X3DH handshake without knowledge of the parties' identity keys, which would prevent two parties from establishing successful communication. Therefore, if a session was established, it is almost always correct to assume that the parties to the session know or have access to their identity keys.

Inescapability (RA4) This requirement is partially fulfilled. The implementation τ currently only supports generating an enclave for newly established sessions, but not importing existing sessions into the enclave. This means that if Alice knows or suspects that Bob was compromised after a certain date, she could refuse to establish any new sessions with Bob after this date, but continue using existing sessions. This is an implementation issue that can be addressed by extending the enclave interface to support importing a pre-existing protocol state. However, doing this poses some additional challenges which need to be addressed externally; in particular, because the information that ties the session to Alice and Bob is exchanged during session establishment and not thereafter, it is impossible for the enclave to make this assurance. A secondary issue is that, even after addressing this authenticity concern, the state in the enclave can only be trusted after an asymmetric ratchet step happening inside the enclave because any message keys before this step exist outside the enclave and can therefore be used to forge messages.

Rigidity (RA5) This requirement is partially fulfilled. While the transcript is authentic (RA3) and reflects an honest execution of τ, it does not cover the transport layer, which means that Mallory may be selective about when τ is executed. In particular, Mallory could simply not decrypt some messages from Alice (but also not learning of their contents) or could encrypt messages to be sent to Alice but never send them. Because the Signal protocol allows for out-of-order decryption, Mallory can also choose the order in which to decrypt messages and send encrypted messages to Alice. Evidence of these attacks is present in the transcript if the session record sealing key is released (as the protocol state can show evidence of out of order or skipped messages), but may remain impossible to provably differentiate between deliberate attacks and honest transport errors. The current attack implementation could be extended to cover not only the Signal protocol but also the TLS session with the Signal service, which could address some of these issues.

Portability (RA6) This requirement is subjective in that the details of the implementation used by Bob is unspecified as is what Mallory considers an easy deployment. However, this attack focuses on generic operation protocols (viz. session establishment, message encryption and message decryption), which are likely to be present in any implementation. Similarly, as it was seen in § 4.4, modifying an existing client could be accomplished with minimum effort; gRPC, the protocol used for RPC communication may present a challenge in quick deployment

for a new implementation if the protocol needs to be implemented anew, but implementations of gRPC exist for many of the most common programming languages. Lastly, the attack uses Intel SGX, a technology widely available in modern desktop and server systems, which reduces the barrier of entry to perform an attack in the form of not requiring the acquisition of special hardware. In light of this, this requirement is considered fulfilled.

5

Defence: Restoring Deniability in the Presence of Remote Attestation

5.1 Introduction

As presented in chapter 4, deniability in the Signal protocol can be compromised by executing the protocol in a TEE by using the concepts discussed in chapter 3. This chapter introduces modifications to the protocol that protect against such an attack if the presence and properties of a TEE are known in advance.

5.2 Attack Setup

The defence is designed to defend against the attack presented in chapter 4. The attack configuration is hence similar, except that Alice or Bob, or both, may be compromised, which is a necessary assumption to provide protection against compromises on either side of the protocol.

Imagine a conversation is to take place between Alice (A) and Bob (B) using a protocol $\pi^\star$ based on the Signal protocol (π). An attacker, Mallory (M) is capable of compromising both Alice and Bob by forcing them, perhaps without their knowledge, to execute a protocol τ inside a TEE instead of π and wishes to obtain cryptographic proof that will convince a sceptical third party, Judy (J), of both communication occurrence and its contents. The entire (encrypted) conversation between Alice and Bob is observed by Eve (E). It is assumed that Alice, Bob and Mallory all have access to TEEs, all of which, for simplicity, behave identically.

5.3 Requirements

Implementing a defence to the attack described in chapter 4 requires taking steps that make it impossible to procure a convincing transcript. Because the attack described applies to the Signal protocol as it exists, it is necessary to effect some protocol changes, but because clients for the current protocol already exist, it is desirable that the protocol modifications be kept to a minimum. In addition, the resulting protocol must offer, if possible and at a minimum, the same assurances as the current Signal protocol; it is unacceptable to, for example, compromise perfect forward secrecy in the pursuit of TEE-resistant deniability. These requirements are formalised as follows:

RD1 *Soundness.* $\pi^\star$ shall be at least as sound as π. In particular, it shall not be possible for Eve to infer any information about the inputs to $\pi^\star$ that she would not be able to infer if Alice and Bob were executing π.

RD2 *TEE-resistance.* Given an unforgeable transcript $\mathcal{T}^\star$ resulting from executing $\pi^\star$ and an unforgeable transcript $\mathcal{T}$ resulting from executing τ, Judy shall not be able to infer any additional information from $\mathcal{T}$ as she would from $\mathcal{T}^\star$. In particular, $\mathcal{T}^\star$ shall appear at least as faithful as $\mathcal{T}$.

RD3 *Similarity.* $\pi^\star$ should be as similar as possible to π.

RD4 *Backwards-compatibility.* A party executing $\pi^\star$ should be able to communicate with another party that executes π.

5.4 Implementation

The defence was designed to follow a similar architecture to that used for the attack implementation seen in fig. 4.2, namely a modified Signal client that communicates with an SGX enclave through an RPC system. Unlike the attack enclave, the defence enclave does not execute the entire protocol, but it is used instead to attest that the material used to derive the MAC keys resides in untrusted memory. The defence also incorporates some simple changes to the asymmetric ratchet step in the Double Ratchet algorithm used in the Signal protocol. Unlike the attack, which was completely implemented as a working demonstration, only the enclave and RPC server components were fully implemented for the defence, but not so the modified Signal client, because of time constraints. As a result of the changes introduced to the Signal protocol itself, it is expected that the modified Signal client implementation would be slightly more complex than for the attack. An overview is shown in fig. 5.1.

Proposed Modified Double Ratchet Algorithm

Initialisation Before communicating, Alice and Bob use their respective enclaves to generate the key pairs (p_A, g^{p_A}) and (p_B, g^{p_B}), respectively. The public keys g^{p_A} and g^{p_B} are made available from their enclaves, and they use their enclaves to produce the attestations σ_A and σ_B, respectively, to these public keys. The private keys p_A and p_B are known only to the TEE and must *not* be made available in untrusted memory.

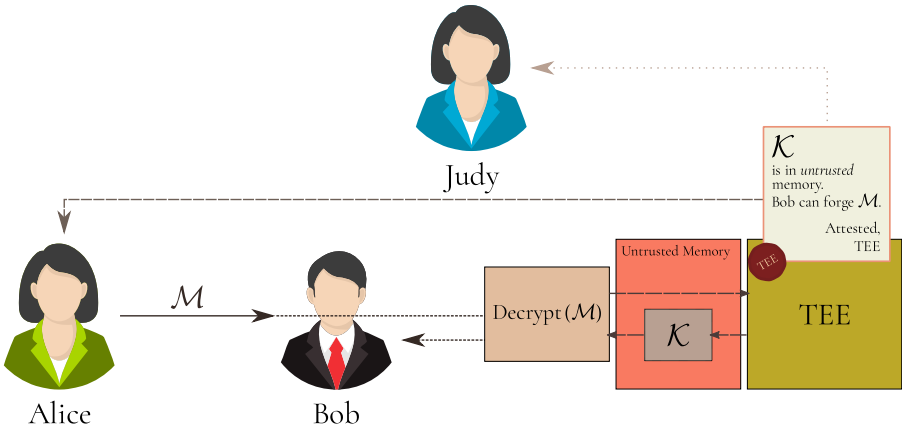

Figure 5.1: Simplified overview of the defence implementation. Alice sends a message $\mathcal{M}$ using some authenticated encryption scheme to Bob, who uses a TEE to produce an attestation that the decryption and authentication keys reside in untrusted memory and sends it to Alice. Judy is unable to tell whether $\mathcal{M}$ is authentic because it could have been forged by Bob.

Double Ratchet Algorithm The Double Ratchet algorithm described in § 2.5 was modified so that the asymmetric ratchet step (item 3 in § 2.5, with $RK_A = \mathbf{H}_1\left(RK_A, \left(\left(g^{R_B}\right)^{R_A}\right)\right)$) uses a modified version of the X3DH handshake that we have named EX3DH. With the proposed modifications, Alice uses her TEE to compute $L_A^\star = \left(\left(g^{R_B}\right)^{R_A}, \left(g^{R_B}\right)^{p_A}, \left(g^{p_B}\right)^{R_A}\right)$[1], which is used to compute $RK_A^\star = \mathbf{H}_1\left(RK_A, L_A^\star\right)$. The TEE takes R_A, g^{R_B} and g^{p_B} as inputs from untrusted memory and places $L^\star$ in untrusted memory. The modified protocol uses $RK_A^\star$ in lieu of RK_A but is otherwise identical to the original protocol.

