Rheinland-Pfälzische
Technische Universität
Kaiserslautern
Landau

R. I. P. B. B. Siriwardana

Secure Communication in Dolphin: Integrating Trusted Execution Environments for Enhanced Caller Privacy

MASTER'S THESIS
to achieve the university degree of
Master of Science

submitted to
RPTU Kaiserslautern-Landau

Supervisor
Dr. Stefanie Roos
RPTU Kaiserslautern-Landau
Distributed and Network Systems Group

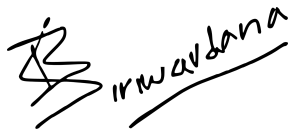
Kaiserslautern, June 2024

Labor omnia vincit (Work conquers all) – Virgil

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material that has been quoted either literally or by content from the sources used. The text document uploaded to the RPTU Kaiserslautern is identical to the present thesis.

I would also like to acknowledge the tools that aided in the writing and editing process, including Grammarly and ChatGPT. These tools were used solely for proofreading, grammar checking, and enhancing the academic style of the content I wrote.

28.06.2024	
Date	Signature

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Stephanie Roos, for her unwavering support, invaluable guidance, and continuous encouragement throughout this research. I should also thank Piyush Kumar, one of the authors of the original Dolphin paper for always helping me out with clarifications on the original Dolphin setup. I am also deeply thankful to my student supervisor, Yu Shen, for his insightful feedback and dedicated assistance. Additionally, I would like to acknowledge three key resources that were immensely helpful during the implementation phase: AWS Nitro Enclaves Samples by AWS, TLS Client Handshake in Pure Python by Neal Yip, and Nitro Enclave Python Demo by Richard Fan. I am sincerely grateful to RPTU Kaiserslautern-Landau for providing a stimulating academic environment and offering courses that positively impacted my thesis work. I extend my heartfelt thanks to the German education system for providing free education and scholarships, enabling me to pursue my studies without financial burden. Additionally, I am thankful to the University of Moratuwa, Sri Lanka for building my foundational knowledge and to the Sri Lankan free education system for supporting my education from grade one through my bachelor's degree. I would also like to thank my colleagues and friends who were always open to discussions regarding various parts of this thesis. Your support and insights were invaluable. Lastly, my deepest appreciation goes to my family for their unconditional love, patience, and unwavering belief in me throughout this journey. Your support has never decreased. Thank you all for your unwavering support and encouragement.

Kaiserslautern, June 2024
Bhanuka

Abstract

Censorship is a significant global issue, with authorities frequently restricting internet access to control the dissemination of information. Previous research, such as the Dolphin project, has provided means of accessing the internet during shutdowns. Dolphin enables data transmission via cellular voice channels; however, it depends on the callee's trustworthiness, posing privacy risks. This thesis addresses the challenge of maintaining private, confidential, and secure communication during internet blackouts—periods when conventional internet-based technologies are rendered ineffective—by building on and extending the Dolphin implementation. Specifically, it proposes augmenting Dolphin by incorporating Trusted Execution Environments (TEEs) on the callee's side. TEEs create secure, isolated environments that safeguard data confidentiality and integrity, reducing the threat posed by potentially malicious applications or users. The proposed method involves transmitting login credentials and message content to a TEE application, which then securely signs in and relays the message to the intended public server. By performing crucial operations within the TEE, the solution enhances security. Although side-channel attacks on TEEs are acknowledged, they are beyond the scope of this research. The study will investigate potential vulnerabilities, assess the effectiveness of attestation mechanisms in verifying program integrity, and analyze the performance impact of integrating TEEs with Dolphin. The objective is to achieve complete privacy, ensuring confidentiality and integrity for the caller, while also maintaining convenience, usability, and performance. Within the stated assumptions, the proposed solution exhibits robust defenses against various attack vectors that a callee might employ to covertly extract confidential information from the caller. Experimental results demonstrate that the additional average time incurred by integrating the proposed extension is between 1-2 seconds, depending on the message size and network latency. This slight delay ensures that the overall communication process remains efficient, maintaining the usability and convenience of the Dolphin system while offering enhanced privacy guarantees.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Censorship is a pervasive issue in the modern world, with national or state-level censorship being carried out by authorities to suppress or restrict information and communication. The internet, being a platform for free expression and the exchange of ideas, is often the target of censorship. Authorities employ tactics such as restricting access to specific online resources[1] or enacting complete internet shutdowns to control free speech and hinder civil movements against governments[2]. The severity of censorship depends on the extent of collateral damage the authorities are prepared to tolerate. A comprehensive and intentional blackout represents an extreme form of censorship, as demonstrated in various regions [3] [4] [5] [6] [7].

The challenge of privacy-preserving communication in the absence of internet access is a pressing concern, as many common technologies and even many popular anonymity networks rely entirely on internet connectivity [8][9]. Even censorship-resistant technologies like decoy routing minimally require at least some sort of access to the internet in order to access covert resources. Within the existing literature, two main categories of solutions emerge: mobile-based ad-hoc networks [10] [11] [12] for internal communication and delay-tolerant internet accessibility mechanisms [13]. Notable examples of ad-hoc networks include Moby [10], Rangzen [11], and Bridgefy [12]. However, there is limited literature on internet accessibility mechanisms during an internet shutdown, with Dolphin [13] being a significant contribution. Dolphin utilizes the cellular voice channel to transmit data between internet shutdown zones and locations with internet connectivity, empowering users without internet access to utilize essential applications like Twitter and email for data transmission. This research aims to address Dolphin's vulnerability related to caller privacy and provide a solution for internet access during an internet blackout that also ensures confidentiality of user information.

1.2 Problem Definition

Currently, accessing the internet during an internet blackout continues to be a research challenge without a clear solution. While Dolphin provides a comprehensive remedy, it necessitates that the caller in the affected area trusts the callee outside of it, ensuring that

they do not attempt to collect the caller's login credentials, extract the message the caller is trying to send to the intended recipient, or misuse their social media accounts.

Although the Dolphin paper assumes that the peer outside of the shutdown area is trustworthy, it provides suggestions to further protect the privacy of the accounts. One alternative is to use SMS-based two-factor authorization, enabling the caller to ensure that the callee can only access a session that the caller initiated. Another solution suggested in the paper is to use auth tokens with fine-grained access permissions, limiting the callee's potentially harmful activities to a certain extent. However, none of these solutions are ideal, as password leakage could have severe consequences, and allowing an outsider into one's social media or email could be viewed as a significant breach of privacy. The Dolphin authors have also explored solutions like password sharing among multiple parties, such as Mailet [14], an unobservable transport proxy, to address this issue. However, Mailet is more of a standalone solution that introduces an additional intermediate layer between the caller and the desired service, and is therefore not a convenient solution. It is essential to note that Mailet serves as a reliable solution to the risk of password leakage. However, it is important to keep in mind that Mailet cannot guarantee preventing the callee from accessing the social website while logged in. The reason behind this is that Mailet is designed to be a standalone solution, and when used as an intermediate hop, it recognizes the callee as the user, making it impossible to differentiate between the actual user and the callee.

Therefore, the need for an efficient, secure in terms of confidentiality, and convenient solution to preserve caller privacy in Dolphin continues to be an open problem.

1.3 Proposed Solution

Over the past decade, Trusted Execution Environments (TEE) have become increasingly popular as a means of enhancing confidentiality and integrity of data and code [15] [16]. TEE refers to a secure, isolated processing environment where applications can be executed securely, ensuring confidentiality and integrity of both the data and code running inside the TEE [17] [18], even in the presence of potential vulnerabilities in the rest of the system. TEEs are designed and considered to be able to mostly prevent unauthorized access, alteration or modification of the data and code stored within it. This includes protecting against unauthorized parties, including the owner of the machine or the rest of the machine it may be on. A significant threat to TEEs is side-channel attacks. However, for the purposes of this research, these attacks and potential defenses are considered out of scope.

In this approach, the callee is viewed as a stealthy and potentially deceitful entity. The callee may independently seek to extract confidential information from the caller for unauthorized purposes, such as gaining access to the caller's digital accounts. Describing the callee as stealthy indicates that the callee aims to extract confidential information from the caller without the caller realizing it, acting secretly while maintaining a facade of cooperation. Therefore, while they may facilitate the caller's communication, following communication protocols, they could still harbor malicious intentions, such as stealing login credentials, exploiting login sessions, extracting communicated information, or misusing the caller's accounts. They might also attempt to decrypt and manipulate messages if the opportunity

arises. It's important to note that the callee does not aim to prevent the caller's communication. If that were the goal, the simplest method would be to ignore the call initiated by the caller. This would either force the caller to abandon internet-based communication during an internet blackout or find an alternative party in an internet-accessible zone to facilitate their communication attempts.

The proposed solution involves extending the existing Dolphin system by incorporating a TEE on the callee's side also referred to as the server side. This approach can be divided into two main parts. First, the message body and login credentials are communicated to an application running within the TEE, ensuring the confidentiality of the data. Second, the TEE application signs in to the intended public server on behalf of the caller using the received credentials and forwards the message body. This signing in and subsequent request to send the message body must be conducted in a manner that secures the integrity and confidentiality of the information. The communication between the TEE and the desired public server is conducted securely through a TLS encrypted channel established between the callee machine and the web service under the orchestration of the TEE application. The extension proposed in this research thesis for the original Dolphin setup ensures that critical processes—such as encryption key generation, decryption of received content, TLS key generation for secure channel establishment with the public server, and encryption/decryption for TLS communication—are all performed entirely within the TEE. By confining these critical operations within a TEE, the proposed extension aims to enhance the security and privacy of the caller to an extent that the confidentiality of the caller's login credentials and the message body would sustain even if the callee is colluding. This is the primary objective of this research. Additionally, the extension integrates an automated mechanism for sharing login information, eliminating the need for manual retrieval and input of login credentials to the intended web service APIs, at the callee side. This system would decrypt the caller's social media credentials within a TEE at the receiving end, ensuring the confidentiality of the information. It is worth noting that credential sharing was not explicitly addressed in the original implementation of Dolphin. All these issues described above, including the lack of an explicit credential sharing mechanism, can be effectively resolved or mitigated by designing a solution based around TEEs for Dolphin.