Key Distribution and Backwards Compatibility This scheme requires for Alice and Bob to know each other's (g^p, σ) pairs (viz., Alice needs to know (g^{p_B}, σ_B) and Bob, (g^{p_A}, σ_A)), as g^p is used for calculating $L^\star$ and σ is used to establish trust in the TEE that performs the calculation. This could be accomplished out-of-band, such as by querying some directory service or in-band, which compromises deniability by producing evidence of a communication attempt. We propose the following in-band mechanism for Alice to establish a deniable session with Bob:

1. Firstly, Alice executes the Signal protocol normally to establish a session with Bob.
2. In her first message, Alice sends a message $\mathcal{M}_A^\star = (g^{p_A}, \sigma_A)$. Alice may also send additional messages normally at this point, but their deniability may be compromised.
3. If Bob does not understand $\mathcal{M}_A^\star$, he replies normally with a response $\mathcal{M}_B$. Otherwise, he verifies σ_A and computes $L_B^* = \left(\left(g^{R_A}\right)^{R_B}, \left(g^{p_A}\right)^{R_B}, g^{R_A w}\right)$, where w is a well-known

[1] Respectively, Bob computes $L_B^\star = \left(\left(g^{R_A}\right)^{R_B}, \left(g^{p_A}\right)^{R_B}, \left(g^{R_A}\right)^{p_B}\right)$.

private key, and computes his root key $RK_B^* = \mathbf{H}_1\left(RK_B, L_B^*\right)$, and then re-computes his derived sending and receiving keys. He finally sends a message $\mathcal{M}_B^* = (g^{p_B}, \sigma_B)$.

4. Bob computes $L_B^\star = \left(\left(g^{R_A}\right)^{R_B}, (g^{p_A})^{R_B}, \left(g^{R_A}\right)^{p_B}\right)$ using his TEE and then derives his root key $RK_B^\star = \mathbf{H}_1\left(RK_B, L_B^*\right)$ and re-computes his derived sending and receiving keys. He may proceed to send additional messages using the new root key, which are deniable.

5. If Bob replies normally with $\mathcal{M}_B$, Alice may continue communicating using the unmodified Signal protocol or abort the communication. Otherwise, if Bob replies with $\mathcal{M}_B^*$, she verifies σ_B and uses her TEE to compute $L_A^\star = \left(\left(g^{R_B}\right)^{R_A}, \left(g^{R_B}\right)^{p_A}, (g^{p_B})^{R_A}\right)$ and derive her root key $RK_A^\star = \mathbf{H}_1\left(RK_A, L_A^\star\right)$.

6. Communication proceeds using $L^\star$ instead of $\left(g^{R_B}\right)^{R_A}$ for the asymmetric ratchet step. Messages sent by either party are deniable.

The steps described for in-band communication provide deniability of the contents exchanged and backwards-compatibility with the normal Signal protocol (in which case deniability is compromised) but are complex and still provide evidence that Alice attempted to communicate with Bob. Because there is no requirement for ephemeral or short-lived enclave keys p, this backwards-compatible exchange only needs to be executed once, and any subsequent sessions between Alice and Bob can use previously learnt keys.

Required Changes to the Signal Service

The described protocol with in-band key exchange executes on top of the existing Signal protocol and all the changes required are limited to client modifications. There is therefore no need for changes to the Signal service, and the modified protocol is indistinguishable from the unmodified Signal protocol from the perspective of the Signal service. However, the Signal service could be modified to provide a directory service providing (g^p, σ) pairs, which eliminates the need for an in-band key exchange mechanism and does not result in evidence of communication attempts.

5.5 Evaluation

Soundness (RD1) We evaluate soundness on the different properties of the Signal protocol π, comparing it with $\pi^\star$.

Confidentiality and Integrity The modified protocol $\pi^\star$ uses additional inputs for $\mathbf{H}_1$ with respect to the original Signal protocol π, namely $\left(g^{R_B}\right)^{p_A}$ and $(g^{p_B})^{R_A}$. We can assume that, because these are *additional* inputs, the output from $\mathbf{H}_1$ in π has at least as much entropy as that in $\pi^\star$ and looks indistinguishable to Eve without additional information.

Perfect Forward Secrecy We assume that Eve later learns the values of p_A and p_B. In this case, Eve is unable to predict the output of $\mathbf{H}_1$ because she still cannot compute the requisite $\left(g^{R_B}\right)^{R_A}$. Hence, learning p_B and p_B does not compromise past communications under those keys and PFS is preserved.

Deniability Without loss of generality, we assume that Eve learns the values of p_A and R_A. She can then compromise the confidentiality of communications using those secrets because she can fully compute the output to $\mathbf{H}_1$, but because none of the values is connected to I_A, she cannot prove that Alice was involved. This is no worse than Eve learning R_A in the original Signal protocol π.

TEE-resistance (RD2) When executing the protocol $\pi^\star$, which has an input $L^\star$ for its asymmetric ratchet step (from which root keys are derived), an attestation is produced to the fact that $L^\star$ exists in untrusted memory, and that, since only a specific TEE knows the true value of $L^\star$, any value of $L^\star$ is indistinguishable to an external observer. Therefore, an unforgeable transcript $\mathcal{T}$ would depend on an untrusted input $L^\star$. A simulator for $\pi^\star$ executing in a TEE would still be unable to prove the true value of $L^\star$ and hence any transcript $\mathcal{T}$ cannot be more trustworthy than $\mathcal{T}^\star$.

Similarity (RD3) The original Signal protocol π and the proposed protocol $\pi^\star$ are essentially identical, except for the additional values used to derive the root keys, and existing clients only need to be modified to change the input used in this step, and to retrieve the additional public key g^p and attestation σ required for computing the extended handshake $L^\star$.

Backwards-compatibility (RD4) Because of the similarity between π and $\pi^\star$, it is easy to implement a client that implements both protocols, and backwards compatibility may be maintained (with potentially degraded deniability) if the values for g^p required for $L^\star$ are unavailable. The proposed in-band mechanism for upgrading from π to $\pi^\star$ further requires no changes to the Signal service, which allows Signal users to modify their current clients to benefit from stronger deniability against TEE-capable adversaries without relying on the Signal service to implement a directory service for providing g^p and σ.

5.6 Other Countermeasures

The defence mechanism presented in § 5.4 uses remote attestation to strengthen deniability by modifying a step in the Double Ratchet algorithm. This defence was designed to be backwards-compatible with the Signal protocol and to be able to negotiable deniability through an in-band mechanism. If these requirements are omitted, there exist other approaches to achieving deniability. All of these countermeasures assume that the TEE platforms an adversary has access to are known.

Deniable X3DH

Consider a TEE that generates key pairs $\left(I, g^I\right)$ such that the private key I never leaves the TEE trusted memory region, and which is capable of using I for computing an X3DH handshake. When the TEE produces an X3DH handshake, it always places the result K in untrusted memory. The TEE also produces an attestation σ_I, and the pair $\left(g^I, \sigma_I\right)$ is uploaded to a directory service. This directory service and TEE combination may be combined to execute the Signal protocol while preserving deniability. It would proceed as follows:

Key Generation Before communicating, Alice (A) and Bob (B) generate their identity keys to be used in the X3DH protocol.

1. Alice and Bob use their TEEs to generate $\left(I_A, g^{I_A}, \sigma_{I_A}\right)$ and $\left(I_B, g^{I_B}, \sigma_{I_B}\right)$, respectively. The TEE provides an interface for exporting $\left(g^I, \sigma_I\right)$.
2. Alice and Bob upload their $\left(g^I, \sigma_I\right)$ pair to the directory service.
3. Alice's and Bob TEEs use their TEEs to compute the required signatures on their signed pre-keys S.

Communication Suppose now that Alice wishes to communicate with Bob using a post-X3DH protocol π. She could execute a deniable variant of the X3DH protocol as follows:

1. Alice retrieves $\left(g^{I_B}, \sigma_{I_B}\right)$ from the directory service and verifies that the attestation is valid.
2. Alice uses her TEE to compute K in the usual way. The TEE places K in untrusted memory.
3. Alice sends $\mathcal{M}$ and an initial message $\mathcal{M}_0$ as usual. Alice's message $\mathcal{M}$ optionally also includes σ_{I_A}.
4. Bob verifies that σ_{I_A} is valid.
5. Bob uses $\mathcal{M}$ to reconstruct K' using his TEE, which places K' in untrusted memory.
6. Bob attempts to decrypt and verify $\mathcal{M}_0$ using key K'. If successful, $K = K'$ and communication between Alice and Bob may continue under protocol π and key K.

Soundness The deniable X3DH protocol performs the same steps as the regular X3DH protocol, except for verifying attestations, and has similar security properties to regular X3DH. However, it ensures that K exists in untrusted memory, and because the protocol π is initialised using the key K, any transcript resulting from executing π could have been forged by Alice, Bob or an attacker who compromised either party. This protocol does not prevent a transcript from being generated in another TEE, but any such transcript cannot be linked to any particular party.

Online-Deniable Protocols

Protocols designed to be deniable against online adversaries, such as DAKEZ [60], can also provide deniability against TEE-capable adversaries if the long-term keys used are known not to have been generated and reside within an adversarial TEE. Remote attestation can enhance these protocols by providing proof that the long-term keys resided in untrusted memory at some time in the past.

TEEs That Do Not Produce Transcripts

All of the countermeasures presented in this chapter include some step that attests that authentication keys (or material sufficient to derive them) existed in untrusted memory, and could therefore have been used to produce forgeries. As a result, the protocol has properties analogous to executing it in an REE.