The use of TEEs like Intel SGX and AWS Nitro Enclaves allow for the production of an enclave measurement, which is essentially a hash of the program. An additional objective of the proposed approach, it is expected to provide a means of attestation for the caller using this affirmation mechanism, allowing them to verify the integrity of the program they are communicating with. The attestation document serves as proof, confirming to the caller that the callee is indeed running the expected code in the specified manner. The expected setup involves secure components operating within the TEE, while an application in the untrusted part of the system facilitates communication between the caller and the callee. The attestation document is intended to verify this configuration, ensuring that the callee is running the code as expected in the secure environment.

The aim is to attain complete privacy in terms of confidentiality and integrity for the caller while upholding convenience, usability, and performance. In pursuit of this objective, a comprehensive examination will be conducted on potential vulnerabilities that could en-

able information extraction by the callee. Consequently, all possible attacks that the callee might attempt to extract confidential information will be considered, and the robustness or vulnerabilities of the proposed extension in such scenarios will be discussed. The success of the proposed solution in this thesis will be evaluated based on the outcomes of this inspection. Furthermore, the effectiveness of the attestation mechanism in allowing the caller to verify the integrity of the program running on the callee's side will be explored. The present study also evaluates the performance impact of introducing TEEs on Dolphin in terms of data exchange rate. This investigation is carried out to comprehend the benefits and drawbacks of employing TEEs in Dolphin.

The current implementation of the solution utilizes AWS (Amazon Web Services) Nitro Enclaves as the TEE, providing a secure environment for executing sensitive operations. To assess the performance impact of the proposed extension, the newly introduced components were isolated from the original Dolphin setup. The time taken from the initiation of the key exchange to the receipt of the intended public server's response was measured over multiple rounds for various message sizes. The average values per message size were then calculated to determine the impact on performance.

Various forms of attacks that are within the scope were considered, and the proposed solution demonstrates robustness against the considered attacking strategies while staying within the assumptions made. Experimental results indicate an average of 2 additional seconds for performing the extra communication rounds—messages that would not be communicated in the original Dolphin setup—for messages under 1024 characters. Thus, the findings suggest that the proposed solution significantly enhances the caller's confidentiality with negligible impact on communication time. This ensures that the improved security does not compromise the system's usability and convenience.

CHAPTER 2

Background and Related Work

2.1 Anonymous Communication

Anonymity can be maintained during communication when the identity of involved parties is concealed. Anonymous communication primarily aims to obscure network layer information, such as IP addresses and routing details, to prevent the identification or tracing of users. While it indirectly protects personally identifiable information, such as location or identity, the main goal is to ensure that users' interactions remain untraceable at the network layer by a censor or other third parties. The primary objective of anonymous communication is to prevent such information from being divulged to those who should not have access to it.

Onion routing [19] is a versatile technology designed to enable anonymous communication on a public network. It does so by establishing encrypted connections that are successively relayed through a series of intermediary nodes, known as "onion routers." At each node, a layer of encryption is peeled away, much like the layers of an onion, until the message reaches its intended recipient. The preeminent and extensively employed onion routing network is Tor (The Onion Router) [8]. Additional privacy-focused technologies include Freenet, now recognized as HyphaNet [9], which operates on a decentralized architecture but does not utilize onion routing. Another notable network is the Invisible Internet Project [20], which employs a variant of onion routing known as garlic routing [21]. In garlic routing, messages with individual delivery instructions are bundled together, forming what is referred to as a "garlic clove" or "bulb." This unique bundling mechanism sets it apart from Tor. However both onion routing and garlic routing utilize a layered encryption mechanism. By combining messages, garlic routing complicates traffic analysis, making it harder to infer information from the data. Additionally, this bundling approach has the potential to enhance data transfer speeds. It is important to note that all of these digital platforms are heavily reliant on the presence of a stable internet connection. This dependency can pose significant challenges in circumstances where access to the internet may not always be guaranteed, leading to potential disruptions in the communication process.

2.2 Censorship Resistance

Censorship refers to the restriction of access to resources or specific information. There are three main categories of censorship: limiting access to certain services on the international internet while leaving other services available [22], cutting off international internet access while the national internet remains accessible [2], and a complete shutdown of the internet. A deliberate and total blackout is considered an extreme form of censorship, as exemplified in regions such as Ethiopia [3], Iraq [4], the Republic of Congo [5], Syria [6], and specific areas in India [7]. The goal of censorship is to impede the flow of information through a particular system. Censors may achieve this by tampering with information by altering or deleting it, or by introducing false information. They may also block access to or prevent the publication of information [23]. Censorship Resistance Systems (CRS) are a crucial component in the battle for online freedom of information. As described in a seminal study by Khattak et al. [24], CRS applications are designed to circumvent censorship measures implemented by governments or other entities. These systems play an important role in ensuring that users have access to information that may be deemed sensitive or controversial, but which is nonetheless essential for a well-functioning democratic society. By enabling users to bypass censorship blocks and access information freely, CRS applications are a key tool in the fight against online censorship.

2.2.1 Pluggable Transports, Bridges

One of the popular tools for anonymous communication is the Tor network [8]. If the service had exclusively relied on onion routing to enable access to restricted resources, it would have been vulnerable to IP-based and pattern-based blacklisting by a powerful censoring entity. In such a situation, the censor could choose to prohibit access to the Tor website if it functioned on a solitary, identifiable public IP address. Alternatively, the censor could implement measures to impede traffic that resembled Tor access patterns. To overcome these obstacles, Tor utilizes bridges and pluggable transports. Bridges are confidential Tor relays that are not visible in the global directory, thus making it arduous for a censor to hinder access to all Tor gateway nodes as the IP addresses to block are unknown. Pluggable transports involve disguising or altering the traffic pattern of an application to mimic another application or a random access pattern [25] [26] [27].

2.2.2 Decoy Routing

Decoy Routing [28], also known as end-to-middle (E2M) proxying, is a highly effective method for accessing content from restricted or censored websites while maintaining an HTTPS connection with an unblocked site. This technique utilizes decoy routers to ensure that even in the face of a formidable censor employing deep packet inspection, censorship resistance is maintained. In decoy routing, users access an uncensored website, also known as the overt site, in a manner where the traffic passes through a friendly relay, commonly referred to as a decoy router. This router identifies the user's true intentions by recognizing a tag embedded in the handshake message. Subsequently, the decoy router establishes a proxy connection with the covert site secretly indicated by the user's tag. Slitheen [29] is an innovative decoy routing

mechanism specifically engineered to exclusively replace the leaf content of the overt website with the content of the covert website. In this context, any content lacking links from the overt web page is substituted with the content from the covert website. This approach effectively emulates the traffic and access patterns of the overt site, thereby increasing the difficulty of detection by deep packet pattern inspection mechanisms, as the content is end-to-end encrypted through the TLS connection. Another variant of decoy routing is demonstrated in the work of Daniel Ellard et al. [30], known as Rebound. Unlike traditional decoy routing approaches, Rebound does not rely on the assumption that both inbound and outbound traffic must pass through the decoy router. Instead, it accommodates asymmetric routes by utilizing HTTP error messages to efficiently relay information back to the client through the decoy host. There are several other decoy routing solutions that differ in how they pass the secret message to the decoy router and relay data back to the client.

2.3 Internet Blackouts

As previously discussed, internet shutdowns, also referred to as blackouts, represent an extreme form of censorship, impacting a virtually indispensable utility in the modern era. Despite the significant collateral damage incurred by those implementing internet blackouts, recent reports [31] [32] reveal a rising trend, with 187 recorded incidents in 35 countries in 2022 alone. The primary motive behind these shutdowns often revolves around suppressing state protests and political movements, with a lesser number of occurrences tied to specific events such as crucial nationwide exams or elections. It's noteworthy that incidents related to hardware or technical failures are excluded from the scope of this thesis discussion.

The prevalence of internet shutdowns has reached a point where a dedicated campaign, #KeepItOn [33], has been initiated to combat and raise awareness about this issue. Managed by Felicia Anthonio [34] and Kassem Mnejja [35] at Access Now, a non-profit organization focusing on digital civil rights since its founding in 2009, the campaign highlights that "Governments wield internet shutdowns as weapons of control and shields of impunity." Additionally, John Graham-Cumming, the CTO of Cloudflare [36], observes a growing tendency among governments to rely on intermittent and recurring shutdowns, predicting that this trend is likely to persist.

2.4 Internet Blackout Resistant Communication

An important characteristic shared by the aforementioned censorship-resistant and anonymous communication technologies is their reliance on a stable internet connection. This dependency is justified given that their functionalities and internal mechanisms are intricately interconnected with the internet, making them apt and effective in various scenarios. However, it is essential to note that in cases of complete internet blackouts—which, admittedly, are not the most common or widely adopted censorship strategies—these technologies become impractical and lose their utility. Within the existing literature, two main categories of solutions emerge: mobile-based ad-hoc networks for internal communication and delay-tolerant

internet accessibility mechanisms.

2.4.1 Mobile Ad-hoc Networks

While most of the systems outlined below effectively address the threat models they are designed for, their primary focus lies in establishing a communication medium that ensures sender anonymity and resilience against intended Denial-of-Service (DoS) attacks within an internet-free zone. Undoubtedly, this capability is crucial. However, these systems lack the functionality to communicate with external parties. In practical terms, if a government opts to shut down the internet to disrupt a nationwide protest, civil movement leaders may employ these systems to communicate internally but would be unable to disseminate information about potential government misconduct to the wider world.

Moby: A Blackout-Resistant Anonymity Network for Mobile Devices

Moby [10] represents a blackout-resistant anonymity network designed for secure message communication, offering features such as end-to-end encryption, sender anonymity, and forward secrecy for messages. The system employs a trust-based model to thwart Denial of Service (DoS) attacks by the censor, introducing a layer of defense against fraudulent users. Moby computes a trust score between two users, considering the nature of their direct communication and shared connections. This trust score plays a pivotal role in routing decisions and prioritization. When users exchange messages, Moby utilizes their trust scores to determine message forwarding. In close proximity, two Moby users establish a connection through Bluetooth for initial handshakes and message exchange, guided by their trust scores (notably, not all messages are forwarded). Intermediate users relay the message until the intended recipient receives it.

Rangzen: Anonymously Getting the Word Out in a Blackout

Rangzen [11] is an innovative messaging system that allows for anonymous, one-to-many messages. It uses a social graph to deprioritize adversarial messages and is accessible through a mobile phone-based ad hoc network, making it independent of the internet. Because Rangzen values freedom of speech and anonymity, one node cannot uniquely calculate the trustworthiness of another specific user as the sender information would not be preserved in the system. Instead, it relies on the idea that a message becomes more trustworthy as it passes through more trusted users before reaching its destination. As the message travels through different nodes, users have the option to increase its trustworthiness based on its content, their previous experiences with the sender, and an existing trust score. Over time, the priority of the messages naturally decreases and those with low scores beneath a certain threshold will be removed by the nodes, eventually being eliminated from the system altogether. The proposed approach entails broadcasting of popular, well-liked, and trusted messages among users, while potentially disregarding the unpopular and adversarial ones.

Bridgefy

Bridgefy [12] is a mesh messaging network developed to facilitate offline communication in densely populated areas. However, despite its initial promotion for use in security-sensitive scenarios like large-scale protests, a security analysis [37] has uncovered multiple vulnerabilities. Among the identified issues, adversaries could construct social graphs of users since the network exposes sender and receiver IDs of messages in plaintext. Furthermore, Bridgefy lacks measures to prevent user tracking, as it employs the same encoded identifier in low-energy Bluetooth (BLE) advertising messages. The system also lacks a mechanism for ensuring message authenticity, leaving it susceptible to impersonation attacks, including potential Man-in-the-Middle (MitM) exploits. Lastly, Bridgefy lacks proper defenses against malicious messages that could be crafted by adversaries, potentially compromising the entire system with a single message.

After the initial analysis, the developers claimed to have addressed the identified issues. However, a recent security assessment [38] of the revised Bridgefy SDK contradicts these claims. The analysis highlights persistent vulnerabilities, revealing that the system remains susceptible to impersonation and Attacker-in-the-Middle attacks. Furthermore, it is still prone to Denial of Service attacks using decompression bombs. The examination also unveiled that confidentiality breaches persist, allowing adversaries to recover broadcast messages without knowledge of the network-wide shared encryption key. While some improvements have been made, such as fixing issues with user identities being exposed in plaintext messages and integrating Signal's protocol through the libsignal library, security gaps persist, particularly in the integration process. It's worth noting that the Signal protocol itself, as analyzed extensively by Cohn-Gordon, Cremers, Dowling, Garratt, and Stebila [39], has been found to be robust, with no significant flaws in its design. However, the issue lies in how it was integrated into the system. Addressing these vulnerabilities is imperative for ensuring the integrity and security of Bridgefy, particularly in critical contexts like large-scale protests. The authors of the second analysis highlight their concern that, despite two consecutive years of security assessments revealing vulnerabilities, the popularity of Bridgefy has not significantly diminished. This observation strongly implies an unmet need for a viable alternative that incorporates robust security features.

2.4.2 Delay-tolerant internet accessibility mechanisms

There is a dearth of literature on internet access for communication during a complete internet blackout. In such scenarios, mobile-based ad-hoc networks are the go-to option for facilitating communication within internet-less zones. However, Dolphin [13] sets itself apart by taking on the unique challenge of enabling individuals within an internet-less zone to access internet-based resources such as email, Twitter (X), or specific data. Comparing these technologies is not straightforward, as they address distinct problems. Nevertheless, among the available literature, Dolphin's contribution stands out as a significant one that specifically addresses the access to internet-based resources from within an area lacking internet connectivity.

Dolphin: A Cellular Voice Based Internet Shutdown Resistance System

Dolphin operates by transmitting encoded data bits through the cellular voice channel, utilizing four main components: the client phone, client machine, server phone, and the server machine. In the typical use case, the two server components should be located outside the internet blackout zone and have access to a functioning internet connection. The server machine needs to be connected to the internet, while the server mobile phone should establish a Bluetooth connection with the server machine. Similarly, the client's mobile phone should be connected to the client's machine via Bluetooth.

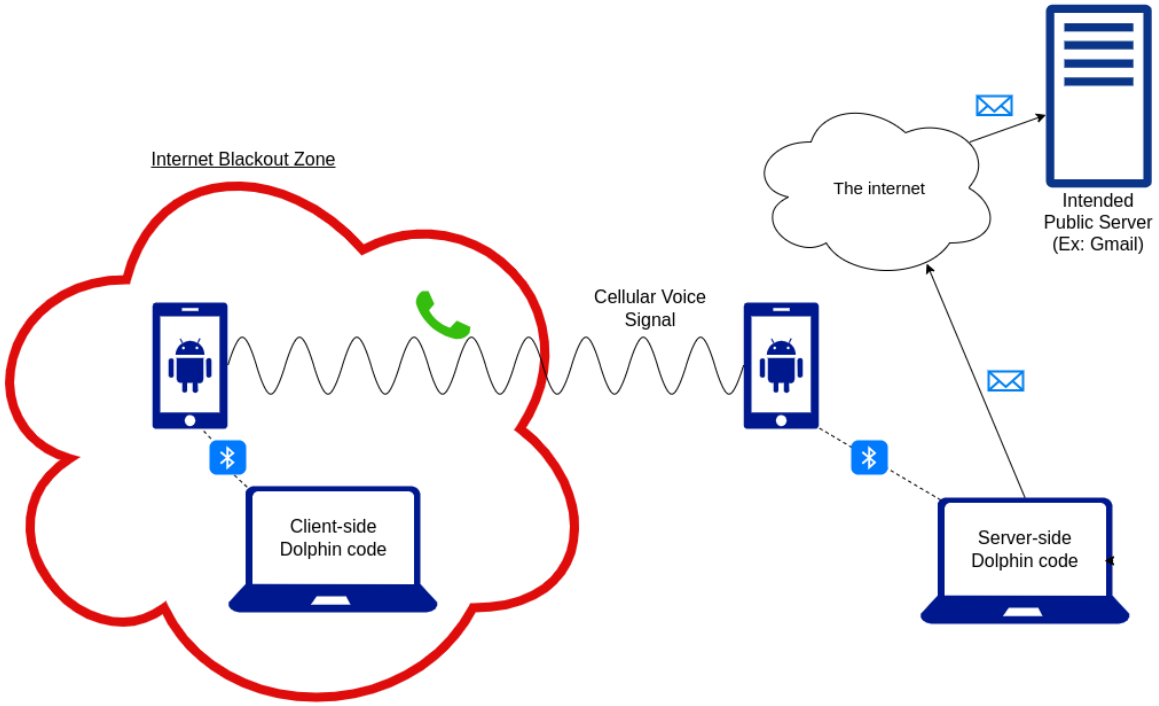


Figure 2.1: Dolphin Architecture

In this process, the client also referred to as the caller, situated in an internet blackout zone, initiates a call to the server phone, also known as the callee, using the Dolphin code for communication. Instead of the typical human voice in the call, the audio is generated through the Dolphin code to encode the data intended for transmission. The server machine receives the input signal from the server phone, decodes the audio, and retrieves the data sent by the client.

The core concept of Dolphin revolves around allowing the client to transmit data, such as an email or a Twitter (X) message while being in an internet blackout zone. The cellular voice signal encodes data, which is secured through encryption. Therefore, even if someone intercepts the signal, no information is leaked. The Diffie-Hellman (DH) key exchange protocol is utilized to establish the shared key used for encryption. This ensures that even if someone intercepts the signal, no information will be leaked. Upon decoding and decrypting

the received signal, extracting the actual data in the process, the Dolphin server can invoke the intended web service API with the received data. This successfully fulfills the client's requested action on behalf of the client. This is why having internet access for the server is an essential requirement when it comes to utilizing internet-based services.

A fair question arises: Would an internet blackout zone still maintain regular cellular voice connectivity? However, considering that the censor, often a powerful entity like a government, is capable of shutting down the internet, they would already be causing significant collateral damage due to the blackout. If the cellular channels were also to be deactivated, the censor would face practical challenges in managing executive and administrative tasks without effective long-distance communication means. Moreover, completely severing cellular connections could potentially incite civil unrest, as restrictions to two of the most popular modern communication methods would impact nearly every individual in society. Additionally, numerous instances of recent internet blackouts [40] [7] [41] [42] provide evidence that cellular voice connectivity tends to remain unaffected even during internet shutdowns. Hence, it is reasonable to assume that cellular connectivity remains intact when an internet shutdown occurs.