While this has the desired effect of providing deniability, it seems intuitive that TEEs could be used to *strengthen* protocols by protecting protocol secrets while also providing deniability. This is, a TEE could produce an attestation that all keys exist only within the TEE *and* that it does not produce any transcripts; for example, the TEE presented in chapter 4 could be trivially modified not to produce any transcripts and to provide an attestation to the X3DH key K existing only within the TEE. Although this could arguably result in a more secure protocol because the TEE provides additional protection against key compromise as well as secure erasure mechanisms. Unfortunately, proving that a protocol under these conditions is deniable requires proving the absence of any authenticators in the TEE outputs, which is undecidable if the messages can take an arbitrary form, as it amounts to the halting problem [61].

6

Related Work

Deniability was introduced in [62] as deniable encryption and [63] in the form of a *plausible deniability service*, in contrast with the *non-repudiation service* in [24]. Deniable message authentication is presented in [64] as a concurrent zero-knowledge problem, and we find later works such as [25] that evaluate plausible deniability in the context of existing Internet protocols like the SIGMA authenticated key exchange scheme used in the IKEv1 protocol. The deniability properties of IKE and SIGMA are further studied in [27]. Other Internet protocols, notably TLS, are studied in [65], which introduces the idea of quantifying deniability as a 1-out-of-n property, where n represents the number of potentially valid senders or generators of a message, and proposes protocols for achieving both authentication and 1-out-of-∞ deniability.

The work in [26] provides a formal treatment of deniability in the generalised universal composability framework [43], demonstrates that deniability is impossible against *adaptive* adversaries, who can corrupt parties at any point in the execution of a protocol, including before, after and during its execution and proposes a new key exchange protocol, Key Exchange with Incriminating Abort (KEIA). KEIA guarantees deniability if the protocol terminates successfully, which allows for a deniable exchange after establishing a shared key, but it still provides evidence that two parties attempted to communicate if it is aborted by a malicious party. The Off-the-Record (OTR) messaging protocol presented in [66] has deniability as an explicit goal in messaging protocols, and [60] proposes three new key exchange protocols that provide strong deniability in secure messaging, both against *offline* adversaries, who see a transcript after communication took place and *online* adversaries, who may collude with one of the parties interactively while communication is taking place. Two of these suggested improved protocols, DAKEZ and XZDH, are to be incorporated in version 4 of the OTR protocol [67], motivated by the desire to provide deniability properties [68].

None of these works considers the specific case of using remote attestation against deniability, although TEE-capable adversaries may be encompassed under the adaptive adversary model used in [43]. The two proposals in [65] for fully deniable protocols based on the Diffie-Hellman key exchange and the RSA cryptosystem conflate the forgeability of MACs with deniability and are susceptible of becoming non-deniable if one of the parties executes the proposed protocols in a TEE, an assumption also made in [66].

The Signal protocol received formal treatment in [41], but this work focuses on key indistinguishability and does not explore its deniability properties. The deniability properties are considered in a survey of different messaging protocol implementations [69], but the scenarios considered do not contemplate trusted hardware or other trusted third parties. In [70], followed up in [71, ch. 5], the Signal *server* is executed in a TEE as a means of enhancing its security, but this is done in the context of strengthening the execution environment of applications (in the particular case of Signal, protecting the secrecy of metadata).

Using remote attestation to impair deniability appears to be a novel concept. The *accountable decryption* concept in [71, ch. 4], for example, is motivated by producing an auditable log of decryption operations, but the idea is not generalised to communication protocols. The Town Crier data feed [72] applies a similar idea to that used in the attack discussed in this thesis to provide authenticated data feeds for smart contracts. The system described uses a TEE based on SGX to retrieve information from HTTPS-enabled websites and use remote attestation to verify its authenticity; the authenticated feed can then be relayed to a smart contract, at which point the attestation is verified and the information used.

The Town Crier scheme elegantly addresses the issue of providing trusted data to smart contracts, which may originate from parties oblivious to the contract. This is especially useful for financial derivative contracts. Consider a simple smart contract that specifies that a buyer will receive 5 ETH from a seller if the price of 1 g of gold at an agreed-upon future time is above 100 USD in an agreed-upon market. This contract depends on external information that is unavailable to the smart contract, and it is difficult to provide a trusted data source. However, if the gold rates from this market are published on an HTTPS-enabled website that both the buyer and the seller trust, the Town Crier system can be used to provide this information to the smart contract when the time arrives.

The reason why Town Crier works as a trusted authenticated feed is that remote attestation is used to show the price of gold *and* that originated from a trusted source, which is accomplished by making the data feed non-repudiable by stripping TLS, upon which HTTPS is based, of its deniability. Although the underlying mechanics of Town Crier are similar to the attack in chapter 4, both with the effect of turning a deniable channel into an undeniable one, its creators did not anticipate the wider implications of their idea, such as in the context of deniability.

7

Discussion

The attack presented in this thesis in chapter 3 is highly relevant to modern communications because the only requirement that it imposes on a would-be attacker is access to a TEE. This requirement is not extraordinary nor extenuating for an attacker, as TEEs are widely available in consumer devices, such as TPMs and SGX-capable processors. The attack against the Signal described in chapter 4 further shows that this attack is not merely theoretical, but that it can also be readily deployed against real-world protocols with minimal effort.

7.1 Contributions

The attack presented can be adapted to work against any authenticated protocol and the impossibility result in chapter 3 means that no mitigations exist for an offline adversary of unknown capabilities. This is a significant finding because it implies that deniable communications are impossible in the general case, strengthening the impossibility results against an online adaptive adversary in [26]. Furthermore, to the knowledge of the author, the specific impact of remote attestation in deniable communication protocols has not been previously studied in the literature, despite the especially devastating consequences it has for deniability.

Trust Assumptions

Successfully executing this or similar attacks has the goal of convincing a *judge* that a transcript is credible and genuine, a judge who may be sceptical of any transcript presented by the attacker. In accomplishing this task, we have used a TEE to provide certain guarantees of the genuineness of a transcript. Credibility demands that the judge trust all the components of a remote attestation, from the hardware the TEE relies upon to the software it is executing to the specific remote authentication scheme used.

This trust component makes a remote attestation attack similar to existing and well-known methods of exacting the same information, such as the forensic data retrieval of information stored on a device or witness accounts, which can yield a similar transcript. The difference

between these other methods and remote attestation is that, if credibility can be established, a remotely attested transcript can be generated and validated almost entirely automatically, significantly reducing the cost and complexity of producing and validating credible transcripts.

7.2 Attack Difficulty

Carrying out the attack described in chapter 3 requires an attacker capable of executing a communication protocol within a TEE, and to make use of the TEE to produce a remote attestation on the transcript. Hence, the difficulty of the attack lies in the difficulty of procuring a TEE and executing the protocol therein. We distinguish two types of adversaries based on their capabilities:

Difficulty of Procuring a TEE

As it has been already stated, TEE-capable hardware is widely available in consumer hardware, so acquiring a TEE is not deemed an insurmountable obstacle for an attacker. In the specific case of SGX, a potential obstacle to deploying this attack lies in the requirement of signing the enclave code with an Intel-whitelisted signing key. However, these keys could be legitimately obtained by an attacker (for example, if the attacker develops enclaves for legitimate purposes), or a determined attacker could similarly steal such whitelisted keys from legitimate developers, as seen in the use of compromised device-signing keys in the Stuxnet malware[1] [73, p. 20]. Note that the use of stolen enclave signing keys does not compromise the trust in the enclave because they do not confer the attacker the ability to manipulate an attestation; they only allow the attacker to deploy an enclave.

Difficulty of Executing a Modified Protocol

Deploying the attack presented in this thesis requires not only having access to a TEE but also compromising one of the parties to a communication. This requires only that an attacker can install or modify the software executed by their target, but it does not require any additional capabilities, such as installing additional hardware or physical or local access. This level of access can be obtained by means such as root-level exploits of the operating system, which are common in Android malware [74]. Once an attacker can deploy attack code to a TEE and has compromised a target, the attack may commence and be executed remotely either by implementing an RPC model similar to that in chapter 4 (if the target's hardware does not provide TEEs or the attacker does not wish to use them) or by modifying the target's software to use local TEE capabilities (if the attacker wishes to use a TEE leveraging the target's hardware).

[1] In this case, the attacker managed to procure signed drivers by compromising the signing keys of not one but *two* hardware manufacturers.

Defending Against Remote Attacks

Thwarting remote attackers is possible by implementing defences such as those presented in chapter 5, like guaranteeing that any keys used for authentication exist in untrusted memory (i.e., outside of a TEE). However, these countermeasures do not protect against adversaries with *hardware-modifying* capabilities, who may install additional TEEs. For example, an attacker with *hardware-modifying* capabilities could install a TEE that produces an attestation of the state of the entire target system, including any existing TEEs. Defending against *hardware-modifying* attackers requires foregoing authentication entirely.

7.3 Implications for Zero-Knowledge Proof Transferrability

The concept of non-transferrability of zero-knowledge proofs was presented in § 2.6, namely that upon executing a zero-knowledge protocol, a verifier V may be assured that a prover P knows some information I, but that V cannot provide a transcript of the exchange that will similarly convince a verifier V'. Zero-knowledge proof transferrability can be modelled similarly to deniability:

> We assume an arbitrary protocol π in which there is a *prover* P *who wishes to prove knowledge of* I, a *verifier* V, an informant E who observes the protocol execution, a misinformer M who does not observe the protocol run but executes a simulated protocol π^* and a judge J who will rule at the end of the protocol whether P knows I. The protocol π *provides a non-transferrable proof of knowledge of* I if J is not able to distinguish between E and M.

This definition allows us to apply the same principles used for attacking deniability to providing proof transferrability, to wit, using executing a protocol τ that produces an attested transcript of its execution, and therefore allows for transferring the knowledge gained from the proof, namely some level of certainty about whether P knows I. In the sample protocol in § 2.6, if the TEE is used on V's side, the attestation would ensure that γ is received before providing the value for b and that b is selected at random; if the TEE is used on P's side, the attestation would ensure that γ is computed honestly (and, by extension, that r is picked at random), and that y is computed honestly.

7.4 Moving Forward: Protocol Design

The significant impact of remote attestation on deniable authentication is compounded by the relative low cost and ease with which an attack can be carried out; this poses new challenges to protocol designers wishing to implement deniable authentication. In particular, it is paramount that the potential availability of remote attestation be incorporated in plausible threat models; despite the result that unconditionally deniable protocols are impossible, it is possible to arrive to an appropriate threat model that provides an adequate level of deniability in the presence of remote attestation.

Deniability can no longer be considered an absolute or unconditional property, but one that can exist to different degrees. The required level of deniability must be carefully assessed and documented, so that potential protocol implementers and users may aware of potential protocol limitations. Protocol designers concerned about deniability may consider incorporating countermeasures such as the ones presented in chapter 5 that raise the barrier to leveraging adversarial TEEs to impact deniability. For example, in the case of protocols meant for mobile platforms, it may be useful to incorporate the TEEs provided by the platform, such as ARM TrustZone, to strengthen the deniability of communications.

Implications for Open Source Projects

Incorporating remote attestation into protocols, which is a valid countermeasure, may pose additional challenges to open source and free software projects. The concept behind incorporating a TEE is that its code can be reasonably trusted behave in a certain fashion (such as ensuring that authentication keys exist in untrusted memory). However, having this trust requires being certain that of the identity of said TEE, which includes the binary code it executes. As a result, it does not scale very well to have many variations of this binary code.

Two of the premises of open source [75] and free software [76] are free source redistribution and realising derived works, which imply the ability to build all of the components of a system from their source code, with or without modifications. Even in the case of building a system as it was originally published, it is possible that the resulting machine code differ from the published one due to differences in the build systems. As these programs have different machine code, the trust in the resulting machine code needs to be evaluated in each instance, an approach that does not scale well for wide-audience services such as *Signal*. On the other hand, *not* building the system entirely from its source code (and using the vendor- or developer-published binaries) may hinder users' abilities to audit the program provided or to exercise the freedoms they seek by using this type of software.

The issue of having the same source code compile into different binaries may be addressed by focusing on having reproducible builds, that is, preparing and documenting a build process that produces deterministic results and bit-identical binaries. The issue of not being able to determine trust in binaries derived from *modified* sources remains an open problem, but it is possible to design simple TEEs so that the need for modifications is reduced and that the resulting modified versions are themselves easy to audit.

7.5 Other Applications: MAC-as-signatures

Authentication schemes can be *verifier-forgeable* or *verifier-unforgeable*. *Verifier-forgeable* schemes, like all symmetric authentication schemes, allow any verifier to produce valid authentication tags, as opposed to *verifier-unforgeable* schemes, like the typical use of digital signatures, that allow verifiers to only verify messages, but not to produce valid authentication tags that can be verified as valid by other verifiers.

The attack presented works against *any* authenticated channel, and in essence turns any verifier-forgeable scheme into a verifier-unforgeable one. Despite the deleterious consequences this has for deniability, it can have positive applications in cases where verifier-unforgeability

is desirable but is not feasible using traditional verifier-unforgeable schemes such as digital signatures. Digital signatures normally demand orders of magnitude more computations than symmetric message authentication codes, as shown in table 7.1. As a result, symmetric authentication may be preferred in resource-constrained or low-cost devices, such as digital sensors.

In the case of digital sensors, it is easy to see why unforgeability is desirable. When sensor data are received and stored for later use, it is useful to verify that the data were not tampered with between the time they were received and the time of use. In these situations, remote attestation could be used to implement schemes that offer similar security guarantees to digital signatures but without incurring the additional cost of actually producing signatures.

Such a scheme could be implemented as follows:

1. A sensor S negotiates a symmetric key K with a receiver R. This may be done using an authenticated Diffie-Hellman exchange to ensure the legitimacy of the data feed (i.e., that it originates from a given sensor). The receiver uses a TEE for this exchange that ensures that K exists only within the TEE.

2. The sensor sends data a data feed $D = \{(\mathcal{M}_n, A_n) \mid n \in \mathbb{N}\}$, where $\mathcal{M}_n$ represents a datum authenticated with an authentication tag A_n, generated using key K and some symmetric authentication scheme.

3. The receiver uses its TEE to validate each datum and produces an output $D' = \{(\mathcal{M}_n, \sigma_n) | n \in \mathbb{N}\}$, where $\mathcal{M}_n$ represents a valid authenticated datum and σ_n is an attestation that it was received with a valid authentication tag A_n.

4. When the sensor data are used, the accompanying attestation may be used to determine whether it is genuine.

If producing an attestation to each datum is deemed exceedingly expensive (in terms of computation, verification, storage or other factors), alternate schemes requiring a single attestation are also possible. One such alternative is generating a digital signature key pair within the enclave, producing an attestation to its public key and using digital signatures for σ_n. Another alternative, if the need for data authentication upon use is expected to be rare, is for the enclave to export a sealed key $\text{🔒}K$ and provide σ_n upon request given $(\mathcal{M}_n, A_n, \text{🔒}K)$.

Table 7.1: CPU cycles spent (as reported by the `RDTSC` instruction) generating authenticators for random 4 KiB messages from a SHA-256 hash using randomly generated keys using the *OpenSSL* library version 1.0.2g (and the *NaCl* library version 20110221 for Poly1305 and EdDSA) on an Intel® Xeon® E5-2670 CPU. $|x| = 32768$. Note that for the best and average cases symmetric authentication is at least two orders of magnitude faster than asymmetric authentication.

Method	Symmetric	$\min(x)$	$\max(x)$	$\bar{x}$	s_x
NULL[a]	N/A	3.9×10^1	6.8×10^1	4.6×10^1	3.2
Poly1305	Yes	9.2×10^3	7.5×10^5	9.4×10^3	1.1×10^4
AES-128-CMAC	Yes	1.5×10^4	1.2×10^6	1.6×10^4	1.9×10^4
HMAC-SHA-256	Yes	5.5×10^4	4.1×10^6	5.8×10^4	6.5×10^4
ECDSA (X9.62-2005 prime256v1)	No	2.0×10^5	9.4×10^5	2.0×10^5	2.4×10^4
DSA-3072	No	2.4×10^6	2.0×10^7	3.1×10^6	3.9×10^5
EdDSA (Ed25519)	No	7.1×10^6	1.9×10^7	7.4×10^6	3.9×10^5
RSA-3072	No	1.2×10^7	3.5×10^7	1.6×10^7	8.6×10^5

[a] NULL denotes that no authentication operation was computed on the message. It is reported for establishing a baseline measurement.

8

Conclusions

It is possible for an adversary to use remote attestation to break deniability in existing communication protocols, an attack that can be applied to any protocol that provides an authenticated channel. As evidenced in the work presented in chapter 4, this is not merely a theoretical concern, but practical attacks are possible by leveraging existing off-the-shelf hardware. The work presented herein can further be extended to compromise non-transferrability of zero-knowledge proofs, as discussed in § 7.3.

The impact of remote attestation is significant. For one, it follows that any claims about providing deniability are invalid if the adversary model does not consider TEE-capable adversaries. For another, it is impossible to unconditionally provide deniability even against offline adversaries, which requires either making assumptions about adversarial capabilities in terms of TEEs or abandoning authentication in situations that require deniability. Neither of these outcomes is ideal: the former because these assumptions may later be found to be invalid and the latter because unauthenticated channels are susceptible to forgeries. Remote attestations are powerful because they result in a paradigm shift in thinking about adversaries: they can provide evidence that a TEE-capable adversary is honest, in stark contrast with the normal case, in which adversaries are malicious and cannot be trusted to be honest. Breaking this assumption results in compromising deniability without breaking the protocols themselves.

The work in chapter 5 presents countermeasures that may provide an acceptable level of deniability depending on the threat model used. Implementing these countermeasures in the Signal protocol requires changes to the protocol itself, which may hinder the ability to deploy these countermeasures for reasons of backwards compatibility. These countermeasures also show that remote attestation may be used to strengthen deniability against TEE-capable adversaries, and it is therefore reasonable to claim that deniable protocols *should* incorporate remote attestation in this manner.

As examined in chapter 6, the use of remote attestation to compromise deniability is an underexposed area in the literature. Hence, this is an area that requires the attention of protocol designers, specifically in the development of strong countermeasures against TEE-capable adversaries and the incorporation of remote attestation in threat models. Further

development in this area may also see the fruition of benign applications to the attack shown, such as for providing computationally inexpensive non-repudiable channels, discussed in § 7.5.