It's important to note that the cellular voice channel poses challenges as it is inherently unreliable, subject to losses, and constrained in bandwidth. As the signal traverses between base stations, intermediate connectivity issues may arise, exacerbating the channel's unreliability. While several studies have explored the feasibility of utilizing the cellular voice channel for data transmission, the Dolphin authors assert that these studies primarily rely on simulations and may not capture all challenges present in practical scenarios. These challenges include issues like Voice Activity Detectors (VOD) in mobile phone networks, regular unrecoverable data losses, background noises, and Automatic Gain Control (AGC) mechanisms in modern telecommunication networks.

For instance, the results presented in the paper "A Data Modem for GSM Voice Channel" [43] are simulation-based. Meanwhile, Perić et al. [44] suggest that, for practical usage, additional processing such as noise reduction should be considered. Furthermore, Chmayssani et al [45]. delve into the destructive compression mechanisms employed by medium-rate voice coders in mobile phone networks. However, their performance evaluation utilizes a medium-rate speech coder, which may not fully replicate real-time and practical conditions. Consequently, claims by Dolphin authors regarding the challenges of the cellular voice channel appear credible. To address this inherent unreliability, Dolphin has implemented its own reliability protocol.

Since VOD filters eliminate frequencies outside the human voice frequency band, the signal generated by the Dolphin code must fall within this range to avoid interference. Furthermore, AGC dynamically adjusts the signal's amplitude, rendering it unsuitable for any form of data encoding. For these reasons, Dolphin employs Frequency Shift Keying (FSK) as the modulation technique for encoding the data bits.

Upon inspecting the Dolphin code, it is noteworthy that the authors have not explicitly implemented a mechanism to securely and automatically exchange sensitive credentials, such as usernames and passwords. The current Dolphin code suggests two potential scenarios: either it assumes that the client places full trust in the server to the extent that credentials are pre-shared and stored on the server, or it leaves the matter of secure credential sharing

for future consideration. However, the authors propose several alternatives. One solution involves the caller enabling SMS-based two-factor authentication (2FA), ensuring that even if the server receives the credentials, they would require the One-Time Password (OTP) code sent to the caller's mobile phone to gain access. This approach allows the client or account owner to mitigate potential damage to a certain extent and ensures that session initiation occurs with the caller's consent. Additionally, the Dolphin authors suggest the use of fine-grained access tokens that could be shared as part of the Dolphin message when needed, limiting potential harmful activities from the callee side. Mailet and multi-party secret sharing are two other approaches recommended by the Dolphin authors, and they will be discussed in a separate section.

2.5 Password Sharing

Password needs to be reconstructed before sending to Gmail

2.5.1 Multi Party Secret Sharing

Secret sharing [46] usually involves two groups of entities: the dealer, the one with the secret, and contributing parties that would receive some part of the secret from the dealer. The concept behind multi-party secret sharing is that when the secret is distributed among the n collaborating parties by the dealer, it would take at least t many parties to reconstruct the secret in a lossless way. Any number less than t in any combination would not be able to gain any information on the secret. Hoogerwerf et al. (2021) [47] propose a secret-sharing approach based on Diffie-Hellman (DH) key exchange, relying on the exchange of data in the exponent of a generator. Thus, the security of the approach can be reduced to the security of the DH key exchange. Secret sharing has been a building block or a tool for multiple cryptographic protocols.

2.5.2 Mailet: Instant Social Networking under Censorship

Mailet [14] is a solution designed to facilitate access to social media via email channels when faced with social media blocking or bans. The fundamental concept of Mailet is to serve as a transport proxy, allowing users to circumvent social media restrictions imposed in their geographical locations. The threat model in Mailet operates under the assumption that some Mailet servers may exhibit behaviors ranging from honesty with curiosity to outright malicious intent. The protocol maintains its effectiveness even in scenarios where all servers are honest yet curious, ensuring they refrain from covertly sharing sensitive details amongst themselves. However, should the servers collude to the extent of internal communication for the sharing of sensitive information and encryption keys, the presence of at least one genuinely honest server among the randomly selected servers becomes crucial.

The primary challenge addressed by Mailet is the secure storage and reconstruction of user credentials when necessary. This reconstruction ensures that even if one or a few servers are corrupt, they cannot reveal the plaintext password. The Mailet design involves N servers, from which the user randomly selects two ($2 \leq N$). The user then divides the sensitive text, in this case, the password, into two parts. These parts, denoted as $\text{part}_1 \oplus \text{part}_2$, constitute the full

1. If Callee is a Mailet server
- Message needs to be exposed
- Loose threat model (at least one honest server)

2. If not, can't do on the spot because password has to be sent to callee on unecrypted mode

password, where $\oplus$ represents the bitwise XOR operation. Mathematically, the full password is expressed as full_password $= \text{part}_1 \oplus \text{part}_2$, and part_1, part_2 values are shared among the selected servers.

When the user wishes to access the internet, one server (referred to as the Initiator), holding part_1 of the password, initiates the client side of the TLS handshake with the social website server. Simultaneously, another server, acting as the Interceptor, serves as a proxy between the Initiator and the social website server. Consequently, from the perspective of the website server, the request appears to originate from the Interceptor.

In the Mailet protocol, the Initiator plays the role of securely initiating communication between the user and the social website server. When a user decides to access the internet, the Initiator encrypts part_1, one half of the user's password, using the symmetric key established during the TLS handshake. This encryption process results in the creation of $E(\text{part}_1)$. The encrypted part_1 is then transmitted to the Interceptor, which acts as a proxy between the Initiator and the social website server. At the Interceptor, an essential operation takes place. The plaintext part_2 (the other half of the password) is XORed with the received ciphertext $E(\text{part}_1)$. This XOR operation produces an output that is equivalent to the encrypted form of the full password: $E(\text{part}_1 \oplus \text{part}_2)$. Symbolically, this can be expressed as $E(\text{full\_password}) = E(\text{part}_1 \oplus \text{part}_2)$. For the credential recovery mechanism, Galois/Counter Mode (GCM) encryption is utilized. It's important to mention that this property is not a generic feature among all encryption mechanisms. While it illustrates a specific process within the Mailet protocol, it's crucial not to generalize this behavior to all encryption methods. The security of this process lies in the fact that the Interceptor, lacking the symmetric key, cannot decrypt $E(\text{part}_1)$. Simultaneously, the Initiator remains unaware of part_2. This ensures that no single entity possesses complete knowledge of the user's entire password. Upon receiving the encrypted output $E(\text{part}_1 \oplus \text{part}_2)$, the social website server can successfully recover the plaintext password using the TLS key. This innovative approach in Mailet not only maintains the security of user credentials but also ensures a seamless and protected interaction between the user and the online platform.

2.6 Trusted Execution Environments

Over the last decade, Trusted Execution Environments (TEE) have gained prominence, serving as a valuable mechanism to fortify existing secure platforms [15][16]. TEE establishes a secure and isolated processing environment where applications can operate securely, independent of the broader system, ensuring the confidentiality and integrity of both data and code within the TEE [17][18]. In essence, the use of Trusted Execution Environments (TEEs) offers a higher level of security guarantees for software execution, particularly with regards to confidentiality and integrity. Notably, this heightened security is attained independently of the inherent trustworthiness of the underlying operating system, as trust is instead vested in the hardware implementation [48]. Widely adopted implementations of TEE include Intel SGX [49] and ARM TrustZone [50].

2.6.1 Attestation

Attestation, in essence, is a process enabling one party to verify the genuineness of another party to a desired extent. When a party can remotely attest to the state of another party it seeks to communicate with, it fosters the establishment of trust. When incorporating TEEs, trust is primarily placed on the hardware vendor of the chip. For example, with Intel SGX, trust is placed on Intel, and for Amazon Enclaves in Elastic Compute Cloud (EC2), it is AWS. Conversely, trust is not extended to any user that has access to the machine or to any programs running outside the TEEs. This distinction is crucial as it ensures that while the TEE and its attested state are trusted, external elements and non-TEE applications are treated as potentially untrustworthy.

Incorporating attestation atop Trusted Execution Environments (TEEs) provides heightened assurances regarding the authenticity and integrity of the communication, thereby enhancing overall security measures. Attestation ties authentication and integrity of the trusted execution environment together by providing cryptographic proof that the TEE and the software running within it are both genuine and have not been compromised. In the realm of Trusted Execution Environments (TEEs), attestation operates through two distinct modes: local attestation and remote attestation [48]. Local attestation entails one TEE verifying the authenticity of another TEE residing within the same CPU, leveraging keys tightly bound to the processor. Trust is placed on the hardware vendor of the TEE, allowing TEEs to prove each other's integrity based on that foundational trust. On the other hand, remote attestation necessitates cryptographic proofs to demonstrate the integrity of both the TEE itself and the software it executes, thereby ensuring that neither has been tampered with. In the survey [48] conducted by Jämes Ménétrey et al., the attestation process involves three distinct parties: the attester, responsible for demonstrating its authenticity and integrity; the verifier, tasked with aiding the "relying party" in verifying the attester's claim; and the relying party, which relies on the verifier's assistance to validate the attester's assertions. In the context of TEEs, the attester would typically be the TEE itself, presenting evidence that it is running genuine and untampered software. This verifier's role is often taken by a trusted external entity or service that can assess the cryptographic proofs provided by the attester. For instance, in an Intel SGX environment, Intel would act as the verifier. The relying party is usually the entity that wants to interact with the TEE, such as a client application or a remote server relying on the integrity and authenticity of the TEE. The relying party trusts that the verifier will accurately and honestly assess the attester's claims without being compromised.