8.1 Future Work

The attack presented against the Signal protocol has a few limitations. One of them is that it was implemented for one-to-one communications, and is ineffective against group conversations. A continuation of this work might incorporate the necessary functionality into the enclave to support group conversations and integrate this new functionality with the modified version of the Signal client. Similarly, the modified client is not one of the official Signal clients; integrating the attack with such an official client would make the attack described more feasible to deploy without the target noticing or requiring them to use unconventional software. The attack enclave could also be improved to fully support exporting the keys used for sealing the session record once a transcript is produced; this would facilitate integration with other unrelated attacks, such as man-in-the-middle sort of attacks, which may reduce the need to trust the TEE.

Likewise, a fully-working client implementing the defence described in chapter 5 was not completed. Since having a means to mitigate potential attacks is critical to protecting deniability, this integration ought to be completed. Because the defence mechanism presented does not require any changes to the Signal service, accomplishing this task would provide *Signal* users with a mitigation against TEE-capable adversaries that can be deployed as needed.

On a more general note, this thesis focused on the Signal protocol. However, the attack described can be implemented against any protocol that provides authentication (and is useful against protocols that provide deniable authentication). As such, potential attacks and countermeasures can be considered for other protocols. Despite the fact that the attacks may not be fully mitigated, it may prove useful for protocol implementers to consider the integration of remote attestation as a means for providing stronger deniability. As a more comprehensive piece of work, a general framework for protocol deniability may be developed that leverages remote attestation available on the most common consumer platforms and can be integrated by protocol implementers seeking deniability.

This work also has important implications for electronic voting schemes. An important feature of electronic voting protocols is their being *receipt-free* [77], meaning that voters cannot carry a receipt proving the way their vote was cast, thereby preventing vote buying and coercion. By using remote attestation, it is possible to subvert the voting device in any electronic voting protocol to generate a receipt. If the information about how the vote was cast is available to the device, this receipt can link a vote to a voter, breaking the *receipt-free* property. Online voting protocols, in which votes may use their own devices, are particularly amenable to this attack. Future work should focus on potential mitigations to prevent surreptitious receipt generation.

Acknowledgements

I would like to extend special thanks to my advisor, Dr. Lachlan Gunn, for his untiring guidance throughout the completion of this work, as well for his useful feedback and reviewing several drafts of this thesis.

Some of the figures in this document feature the 'faceless man' and 'faceless woman' clip art images contributed by the user *Firkin* to Openclipart.

This work is supported in part by Intel (ICRI-CARS) and by the Academy of Finland (grant 309195).

References

[1] L. J. Gunn, R. I. Vieitez Parra and N. Asokan, 'On The Use of Remote Attestation to Break and Repair Deniability', 424, 2018. IACR: 2018/424.

[2] A. McCullagh and W. Caelli, 'Non-repudiation in the digital environment', *First Monday*, vol. 5, no. 8, 7th Aug. 2000, ISSN: 13960466. DOI: 10.5210/fm.v5i8.778.

[3] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, 'Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile', RFC Editor, RFC5280, May 2008. DOI: 10.17487/rfc5280.

[4] We Are Social. (). Most popular mobile messaging apps worldwide as of April 2018, based on number of monthly active users (in millions), [Online]. Available: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ (visited on 26/06/2018).

[5] WikiLeaks. (4th Nov. 2016). DKIM Verification, [Online]. Available: https://wikileaks.org/DKIM-Verification.html (visited on 01/07/2018).

[6] H. Enten, 'How Much Did WikiLeaks Hurt Hillary Clinton?', *FiveThirtyEight*, 23rd Dec. 2016. [Online]. Available: https://fivethirtyeight.com/features/wikileaks-hillary-clinton/ (visited on 01/07/2018).

[7] D. J. Bernstein, T. Lange and P. Schwabe, 'The Security Impact of a New Cryptographic Library', in *Progress in Cryptology – LATINCRYPT 2012*, A. Hevia and G. Neven, Eds., red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi and G. Weikum, vol. 7533, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 159–176, ISBN: 978-3-642-33480-1, ISBN (online): 978-3-642-33481-8. DOI: 10.1007/978-3-642-33481-8_9.

[8] M. J. Dworkin, 'Recommendation for block cipher modes of operation :: GaloisCounter Mode (GCM) and GMAC', National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-38d, 2007. DOI: 10.6028/NIST.SP.800-38d.

[9] 'Advanced Trusted Environment: OMTP TR1', OMTP Limited, 28th May 2009, v1.1. [Online]. Available: http://www.omtp.org/OMTP_Advanced_Trusted_Environment_OMTP_TR1_v1_1.pdf (visited on 29/05/2018).

[10] 'TEE System Architecture v1.1', GlobalPlatform, Inc., GPD_SPE_009, Jan. 2017.

[11] B. McGillion, T. Dettenborn, T. Nyman and N. Asokan, 'Open-TEE – An Open Virtual Trusted Execution Environment', in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 400–407. DOI: 10.1109/Trustcom.2015.400.

[12] M. Sabt, M. Achemlal and A. Bouabdallah, 'Trusted Execution Environment: What It is, and What It is Not', IEEE, Aug. 2015, pp. 57–64, ISBN: 978-1-4673-7952-6. DOI: 10.1109/Trustcom.2015.357.

[13] M. Marlinspike. (26th Sep. 2017). Technology Preview: Private Contact Discovery for Signal, [Online]. Available: https://signal.org/blog/private-contact-discovery/ (visited on 30/05/2018).

[14] S. Tamrakar, J. Liu, A. Paverd, J.-E. Ekberg, B. Pinkas and N. Asokan, 'The Circle Game: Scalable Private Membership Test Using Trusted Hardware', in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17, New York, NY, USA: ACM, 2017, pp. 31–44, ISBN: 978-1-4503-4944-4. DOI: 10.1145/3052973.3053006.

[15] J. P. Achara, G. Acs and C. Castelluccia, 'On the Unicity of Smartphone Applications', in *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, ser. WPES '15, New York, NY, USA: ACM, 2015, pp. 27–36, ISBN: 978-1-4503-3820-2. DOI: 10.1145/2808138.2808146.

[16] *Intel® 64 and IA-32 Architectures Software Developer's Manual*, 10 vols., System Programming Guide 4. Intel Corporation, May 2018, vol. 3D, Order Number: 332831-067US. [Online]. Available: https://software.intel.com/sites/default/files/managed/7c/f1/332831-sdm-vol-3d.pdf (visited on 25/05/2018).

[17] Intel Corporation, 'Intel® Software Guard Extensions (Intel® SGX)', Reference Number: 332680-001, Revision Number: 1.0, Jun. 2015, [Online]. Available: https://software.intel.com/sites/default/files/332680-001.pdf (visited on 22/05/2018).

[18] V. Costan and S. Devadas, 'Intel SGX Explained', 086, 2016. IACR: 2016/086.

[19] Intel Corporation. (4th Oct. 2016). Intel® Software Guard Extensions Commercial Licensing FAQ | Intel® Software, [Online]. Available: https://software.intel.com/en-us/articles/intel-software-guard-extensions-product-licensing-faq (visited on 26/06/2018).

[20] ——, 'Intel(R) Software Guard Extensions Developer Reference for Linux OS', Apr. 2018, Revision: 2.1.3.

[21] I. Anati, S. Gueron, S. P. Johnson and V. R. Scarlata, 'Innovative Technology for CPU Based Attestation and Sealing', Intel Corporation, 14th Aug. 2013.

[22] Intel Corporation, 'Intel® Software Guard Extensions Platform Software for Windows OS Release Notes', 2nd Oct. 2017, Revision: 1.8.5. [Online]. Available: https://downloadmirror.intel.com/27118/eng/Intel%20SGX%20Release%20Notes%201.8.106.40803.pdf (visited on 27/06/2018).

[23] ——, (8th Jul. 2016). Intel® Software Guard Extensions Remote Attestation End-to-End Example | Intel® Software, [Online]. Available: https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example (visited on 26/06/2018).

[24] ISO/IEC JTC 1, *ISO/IEC 10181-4:1997, Information Technology - Open Systems Interconnection - Security Frameworks for Open Systems: Non-Repudiation Framework - Part 4:* Multiple. Distributed through American National Standards Institute, 23rd Aug. 2007, 30 pp.

[25] W. Mao and K. G. Paterson, 'On The Plausible Deniability Feature of Internet Protocols', 2002.

[26] Y. Dodis, J. Katz, A. Smith and S. Walfish, 'Composability and On-Line Deniability of Authentication', in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 15th Mar. 2009, pp. 146–162, ISBN: 978-3-642-00456-8, ISBN (online): 978-3-642-00457-5. DOI: 10.1007/978-3-642-00457-5_10.

[27] M. Di Raimondo, R. Gennaro and H. Krawczyk, 'Deniable authentication and key exchange', ACM Press, 2006, p. 400, ISBN: 978-1-59593-518-2. DOI: 10.1145/1180405.1180454.

[28] Open Whisper Systems, 'Signal >> Home', 30th May 2018. [Online]. Available: https://www.signal.org/ (visited on 30/05/2018).