2.6.2 AWS Nitro Enclaves

Addressing some of the challenges associated with traditional TEEs, Amazon Web Services (AWS) introduces a solution in the form of Nitro Enclave [52]. Leveraging Trusted Execution Environments, AWS Nitro Enclaves aims to mitigate complexities inherent in conventional TEE usage, such as limited programming options typically confined to the C language and operations restricted to Linux-based systems. Notably, AWS Nitro Enclaves supports programming in Python and Rust, offering a framework to convert standard Docker images into the Enclave Image File (.EIF) format. This methodology is commonly referred to as "lift and

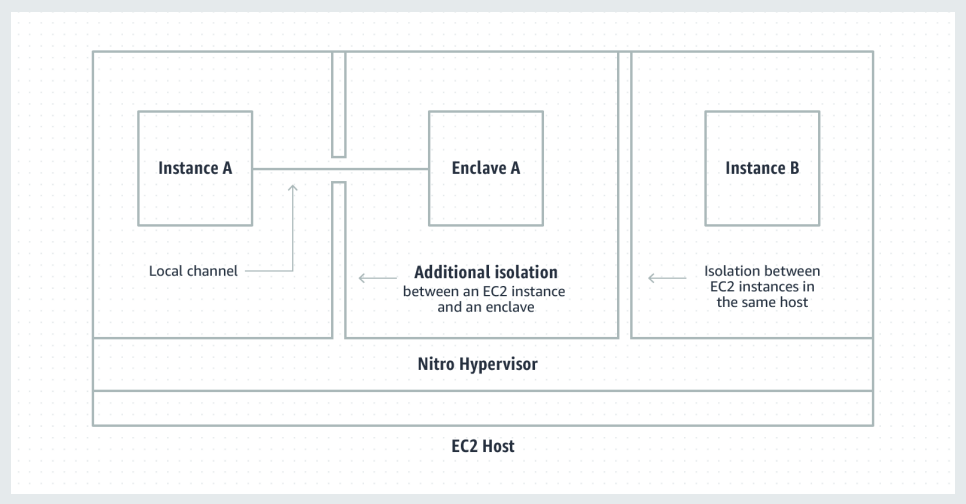


Figure 2.2: AWS Nitro Architecture [51]

shift" [53]. Communication with an application operating in enclave mode is exclusively facilitated through a local socket. When comparing AWS Nitro Enclaves with traditional Trusted Executional Environments, the Nitro Enclave is comparable to the TEE itself in terms of its capabilities and functionality. It's important to note that in order to run this setup, you will need an EC2 [54] machine with Nitro Enclaves. Enclave images are generated using the nitro-cli [55] tool within the EC2 machine, which can be installed on Amazon Linux [56] Operating System. However, it's worth mentioning that nitro-cli can also be installed on an Ubuntu machine, enabling local enclave image generation [57].

In the realm of attestation within AWS Nitro Enclaves, the real-world entity that acts as the root of trust is AWS. The Nitro Hypervisor serves as the foundational component inherently trusted for this purpose, with AWS providing the necessary cryptographic evidence to verify the integrity and authenticity of the enclave [58]. The Nitro Hypervisor is endowed with the capability to generate attestation documents, substantiating the enclave's trustworthiness. These documents encapsulate vital information such as PCR (Platform Configuration Registers) values, the corresponding enclave signing key, hashes of the enclave image and parent instance ID, along with optional dynamic or static user data [58]. Cryptographically signed using the AWS Nitro Attestation Public Key Infrastructure (PKI), these attestation documents are fortified against tampering. The Nitro Attestation PKI incorporates a publicly accessible root certificate, verifiable through a known SHA256 checksum, facilitating the authentication of attestation documents. In addition to that, for each attestation document generated, a unique RSA key pair is created. The attestation document includes the public key, allowing any entity to access it. Consequently, any ciphertext encrypted with this public key can only be decrypted by the private key associated with the attestation document, which is held exclusively by the party that generated the attestation document. As for the PCR values, they encompass a series of hashes, presently totaling six, elucidating diverse dynamic and static attributes of the enclave [59]. Notably, PCR0 reflects the entirety of the enclave image file's content, PCR1 records the measurements of the kernel and boot data and PCR2

denotes a measurement of user applications within the enclave. Additionally, two PCR values are designated for specific measurements of the parent instance, ensuring holistic integrity verification, while the remaining PCR value serves to validate the enclave's booting from a image which is signed by a valid certificate. The protocol to validating the optional user data included in the certificate should be designed accordingly, depending on the specific purpose.

2.6.3 Side Channel Attacks

A side-channel attack [60] [61] [62] [63] can be described as an exploit that leverages ancillary information—primarily physical parameters such as variations in power supply, execution timing, and electromagnetic emissions—arising from the fundamental implementation of a system. Unlike direct attacks targeting a program or algorithm, side-channel attacks deduce information from the collected ancillary data. These attacks have demonstrated their effectiveness in compromising cryptographic operations that are otherwise algorithmically secure [64] [65] [61]. Various forms of side-channel attacks [63] [62] include cache attacks, power analysis attacks including differential power analysis (DPA), and timing attacks.

Countermeasures [62] [61] against side-channel attacks can be classified into two primary categories: reducing the emission of side-channel information and decoupling the relationship between leaked data and the confidential information. Measures to reduce emissions include introducing random noise into the emitted channels, shielding electromagnetic emissions, and implementing power line conditioning and filtering. Blinding, a strategy that falls into the second category, aims to obscure the correlation between emissions and sensitive data by altering inputs into unpredictable states before processing.

Despite the cryptographic robustness of Trusted Execution Environments (TEEs) against various attacks, they remain susceptible to side-channel attacks [66] [67]. These vulnerabilities arise because, while the secure enclave within a TEE may not be directly compromised, interactions with memory, cache, and file systems can be exploited. Attackers can infer details about the computations occurring within the TEE by exploiting vulnerabilities in the CPU microarchitecture. When TEEs access shared hardware resources such as CPU caches or memory buses, they can be monitored by an attacker to glean sensitive information. In essence, the inherent need for TEEs to interact with these shared resources exposes them to various side-channel exploits.

One form of attack is cache timing attacks [67] in which the attacker first evicts TEE-related data from the memory, triggers a memory access by the TEE and then analyzes the cache again to deduce information. Another form of attack, as described in Mubashwir et al. [66] is controlled channel attacks, where the attacker manipulates page faults and page-table entries to infer information through repetitive and strategic manipulation. Some powerful attacks like Foreshadow [68] have successfully extracted secrets from within a TEE [66]. Side-channel attacks on Trusted Execution Environments (TEEs) can target both confidentiality and integrity. For example, cache timing attacks exploit the differences in data access times to extract sensitive information, thereby compromising confidentiality. On the other hand, there exist attacks that target the integrity of TEEs as well. CLKSCREW [69] consists a type of attack in which it manipulates the signature generation process within a TEE to calculate a faulty hash.

However, just as side-channel attacks targeting TEEs exist, there are specialized defenses [66] designed to mitigate these threats. These countermeasures include cache partitioning [70], constant-time algorithms [71], noise injection [72], and both software and hardware modifications to enhance security [73]. It is important to note that implementing these defenses often incurs a higher cost in terms of longer computation times and increased memory usage, which can impact system performance [74].

CHAPTER 3

Goals, Assumptions and Threat Model

3.1 Parties Involved

- **Caller (Dolphin Client):** The entity, facing an internet blackout in their area, initiating communication and utilizing the Dolphin service. The caller has the client side dolphin code locally running in their computer and their mobile phone connected to the computer via Bluetooth. The caller's primary goal is to securely communicate with a public server such as a social media website or an email server, via the Dolphin server, while maintaining confidentiality and integrity.

- **Censor:** The entity responsible for initiating the internet blackout, potentially having authority over the telecommunication infrastructure. The censor's goal may include disrupting Dolphin communication, intercepting transmitted content, and identifying Dolphin clients. Their actions may impact the availability and reliability of the communication channel between the caller and the server. The impact of the censor is already thoroughly covered in the original Dolphin study.

- **Callee (Dolphin Server):** The server-side entity situated outside the internet blackout zone, contacted by the caller, where the Dolphin server-side application is hosted. Additionally, the callee's mobile phone, which received the call from the caller, is connected to the server's computer via Bluetooth. Users on the server side may demonstrate semi-honest or even malicious intents in extracting information. Primary goal of the callee is to gather information shared by the caller. This entity serves as the primary adversary in the current study. The callee is assumed not to collaborate with the censor who initiated the internet shutdown in the caller's area, as it would facilitate the censor in identifying a potential Dolphin client, i.e., the caller. The callee's sole interest lies in extracting communicated information and not exposing the caller to the censor. From a network security standpoint, colluding with the censor would provide the callee with no additional valuable information, as the callee already receives the transmitted signal regardless. Similarly, the callee would not collaborate with their network provider, as doing so would not provide any additional disadvantage. The traffic already goes

through the parent instance of the callee application, over which the callee has full control.

- **Intended Public Server:** The online service targeted by the caller for access, which could range from email servers like Gmail to social media platforms like Twitter (referred to as X). This service has no incentive to extract client information and is regarded as an honest entity in this study. Neither the censor nor the callee would have the capability to collude with this service.
- **TEE Vendor:** The entity responsible for providing the Trusted Execution Environment, such as Intel for Intel SGX or AWS for AWS Nitro Enclaves. It is assumed that the TEE vendor remains trustworthy and does not collude with any other party. The integrity and security of the TEE, along with the corresponding attestation strategy, are trusted to be robust and uncompromised. The vendor's infrastructure and its proper functioning are essential for maintaining the overall security of the system.