[29] D. J. Bernstein, 'Curve25519: New Diffie-Hellman Speed Records', in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias and T. Malkin, Eds., vol. 3958, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228, ISBN: 978-3-540-33851-2, ISBN (online): 978-3-540-33852-9. DOI: 10.1007/11745853_14.

[30] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback and J. F. Dray, 'Advanced Encryption Standard (AES)', National Institute of Standards and Technology, Gaithersburg, MD, NIST FIPS 197, Nov. 2001. DOI: 10.6028/NIST.FIPS.197.

[31] H. Krawczyk, M. Bellare and R. Canetti, 'HMAC: Keyed-Hashing for Message Authentication', RFC Editor, RFC2104, Feb. 1997. DOI: 10.17487/rfc2104.

[32] Q. H. Dang, 'Secure Hash Standard', National Institute of Standards and Technology, NIST FIPS 180-4, Jul. 2015. DOI: 10.6028/NIST.FIPS.180-4.

[33] (2018). Is it private? Can I trust it?, [Online]. Available: http://support.signal.org/hc/en-us/articles/212477768-Is-it-private-Can-I-trust-it- (visited on 30/05/2018).

[34] M. Marlinspike, 'The Double Ratchet Algorithm', 20th Nov. 2016.

[35] ——, (27th Jul. 2013). Simplifying OTR deniability, [Online]. Available: https://signal.org/blog/simplifying-otr-deniability/ (visited on 30/05/2018).

[36] ——, 'The X3DH Key Agreement Protocol', 4th Nov. 2016.

[37] Open Whisper Systems, 'Signal >> Documentation', 30th May 2018. [Online]. Available: https://www.signal.org/docs/ (visited on 30/05/2018).

[38] E. Rescorla, 'Diffie-Hellman Key Agreement Method', RFC Editor, RFC2631, Jun. 1999. DOI: 10.17487/rfc2631.

[39] A. J. Menezes, J. Katz, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, 1 edition. Boca Raton: CRC Press, 16th Dec. 1996, 780 pp., ISBN: 978-0-8493-8523-0.

[40] K. Cohn-Gordon, C. Cremers and L. Garratt, 'On Post-Compromise Security', 221, 2016. IACR: 2016/221.

[41] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt and D. Stebila, 'A Formal Security Analysis of the Signal Messaging Protocol', 1013, 2016. IACR: 2016/1013.

[42] D. Chaum, J.-H. Evertse and J. van de Graaf, 'An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations', in *Advances in Cryptology — EUROCRYPT' 87*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 13th Apr. 1987, pp. 127–141, ISBN: 978-3-540-19102-5, ISBN (online): 978-3-540-39118-0. DOI: 10.1007/3-540-39118-5_13.

[43] R. Canetti, Y. Dodis, R. Pass and S. Walfish, 'Universally Composable Security with Global Setup', in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 21st Feb. 2007, pp. 61–85, ISBN: 978-3-540-70935-0, ISBN (online): 978-3-540-70936-7. DOI: 10.1007/978-3-540-70936-7_4.

[44] G. H. Mealy, 'A method for synthesizing sequential circuits', *The Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, Sep. 1955, ISSN: 0005-8580. DOI: 10/gc3fv2.

[45] D. Brand and P. Zafiropulo, 'On Communicating Finite-State Machines', *J. ACM*, vol. 30, no. 2, pp. 323–342, Apr. 1983, ISSN: 0004-5411. DOI: 10/dw9xwr.

[46] D. P. Sidhu and T. K. Leung, 'Formal methods for protocol testing: A detailed study', *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 413–426, Apr. 1989, ISSN: 0098-5589. DOI: 10/cr5hrj.

[47] S. Schröder, M. Huber, D. Wind and C. Rottermanner, 'When SIGNAL Hits the Fan: On the Usability and Security of State-of-the-Art Secure Mobile Messaging', Internet Society, 2016, ISBN: 978-1-891562-45-7. DOI: 10.14722/eurousec.2016.23012.

[48] K. Ermoshina, F. Musiani and H. Halpin, 'End-to-End Encrypted Messaging Protocols: An Overview', in *Internet Science*, ser. Lecture Notes in Computer Science, Springer, Cham, 12th Sep. 2016, pp. 244–254, ISBN: 978-3-319-45981-3, ISBN (online): 978-3-319-45982-0. DOI: 10.1007/978-3-319-45982-0_22.

[49] B. Schneier. (28th Dec. 2016). How Signal Is Evading Censorship - Schneier on Security, [Online]. Available: https://www.schneier.com/blog/archives/2016/12/how_signal_is_e.html (visited on 23/06/2018).

[50] F. Valsorda. (6th Dec. 2016). I'm giving up on PGP, [Online]. Available: https://blog.filippo.io/giving-up-on-long-term-pgp/ (visited on 23/06/2018).

[51] G. Gebhart. (27th Mar. 2018). Why We Can't Give You A Recommendation, [Online]. Available: https://www.eff.org/deeplinks/2018/03/why-we-cant-give-you-recommendation (visited on 23/06/2018).

[52] R. Wyden, *Ron Wyden letter on Signal encrypted messaging*, in collab. with F. Larkin, Letter, 9th May 2017. [Online]. Available: https://www.documentcloud.org/documents/3723701-Ron-Wyden-letter-on-Signal-encrypted-messaging.html (visited on 23/06/2018).

[53] 'Linux-Sgx: Intel SGX for Linux', version 2.1.3, 10th May 2018. [Online]. Available: https://github.com/intel/linux-sgx (visited on 06/06/2018).

[54] 'Libsignal-Protocol-c: Signal Protocol C Library', version 2.3.2, 14th May 2018. [Online]. Available: https://github.com/signalapp/libsignal-protocol-c (visited on 06/06/2018).

[55] (). Clang Static Analyzer, [Online]. Available: https://clang-analyzer.llvm.org/ (visited on 29/06/2018).

[56] (). Cppcheck - A tool for static C/C++ code analysis, [Online]. Available: http://cppcheck.sourceforge.net/ (visited on 29/06/2018).

[57] A. Potapenko, ygribov-samsung, A. Samsonov, K. Serebryany, E. Stepanov, chefmax, V. Buka, H. Böck and M. Morehouse. (29th Jun. 2018). Sanitizers: AddressSanitizer, ThreadSanitizer, MemorySanitizer, [Online]. Available: https://github.com/google/sanitizers (visited on 29/06/2018).

[58] The Clang Team. (). UndefinedBehaviorSanitizer — Clang 7 documentation, [Online]. Available: https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html (visited on 29/06/2018).

[59] AsamK, 'Signal-Cli', version 0.6.0, 3rd May 2018. [Online]. Available: https://github.com/AsamK/signal-cli (visited on 07/06/2018).

[60] N. Unger and I. Goldberg, 'Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging', *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 1, 1st Jan. 2018, ISSN: 2299-0984. DOI: 10/gdp86s.

[61] A. M. Turing, 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937, ISSN: 00246115. DOI: 10/dcs37s.

[62] R. Canetti, C. Dwork, M. Naor and R. Ostrovsky, 'Deniable Encryption', in *Advances in Cryptology — CRYPTO '97*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 17th Aug. 1997, pp. 90–104, ISBN: 978-3-540-63384-6, ISBN (online): 978-3-540-69528-8. DOI: 10.1007/BFb0052229.

[63] M. Roe, 'Cryptography and Evidence', Ph.D. University of Cambridge, Apr. 1997, 74 pp.

[64] C. Dwork and A. Sahai, 'Concurrent zero-knowledge: Reducing the need for timing constraints', in *Advances in Cryptology — CRYPTO '98*, H. Krawczyk, Ed., red. by G. Goos, J. Hartmanis and J. van Leeuwen, vol. 1462, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 442–457, ISBN: 978-3-540-64892-5, ISBN (online): 978-3-540-68462-6. DOI: 10.1007/BFb0055746.

[65] L. Harn, C.-Y. Lee, C. Lin and C.-C. Chang, 'Fully Deniable Message Authentication Protocols Preserving Confidentiality', *The Computer Journal*, vol. 54, no. 10, pp. 1688–1699, 1st Oct. 2011, ISSN: 0010-4620, ISSN (online): 1460-2067. DOI: 10/d7fj99.

[66] N. Borisov, I. Goldberg and E. Brewer, 'Off-the-record communication, or, why not to use PGP', ACM Press, 2004, p. 77, ISBN: 978-1-58113-968-6. DOI: 10.1145/1029179.1029200.

[67] I. Pazmiño, S. Celi, Y. Dixon, R. de Souza, T. S, A. Cruz, N. Eskinazi, R. Tolentino and F. Torchz, 'Otrv4: Off-the-Record Messaging Protocol version 4', otrv4, 24th Jun. 2018. [Online]. Available: https://github.com/otrv4/otrv4 (visited on 24/06/2018).

[68] R. Tolentino, R. de Souza and S. Celi, 'ADR 8: Interactive DAKE', otrv4, 24th Jun. 2018. [Online]. Available: https://github.com/otrv4/otrv4 (visited on 24/06/2018).

[69] C. Johansen, A. Mujaj, H. Arshad and J. Noll, 'Comparing Implementations of Secure Messaging Protocols (long version)', University of Oslo, Research Report, 2017. [Online]. Available: http://urn.nb.no/URN:NBN:no-63579 (visited on 26/06/2018).