3.2 Goals

1. **Confidentiality of the Caller's Credentials (Privacy):** The caller's social media signing-in credentials should remain a secret for all but the client. Simultaneously, no one should be able to secretly use the account in any way that the caller is not expecting. This includes safeguarding third-party access tokens/credentials, like those used by AWS Simple Email Service (SES) for email sending. Allowing access to such tokens could enable actions, such as sending emails via the caller's integrated email account, without requiring the caller's original email credentials. This capability poses a risk to the caller's privacy and is thus unsuitable for inclusion in this study.
2. **No Considerable Performance Hit (Performance):** The email sending time of the new solution should not greatly increase to a level that it makes the new dolphin setup practically unusable.
3. **Confidentiality of the Caller's Data (Security):** Neither a censor nor the user at the callee side should be able to retrieve the data sent by the caller in plaintext.
4. **Verify the Trust (Security):** The best-case scenario is that the caller can identify malicious behaviors, such as any deviations from the expected TEE application code, during the ongoing Dolphin communication before sharing any sensitive details. However, achieving the worst-case scenario — where they can detect such activities upon regaining internet access — also fulfills the goal of ensuring security.

3.3 Assumptions

1. **No Internet Connectivity but Functioning Cellular Services:** This scenario represents the environment where the new solution could be utilized. Existing literature [40]

[7] [41] [42] suggests that such situations, characterized by a forced internet blackout, are practically feasible. Hence, it is reasonable to expect consistent cellular connectivity, albeit without access to the internet due to an ongoing shutdown.

2. **Semi Honest / Malicious User at the Server Side:** The server, also referred to as the callee, situated outside the internet blackout zone, whom the caller, also referred to as the client, will contact, is expected to adhere to the protocol by setting up the system and running the setup. However, there exists a possibility that they may attempt to retrieve credentials or access the message body if given the opportunity. Moreover, the callee can be malicious and may engage in a Man-in-the-Middle attack, impersonating the social media server the client is attempting to access. But in any case, it is assumed that the callee is not interested in launching a DoS attack. Since the callee is getting the Dolphin call from the caller while the caller is expecting the callee to help out, it is assumed that there is some type of personal relationship between the caller and callee, such as a friendship. Therefore, it is assumed that the callee would want to be stealthy and not get caught trying to retrieve the caller's confidential information. Even in scenarios where the caller is unable to detect any malicious activities on the callee's end during Dolphin communication but can detect such activities upon regaining internet access, it is expected that the callee would refrain from exploiting these vulnerabilities. The rationale behind this expectation is that the caller would eventually detect such activities upon regaining internet access, and the callee would prefer to maintain stealthiness and avoid being caught attempting to access confidential information.

3. **Secure Trusted Execution Environments (TEE):** It is assumed that the Trusted Execution Environments are functioning as intended. Consequently, server-side users cannot examine or tamper the secure code running inside the TEE or AWS Enclave when operational. Any such attempts should be identifiable through the validation of the attestation document. Essentially, the assumption model transfers trust to the hardware vendor of the TEE (e.g., Intel for Intel SGX, Amazon for AWS Nitro Enclaves, etc.) with the expectation that they construct the TEE to be tamper-proof. For the purposes of this research, it is presumed that the TEE is resilient to side-channel attacks. The implementation of countermeasures against such attacks may be considered in future work.

4. **Attestable Trusted Execution Environments (TEE):** It is presumed that the legitimacy of the TEE and the code executing within it can be attested and verified remotely. In essence, the attestation mechanism is relied upon to identify any fraudulent activities occurring on the TEE.

5. **Non-Forgeable Public Server Certificate:** It is assumed that users on the callee side are unable to forge the TLS certificate of the public server (e.g., Gmail) in a manner that deceives certificate verification software.

6. **Robustness of the TLS Encryption:** This study operates under the assumption that the TLS channel formed between the public server and the TEE executing code is

resilient against cryptographic attacks. Implicit in this assumption is the trust placed in the public server's selection of a cipher suite and TLS version devoid of known vulnerabilities. Similar to the presumption of the TEE's resistance to side-channel attacks, the TLS channel is also believed to possess adequate resistance.

3.4 Threat Model

The solution contends with two primary threats: firstly, the censor, who aims to disrupt Dolphin communication, intercept transmitted content, and potentially identify clients attempting to utilize the Dolphin service [13]; and secondly, the users on the server side, endeavoring to access credential information. The original Dolphin paper addresses the first threat comprehensively, discussing potential DoS attacks and the challenges faced by censors in disrupting Dolphin communication without significantly impacting regular cellular usage. Furthermore, the paper explores the ineffectiveness of replay attacks and emphasizes the encryption of communicated content, rendering it incomprehensible to eavesdroppers. Additionally, the authors underscore the practical difficulty of offline attacks on a large scale, attributing this challenge to the similarity between the signal waveform of superimposed Dolphin and normal human speech audio signal. However, this study primarily focuses on the second threat posed by the users on the server side, who may exhibit curiosity or malicious intent in extracting information. They may resort to inspecting the computer's runtime and operations to glean sensitive data. Additionally, there is a possibility that they may deploy a similarly appearing but inherently malicious application to deceive the client and gather information surreptitiously. That attack could come in the form of spoofing attack or a Man-in-the-Middle (MitM) attack. The censor and the callee have divergent intentions, leading to the assumption of non-collaboration between them. Additionally, neither the censor nor the callee would have the ability to collude with the public server that the caller aims to communicate with.

CHAPTER 4

Comprehensive System Analysis: Design, Implementation, and Security

4.1 System Design

4.1.1 Design Architecture

Let's delve into the design of the new Dolphin extension, focusing on its architecture on the server side. We'll start by delineating the various functional interfaces on the server side, emphasizing that these interfaces may not always correspond to physically distinct entities. The architecture operates under the assumption of a conventional Trusted Execution Environment (TEE), wherein the server machine incorporates an internal TEE. To clarify, the term "parent instance" refers to the entirety of the callee's computer system excluding the TEE utilized in our implementation.

The server-side process commences upon the receipt of a Dolphin call from the client to the callee's phone. As the call is answered, the audio signal is transmitted to the server machine via Bluetooth. Subsequently, the server-side code decodes the audio into a byte stream using the minimodem software. Notably, all communication between the client and server is encrypted following the Diffie-Hellman (DH) key exchange protocol. Similar to the initial Dolphin implementation, AES-128 in GCM mode is used for encryption/decryption. While the original Dolphin code assumes full trust in the server, the encryption key generation now occurs within the Trusted Execution Environment (TEE), marking the first departure of the new extension from the original Dolphin setup. Here, only the callee's public key generated within the TEE is transmitted back to the parent instance for relay to the client, while the client's public key is received by the TEE through the parent instance. Consequently, the TEE can generate the symmetric key (illustrated in red in Figure 4.1) for subsequent data encryption/decryption operations. The use of red color arrows in the figure signifies that the communication is encrypted using this same key. Thus, any data sent by the client, such as message bodies or user credentials, is encrypted by the client before transmission. This encryption ensures that the data remains secure and inaccessible to entities outside the TEE until it is decrypted within the TEE. This includes the server-side code running within the parent instance, which do not have access to the symmetric key for the encrypted content. Essentially, the parent instance

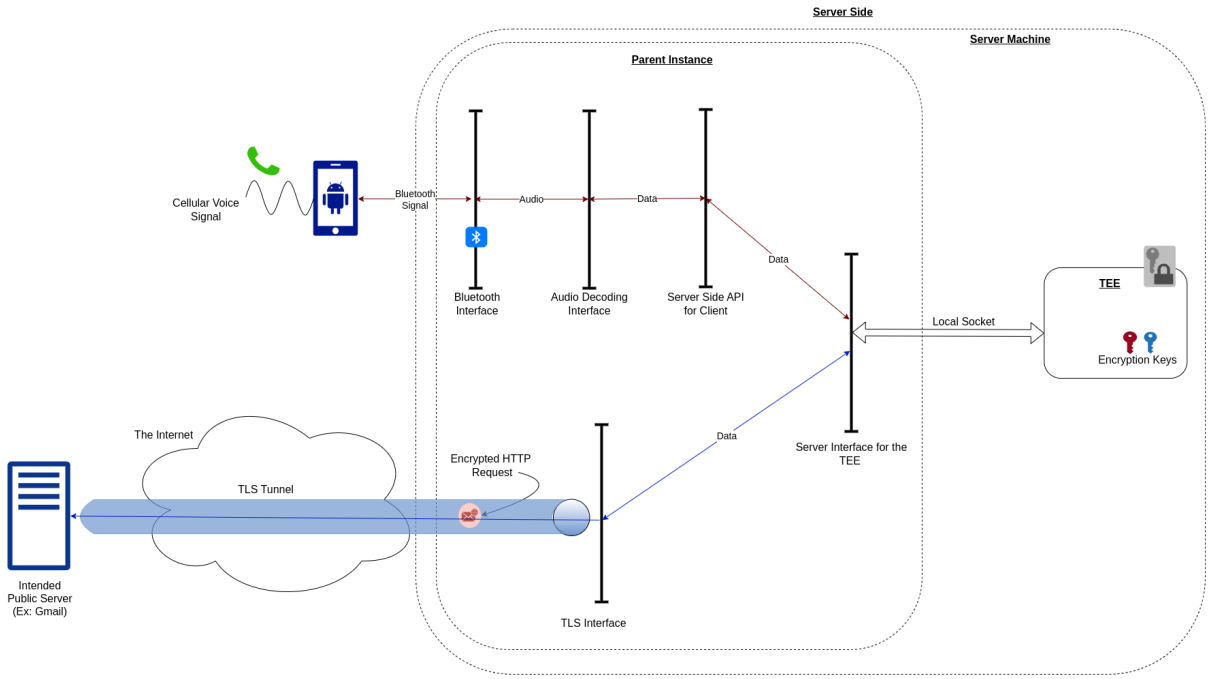


Figure 4.1: Server Side Functional Architecture

serves as a proxy, facilitating the exchange of messages between the client and the Trusted Execution Environment (TEE). Upon receiving data from the client, the application within the TEE decrypts it, stores it in local memory, and sends an acknowledgment back to the parent instance, which is then relayed to the client. This guarantees that all information intended by the client remains confidential until it is received by the TEE running code, where it is securely stored within the TEE application. A more comprehensive depiction of the communication is illustrated in Figure 4.2. Please be advised that the notation E(D, K) denotes the encryption of data D using the key K.