[70] K. Severinsen, C. Johansen and S. Bursuc, 'Securing the End-points of the Signal Protocol using Intel SGX based Containers', in *Security Principles and Trust Hotspot 2017*, Uppsala, Sweden, 23rd Apr. 2017, pp. 40–48. [Online]. Available: https://sec.uni-stuttgart.de/_media/events/hotspot2017/proceedings.pdf (visited on 26/06/2018).

[71] K. Severinsen, 'Secure Programming with Intel SGX and Novel Applications', Master's Thesis, University of Oslo, 2017, 87 pp. [Online]. Available: http://urn.nb.no/URN:NBN:no-62997 (visited on 26/06/2018).

[72] F. Zhang, E. Cecchetti, K. Croman, A. Juels and E. Shi, 'Town Crier: An Authenticated Data Feed for Smart Contracts', 168, 2016. IACR: 2016/168.

[73] N. Falliere, L. O. Murchu and E. Chien, 'W32.Stuxnet Dossier', Symantec Corp. Security Response, Cupertino, CA, USA, White paper, 5th Feb. 2011, Version 1.4. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf (visited on 21/06/2018).

[74] Y. Zhou and X. Jiang, 'Dissecting Android Malware: Characterization and Evolution', in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 95–109. DOI: 10.1109/SP.2012.16.

[75] Open Source Initiative. (22nd Mar. 2007). The Open Source Definition | Open Source Initiative, [Online]. Available: https://opensource.org/osd (visited on 23/06/2018).

[76] Free Software Foundation, Inc. (). What is free software?, [Online]. Available: https://www.gnu.org/philosophy/free-sw.html (visited on 23/06/2018).

[77] J. Benaloh and D. Tuinstra, 'Receipt-free Secret-ballot Elections (Extended Abstract)', in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94, New York, NY, USA: ACM, 1994, pp. 544–553, ISBN: 978-0-89791-663-9. DOI: 10.1145/195058.195407.

A

Attack Enclave Interface

Argument Notation

⟵OUTPUT	a fixed-size output value
INPUT⟶	a fixed-size input value
⟵▷OUTPUT BUFFER◁	a variable-size output value
▷INPUT BUFFER◁⟶	a variable-size input value
⟵▷🔒SEALED OUTPUT BUFFER◁	a variable-size output buffer used for storing sealed outputs
▷🔒SEALED INPUT BUFFER◁⟶	a variable-size input buffer used for reading sealed inputs

A.1 Signal Protocol Implementation

These functions implement operations that are part of the Signal protocol and which must be executed in a TEE to provide an unforgeable transcript.

`ecall_generate_pre_key`

Arguments

⟵▷🔒P◁	Serialised structure with the private signed pre-key of the local party and related pre-key metadata.
⟵▷P_{sc}◁	Serialised structure with the public pre-key of the local party and related signed pre-key metadata.
id⟶	A pre-key ID number.

Description

Generate a pre-key pair $\left(P, g^P\right)$ with an associated id $\overrightarrow{\texttt{id}}$. Store the pre-key structure in the buffer $\overleftarrow{\texttt{P}}$. Write zeros over the secret value P in the data structure and store it in the buffer $\overleftarrow{\texttt{P}_{\texttt{SC}}}$.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Wraps the `session_pre_key_create` and `session_pre_key_serialize` functions in *libsignal-protocol-c*.

`ecall_generate_signed_pre_key`

Arguments

$\overleftarrow{\triangleright S \triangleleft}$	Serialised structure with the private signed pre-key of the local party and related signed pre-key metadata.
$\overleftarrow{\triangleright S_{\texttt{SC}} \triangleleft}$	Serialised structure with the public signed pre-key of the local party and related signed pre-key metadata.
$\overrightarrow{\texttt{id}}$	A signed pre-key ID number.
$\overrightarrow{\texttt{timestamp}}$	Current Unix time.
$\overrightarrow{\triangleright I \triangleleft}$	Serialised structure with the private identity key of the local party.

Description

Generate a signed pre-key pair $\left(S, g^S\right)$ with an associated id $\overrightarrow{\texttt{id}}$ and timestamp $\overrightarrow{\texttt{timestamp}}$, then sign it using $\overrightarrow{\texttt{I}}$. Store the signed pre-key structure in the buffer $\overleftarrow{\triangleright \texttt{S} \triangleleft}$. Write zeros over the secret value S in the data structure and store it in the buffer $\overleftarrow{\triangleright \texttt{S}_{\texttt{SC}} \triangleleft}$.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Wraps the `session_signed_pre_key_create` and `session_signed_pre_key_serialize` functions in *libsignal-protocol-c*.

`ecall_session_builder_process_pre_key_signal_message`

Arguments

$\overleftarrow{\triangleright SR_{n+1} \triangleleft}$	Sealed session record, corresponding to the next session state.

$\overrightarrow{g^{I_B}}$	Structure with the public identity key of the recipient.
$\overrightarrow{g^{R_B}}$	Structure with the public ratchet key of the recipient.
$\overrightarrow{I_B}$	Serialised structure with the private identity key of the recipient.
$\overrightarrow{S_B}$	Serialised structure with the private signed pre-key of the recipient and related metadata.
$\overrightarrow{P_B}$	Serialised structure with the private pre-key of the recipient and related metadata.
$\overrightarrow{\texttt{RID}}$	The registration number assigned to the sender.
$\overrightarrow{\texttt{LID}}$	The registration number assigned to the recipient.
$\overrightarrow{SR_n}$	Sealed session record, corresponding to the current session state.

Description

Used to process the information in a pre-key message, which typically is the first message received upon establishing a session. This function takes inputs necessary for the performing an X3DH handshake, including some metadata like the local registration ID $\overrightarrow{\texttt{LID}}$ and the remote registration ID $\overrightarrow{\texttt{LID}}$. Given a session record $\overrightarrow{SR_n}$, the function updates it with the necessary session information and places the result in $\overleftarrow{SR_{n+1}}$. The internal hash state H is also updated.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Roughly corresponds to the internal `session_builder_process_pre_key_signal_message` function in *libsignal-protocol-c*.

`ecall_session_builder_process_pre_key_bundle`

Arguments

$\overleftarrow{SR_{n+1}}$	Sealed session record, corresponding to the next session state.
$\overrightarrow{g^{I_B}}$	Structure with the public identity key of the recipient.
$\overrightarrow{g^{S_B}}$	Structure with the public signed pre-key of the recipient.
$\overrightarrow{g^{P_B}}$	Structure with a public pre-key of the recipient (optional).
$\overrightarrow{g^{P_B}_{\texttt{id}}}$	Recipient's pre-key ID number (optional).

$\overrightarrow{g_{\mathtt{id}}^{S_B}}$	Recipient's signed pre-key ID number.
$\overrightarrow{\triangleright I_A \triangleleft}$	Serialised structure with the private identity key of the sender.
$\overrightarrow{\mathtt{RID}}$	The registration number assigned to the recipient.
$\overrightarrow{\mathtt{LID}}$	The registration number assigned to the sender.
$\overrightarrow{\triangleright SR_n \triangleleft}$	Sealed session record, corresponding to the current session state.

Description

Used to process the information in a pre-key bundle obtained from the Signal service, which is used for establishing a new Signal session. This function takes inputs necessary for the performing an X3DH handshake, including some metadata like the pre-key id $\overrightarrow{g_{\mathtt{id}}^{P_B}}$, the local registration ID $\overrightarrow{\mathtt{LID}}$ and the remote registration ID $\overrightarrow{\mathtt{LID}}$, and is typically called on an empty session record. Given a session record $\overrightarrow{\triangleright SR_n \triangleleft}$, the function updates it with the necessary session information and places the result in $\overleftarrow{\triangleright SR_{n+1} \triangleleft}$. The internal hash state H is also updated.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Roughly corresponds to the `session_builder_process_pre_key_bundle` function in *libsignal-protocol-c*.

`ecall_session_cipher_encrypt_message_from_record`

Arguments

$\overleftarrow{\triangleright \mathcal{C} \triangleleft}$	Message ciphertext.
$\overleftarrow{\mathcal{C}_{\mathtt{type}}}$	Output message type in the Signal protocol. It can be a pre-key Signal message or a Signal message.
$\overleftarrow{\triangleright SR_{n+1} \triangleleft}$	Sealed session record, corresponding to the next session state.
$\overrightarrow{\triangleright \mathcal{M} \triangleleft}$	Message plaintext.
$\overrightarrow{\triangleright SR_n \triangleleft}$	Sealed session record, corresponding to the current session state.

Description

Given a plaintext message in $\overrightarrow{\triangleright \mathcal{M} \triangleleft}$ and a session record in $\overrightarrow{\triangleright SR_n \triangleleft}$, attempt to encrypt the message and store it in $\overleftarrow{\triangleright \mathcal{C} \triangleleft}$. If the operation is successful, the Signal message type is

also stored in $\overleftarrow{\mathcal{C}_{\texttt{type}}}$. The function performs a symmetric ratchet step, and the updated session record is stored in $\overleftarrow{\blacktriangleright\blacksquare SR_{n+1}\blacktriangleleft}$. The internal hash state H is updated.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Roughly corresponds to the `session_cipher_encrypt` function in *libsignal-protocol-c*.