Following the secure retrieval of all necessary information to initiate the request to the designated public server, the subsequent challenge lies in the execution of the request. Returning to the discussion on Trusted Execution Environments (TEEs), these isolated and secure areas of memory and CPU serve as the foundation for our implementation. Notably, the TEE utilized in our implementation possesses solely one socket for communication with the parent instance, precluding direct socket connection to the Network Interface Card (NIC) and, consequently, impeding direct access to the internet from within the TEE. To devise a solution that is resilient across diverse TEE implementations, we presume that the TEE lacks direct means to dispatch an HTTP request to a public server. In essence, the TEE must navigate a pathway to transmit the HTTP request through the parent instance while safeguarding the integrity of the client's securely transmitted information. Unfortunately, most client-side libraries that are capable of making HTTP requests are developed under the assumption that the client machine is trusted. This creates a challenge in finding an off-the-shelf library that can be used in the current scenario, where a request must be made from a machine without exposing information and credentials in plaintext. To the author's knowledge, no existing libraries cater to the specific

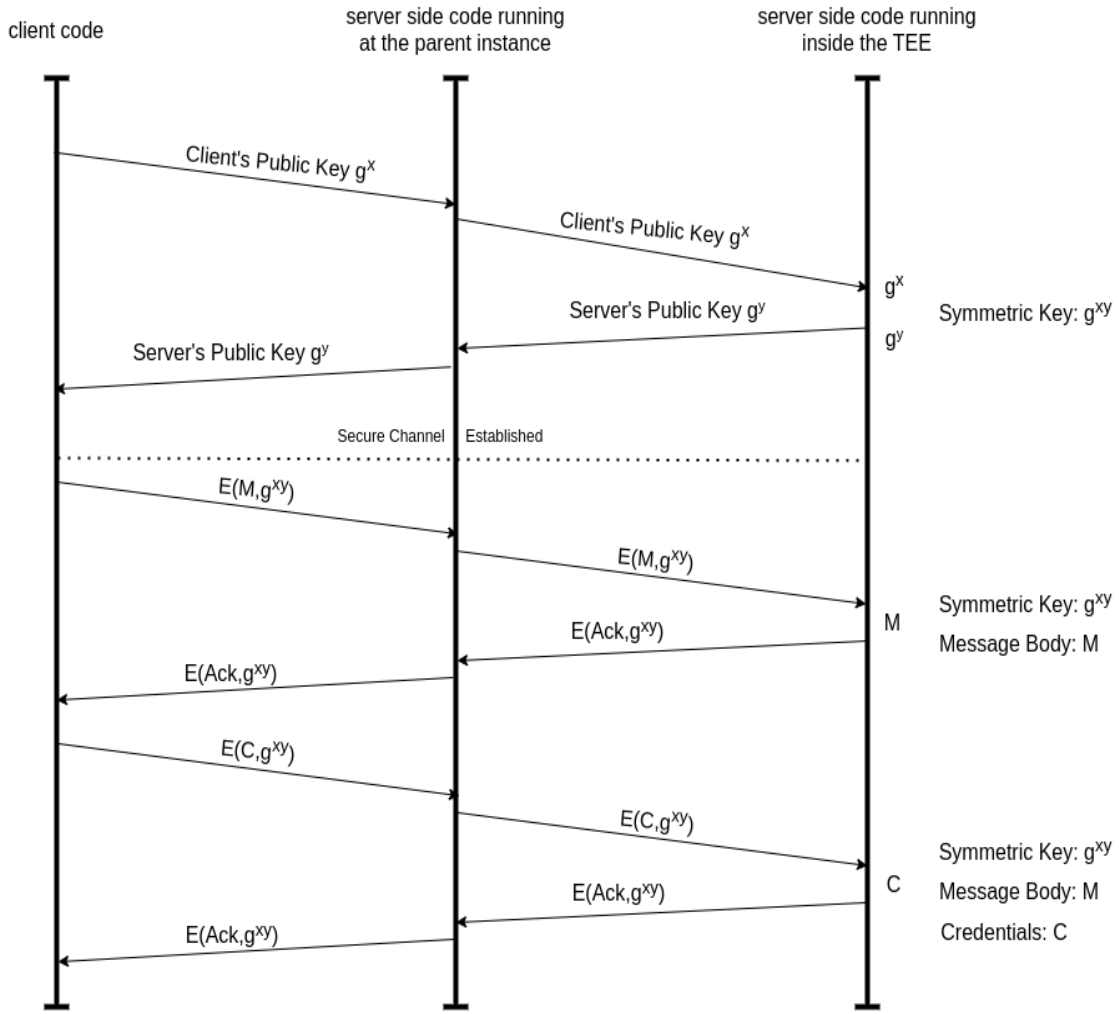


Figure 4.2: Communication Between Server-side Entities

scenario where requests must be made from a machine without exposing credentials and information in plaintext.

One potential solution for meeting the email-sending requirement could involve utilizing an email transmission service such as Amazon SES (Simple Email Service), which does not necessitate the inclusion of email usernames and passwords for sending emails. In certain email clients, such as Amazon SES, email integration is carried out independently. To send an email, credentials/tokens from the client service are necessary instead of the actual email account. While this approach doesn't reveal email credentials, it does entail exposing the email client's token and body to the parent machine. Enabling the parent instance to utilize those tokens to send emails without the caller's authorization would breach the initial security goal outlined in the previous chapter. Moreover, revealing the email content in plain text to the parent would compromise the security objective of safeguarding the client's data privacy. Hence, this approach did not align with the anticipated security objectives.

Another potential approach involves utilizing the parent instance as an HTTPS proxy. In this scenario, if the request is made via HTTPS, the proxy would not have access to the content in plaintext. However, for this setup to function effectively, the TEE would need to be the entity initiating the HTTPS request. While this arrangement may have been feasible for certain TEE configurations, it was not achievable in the implementation phase of this study due to limitations in the communication protocol between the TEE and the parent instance via the local socket. Consequently, this option was not pursued as a viable solution.

The approach employed in this study involves establishing a TLS connection from the parent instance to the public server. However, a key distinction lies in the handling of encryption keys within this process. Rather than the parent instance generating the client-side encryption key for the TLS protocol and subsequently calculating the final symmetric key, these tasks are delegated to the TEE. From the perspective of the public server, the parent instance is identified as the client, yet it lacks control over the encryption keys. Following the TLS handshake and the generation of the symmetric key within the TEE, the TEE encrypts the intended request, which is then transmitted to the parent instance. Subsequently, the parent instance passes the encrypted request via the TLS tunnel, thereby completing the process of sending the request. One might question why HTTPS is not utilized directly, as it essentially encapsulates HTTP within TLS encryption. However, as mentioned previously, existing client-side tools for sending HTTPS requests are typically developed with the assumption that the initiating machine is inherently trusted, at least to the best knowledge of the author. Consequently, it is necessary to establish a TLS connection first and then utilize that channel for the actual request transmission, ensuring a more secure communication process.

Figure 4.3 illustrates the orchestration of the TLS handshake by the TEE-based application between the parent instance and the target public server. Messages exchanged between the TEE-based application and the parent instance are treated as stringified messages. Upon receiving a response from the server, the parent instance string-encodes the message and forwards it to the TEE. Conversely, it decodes stringified messages from the TEE into bytes, constructs corresponding handshake messages with the received data, and transmits them to the public server. Notably, the parent instance functions as a passive intermediary, merely relaying messages without decrypting the content, for which it lacks the knowledge of the session key. Please be aware that in Figure 4.1, the blue encryption key signifies the TLS session key that is computed within the TEE-based application. Once established, the TEE instance and the intended public server can exchange information securely without disclosing the content to any third-party, including the parent instance. This encrypted communication is represented by the blue arrows in the same figure, connecting the TEE and the public server via the parent instance.