`ecall_session_cipher_decrypt_message_from_record`

Arguments

$\overleftarrow{\blacktriangleright\mathcal{M}\blacktriangleleft}$	Message plaintext.
$\overleftarrow{\blacktriangleright\blacksquare SR_{n+1}\blacktriangleleft}$	Sealed session record, corresponding to the next session state.
$\overrightarrow{\blacktriangleright\mathcal{C}\blacktriangleleft}$	Message ciphertext.
$\overrightarrow{\blacktriangleright\blacksquare SR_{n}\blacktriangleleft}$	Sealed session record, corresponding to the current session state.

Description

Given a plaintext message in $\overrightarrow{\blacktriangleright\mathcal{C}\blacktriangleleft}$ and a session record in $\overrightarrow{\blacktriangleright\blacksquare SR_{n}\blacktriangleleft}$, attempt to decrypt the message and store it in $\overleftarrow{\blacktriangleright\mathcal{M}\blacktriangleleft}$. The function performs a symmetric ratchet step or an asymmetric ratchet step, and the updated session record is stored in $\overleftarrow{\blacktriangleright\blacksquare SR_{n+1}\blacktriangleleft}$. The internal hash state H is updated.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Roughly corresponds to the `session_cipher_decrypt` function in *libsignal-protocol-c*.

`ecall_session_record_create`

Arguments

$\overleftarrow{\blacktriangleright\blacksquare SR_{0}\blacktriangleleft}$	Empty session record.

Description

Generate an empty session record and store it in $\overleftarrow{\blacktriangleright\blacksquare SR_{0}\blacktriangleleft}$.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Wraps the `session_record_create` and `session_record_serialize` functions in *libsignal-protocol-c*.

A.2 Attestation-specific Methods

These functions handle operations strictly related to attestation and do handle the Signal protocol.

`ecall_get_public_signing_key`

Arguments

$\triangleright\overleftarrow{g^E}\triangleleft$	The enclave public signing key.

Description

Obtain g^E, the public key corresponding to a private key E generated in the enclave and used for digital signatures. g^E is stored in the untrusted memory region pointed by $\overleftarrow{g^E}$. This is an optional feature.

Return Value `0` on success; a non-zero value on failure.

`ecall_set_la_target_info`

Arguments

$\overrightarrow{\text{TIL}}$	A TARGETINFO data structure containing information about an enclave used for local attestation.

Description

Store the identity $\overrightarrow{\text{TIL}}$ of a quoting enclave. This identity is used to produce local attestations. This is an optional feature.

Return Value `0` on success; a non-zero value on failure.

`ecall_set_ra_target_info`

Arguments

$\overrightarrow{\texttt{TIR}}$	A TARGETINFO data structure containing information about the quoting enclave.

Description

Store the identity $\overrightarrow{\texttt{TIR}}$ of a quoting enclave. This identity is used to produce remote attestations.

Return Value `0` on success; a non-zero value on failure.

`ecall_end_session`

Arguments

$\overleftarrow{\sigma}$	A data structure containing attestations and signatures of the internal hash state H.
$\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$	Sealed session record, corresponding to the current session state.

Description

Given a session record in the buffer $\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$, store an attested proof in $\overleftarrow{\sigma}$ based on the internal hash state H that can be included in transcripts to verify their authenticity. Depending on the options used, the proof may contain a local attestation (using `TIL`), a signature (using E) and a report that can be converted to a remote attestation (using `TIR`).

Return Value `0` on success; a non-zero value on failure.

A.3 State Querying

These functions are used for querying information stored in the sealed session record from the Signal client. Even though the information they provide is necessary for the client to adequately function, they are not a strictly necessary part of the enclave. An alternate enclave implementation may provide an unsealed version session record containing only this information, which would avoid the need for providing these functions.

`ecall_session_record_get_session_version`

Arguments

$\overleftarrow{\texttt{version}}$	Signal protocol version.

$\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$	Sealed session record, corresponding to the current session state.

Description

Given a session record in the buffer $\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$, store the version of the Signal protocol used in a session in $\overleftarrow{\texttt{version}}$.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Wraps the `session_cipher_get_session_version` function in *libsignal-protocol-c*.

`ecall_session_record_get_remote_registration_id`

Arguments

$\overleftarrow{\texttt{RID}}$	The registration number assigned to the remote party.
$\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$	Sealed session record, corresponding to the current session state.

Description

Given a session record in the buffer $\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$, store the value of the remote registration ID in a session in $\overleftarrow{\texttt{rid}}$. The remote registration ID corresponds to a unique identifier of the other party to a session assigned by the Signal service.

Return Value `0` on success; a non-zero value on failure.

***libsignal-client-c* Functionality** Wraps the `session_cipher_get_remote_registration_id` function in *libsignal-protocol-c*.

`ecall_session_record_has_session_state`

Arguments

$\overrightarrow{\texttt{version}}$	Signal message version
$\overrightarrow{\triangleright g^R\triangleleft}$	The public ratchet key of the remote party.
$\overrightarrow{\triangleright\text{🔒}SR_n\triangleleft}$	Sealed session record, corresponding to the current session state.

Description

Given a session record in the buffer $\overrightarrow{\blacktriangleright\text{🔒}SR_n\blacktriangleleft}$, a message version $\overrightarrow{\texttt{version}}$ and a remote party ratchet key $\overrightarrow{g^R}$, return `1` if the receiving chain has a state for the $\overrightarrow{g^R}$ key and `0` otherwise.

Return Value `0` or `1` on success; a negative value on failure.

***libsignal-client-c* Functionality** Wraps the `session_record_has_session_state` function in *libsignal-protocol-c*.

B

Defence Enclave Interface

Argument Notation

$\overleftarrow{\texttt{OUTPUT}}$ a fixed-size output value
$\overrightarrow{\texttt{INPUT}}$ a fixed-size input value

B.1 Entry Points

`ecall_get_public_ex3dh_key`

Arguments

$\overleftarrow{g^p}$	The enclave public EX3DH key of the local party.

Description

Obtain g^p, the public key corresponding to a private key p generated in the enclave and used to compute the EX3DH agreement. g^p is stored in the untrusted memory region pointed by $\overleftarrow{g^p}$.

Return Value `0` on success; a non-zero value on failure.

`ecall_get_public_ex3dh_key_report`

Arguments

$\overleftarrow{\sigma}$	A local attestation report.
$\overrightarrow{\mathrm{TI}}$	A TARGETINFO data structure containing information about an enclave used for local attestation.

Description

Obtain a local attestation report on g^p, the public key corresponding to a private key p generated in the enclave and used to compute the EX3DH agreement, for an enclave with an identity $\overrightarrow{\mathrm{TI}}$. The attestation report is stored in the untrusted memory region pointed by $\overleftarrow{\sigma}$. If $\overrightarrow{\mathrm{TI}}$ contains information about the quoting enclave, the resulting report may be used for remote attestation.

Return Value `0` on success; a non-zero value on failure.

`ecall_get_public_signing_key`

Arguments

$\overleftarrow{\triangleright g^E \triangleleft}$	The enclave public signing key.

Description

Obtain g^E, the public key corresponding to a private key E generated in the enclave and used for digital signatures. g^E is stored in the untrusted memory region pointed by $\overleftarrow{g^E}$. This is an optional feature.

Return Value `0` on success; a non-zero value on failure.

`ecall_get_public_ex3dh_key_signature`

Arguments

$\overleftarrow{\sigma}$	A digital signature.

Description

Obtain a signature σ of g^p, the public key corresponding to a private key p generated in the enclave and used to compute the EX3DH agreement. σ is generated using the private key E and is stored in the untrusted memory region pointed by $\overleftarrow{\sigma}$. This is an optional feature.

Return Value `0` on success; a non-zero value on failure.

ecall_get_ex3dh_key_material_sending

Arguments

$\overleftarrow{L_A^\star}$	Value used for deriving root keys.
$\overrightarrow{R_A}$	The private ratchet key of the local party.
$\overrightarrow{g^{R_B}}$	The public ratchet key of the remote party.
$\overrightarrow{g^{p_B}}$	The enclave public EX3DH key of the remote party.

Description

Compute an EX3DH handshake $L_A^\star = \left(\left(g^{R_B}\right)^{R_A}, \left(g^{R_B}\right)^{p}, \left(g^{p_B}\right)^{R_A}\right)$ and place the result in the untrusted memory region pointed by $\overleftarrow{L_A^\star}$.

Return Value `0` on success; a non-zero value on failure.

ecall_get_ex3dh_key_material_receiving

Arguments

$\overleftarrow{L_B^\star}$	Value used for deriving root keys.
$\overrightarrow{R_B}$	The private ratchet key of the local party.
$\overrightarrow{g^{R_A}}$	The public ratchet key of the remote party.
$\overrightarrow{g^{p_A}}$	The enclave public EX3DH key of the remote party.

Description

Compute an EX3DH handshake $L_B^\star = \left(\left(g^{R_A}\right)^{R_B}, \left(g^{p_A}\right)^{R_B}, \left(g^{R_A}\right)^{p}\right)$ and place the result in the untrusted memory region pointed by $\overleftarrow{L_B^\star}$.

Return Value `0` on success; a non-zero value on failure.