In network protocols, maintaining backward compatibility is often crucial for seamless communication. However, this necessity introduces a security vulnerability commonly known as downgrade attacks, particularly evident in the realm of SSL/TLS versions, where older protocols may lack the robust security features of their successors. A notable point of vulnerability lies in the unencrypted nature of the server hello message during TLS handshakes, leaving it susceptible to tampering by intermediary components like the parent instance. Various strategies exist for such components to exploit this vulnerability. One approach

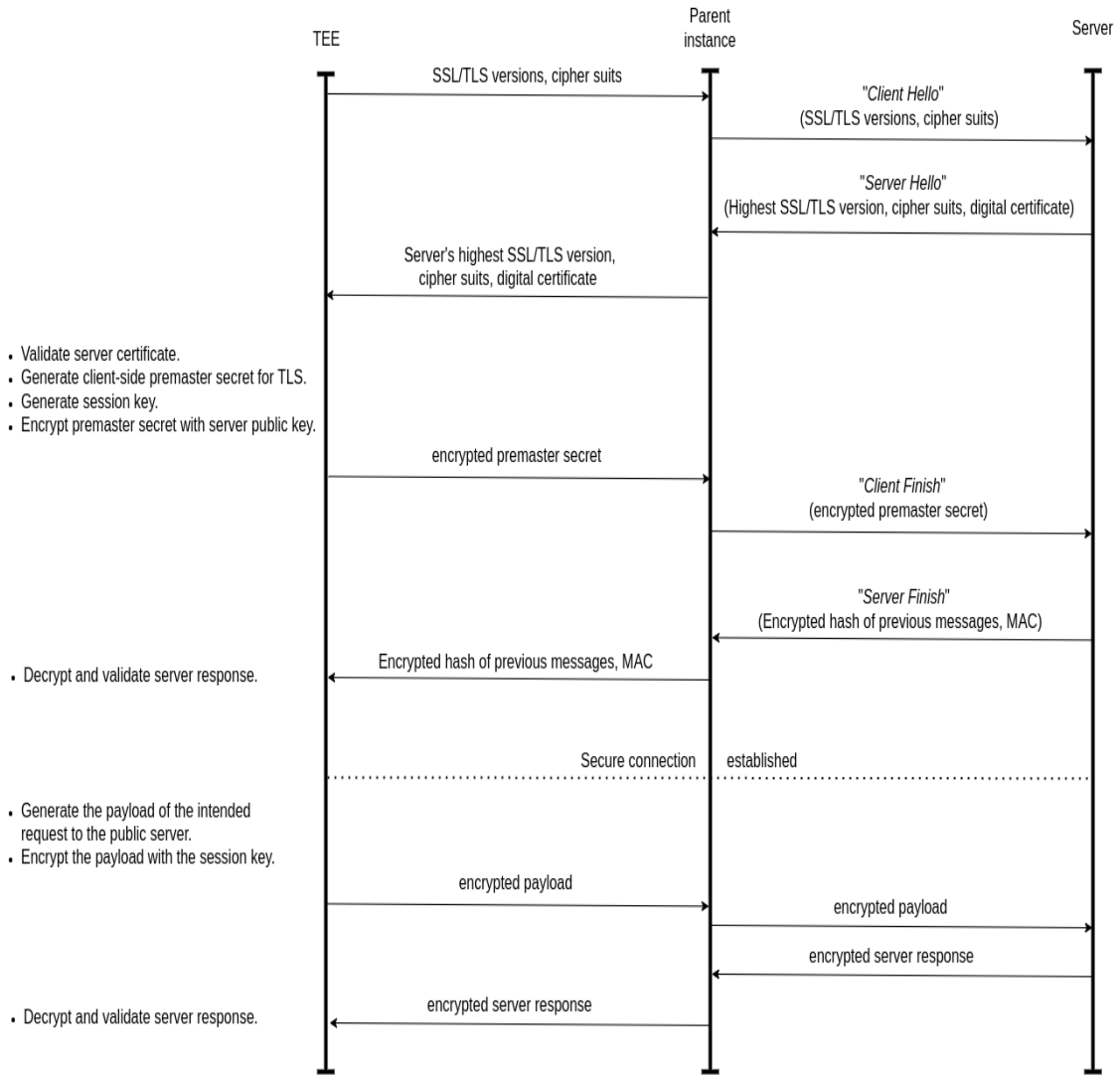


Figure 4.3: TLS-based Communication Between Server-side Entities

involves substituting the cipher suite with a new selection (not originally offered by the server) known to harbor vulnerabilities or susceptibilities exploitable by the parent instance. However, should the TEE application select one of these compromised suites, the public server would reject it, as it does not support suites not originally offered. Another tactic involves the elimination of stronger ciphers, leaving only those deemed less secure. Alternatively, the parent application may opt to retain only the lowest TLS version offered by the public server. However, it's essential to recognize that these strategies represent downgrade attacks, potentially compromising the security of the TLS handshake. Notably, the default defense against latter two approaches rely heavily on key assumptions. To handle situations where the parent instance removes strong TLS versions and/or ciphers, the TEE application must rely on the premise that the lowest TLS version provided by the public server and/or the least secure cipher suite sent still maintain sufficient resilience against practical decryption, upholding

satisfactory standards of confidentiality. Rather than solely relying on such assumptions, a more robust solution involves integrating specific constraints into the TEE application. This logic would dictate proceeding with the TLS handshake only if the cipher suite's algorithms are devoid of known vulnerabilities and if the TLS version surpasses a predefined threshold of security. By adopting this approach, the TEE application can effectively mitigate downgrade attacks, thereby ensuring the integrity and confidentiality of the communication channel.

4.1.2 High-Level Solution Overview

Bringing together all the components, the high-level solution that extends the Dolphin system to enhance caller privacy consists of five stages:

1. **DH Key Exchange between Caller and Callee:**

   In this stage, illustrated in Figure 4.2, the callee's side key generation occurs securely within the TEE, with the parent instance serving solely as a proxy between the client and the TEE. Following completion of this stage, both the caller and the application within the TEE can construct the final symmetric key.

2. **Attestation Document Verification:**

   Here, the caller requests an attestation document from the callee. The parent instance relays this instruction to the TEE, where the attestation document is generated. Containing runtime information of the TEE, including the code running inside it, a piece of data that connects the attestation document with the ongoing communication session (such as the constructed symmetric key itself encrypted with the symmetric key), and data ensuring the integrity of the TEE, this document is then passed back to the caller via the parent instance. Attestation documents of TEEs are designed to uphold integrity standards by being signed, , typically by the TEE's root of trust or an equivalent trusted authority, thus preventing unauthorized alterations. However, for broader applicability, it is recommended to further secure the attestation document by encrypting it with the established symmetric key. This additional measure ensures that even if the attestation document is tampered with, it would be difficult to alter the encrypted data in a meaningful way—meaning that any attempts to manipulate the attestation process and compromise the integrity of the attestation document would likely be unsuccessful.

   Upon receipt, the caller validates the document to ensure it's communicating with an untampered TEE, that the TEE's code aligns with expectations, and that the attestation document originates from the expected TEE. The expectation is that even if a single character differs from the expected code, the generated hash representing the Trusted Execution Environment (TEE) running code in the attestation document would be distinct. This approach allows the caller to ensure with absolute certainty that the attestation document was produced by a TEE running the server-side code that the caller is familiar with. Successful validation establishes trust for subsequent content transmission. If the concrete attestation process allows for the inclusion of dynamic user data within the attestation document, which the relying party (in this case, the caller)

could extract upon validation, it is advised to include some pre-agreed data encrypted with the shared key as user data. This enables the caller to validate that the attestation document was indeed produced by the TEE that the caller is actually communicating with via the Dolphin setup. It should be noted that coming up with a strategic protocol to verify the attestation document and shared key generation as one atomic unit should be decided accordingly, depending on the concrete TEE implementation used in the solution. This step is crucial since it confirms that the TEE instance that generated the attestation document is indeed the TEE instance that generated the callee-side private key during the DH key exchange. Without it, the callee could launch an attack where two applications run in parallel on the callee's end. More details of this attack are covered in section 4.3, "Possible Attacks and Defenses."

Another concerning point is that, depending on the specific TEE used, the client may not be able to validate the attestation report comprehensively and might need to defer complete validation until internet access is restored. This comprehensive validation could typically involve verifying the authenticity and integrity of the attestation report using external services or certificates, which may require internet access. Consequently, the actual verification of the attestation document could occur after the Dolphin-based communication, even at a later point such as when the internet is restored for the caller. Under the assumption in section 3.3, that the callee aims to conceal their malicious attempts, it is irrelevant whether the client validates the attestation document before or after sharing credentials and information. Nonetheless, without this assumption, pre-attestation is crucial as it prevents the caller from initially sharing sensitive data with an untrusted entity, while post-attestation only identifies fraudulent attempts after they have occurred.

3. **Caller Sends Message Body and Credentials:**

   Encrypted using the symmetric key, the caller transmits the message body and social media account credentials to the callee. Only the TEE possesses the decryption capability, securely storing this information in local memory without exposure to the parent instance.

4. **TLS Handshake with Intended Public Server:**

   With all necessary information safely relayed to the TEE, the next step involves establishing a TLS connection between the parent instance and the intended public server, as depicted in Figure 4.3. The premaster key for the TLS session is exclusively generated within the TEE to prevent exposure in plaintext.

   When the TEE receives the server's digital certificate in the server hello message, there are two possible approaches. The first option is to relay the certificate back to the caller, enabling independent validation. Moreover, alongside the standard validation methods, the caller could preemptively download the digital certificate of the intended server before the internet blackout and cross-reference it with the newly received one. Alternatively, digital certificate validation could be integrated into the logic of the TEE application, incorporating digital certificates of commonly used public servers into the

TEE running code. Since the caller has already attested to the code running inside the TEE in the previous step, this method should be equally secure as validating the digital certificate at the caller side.

Depending on the assumption mentioned in 3.4.5, whereby users at the callee end cannot forge the digital certificate of the public server in a manner that would pass validation, it can be safely assumed that the callee cannot successfully launch a MitM attack or a spoofing attack pretending to be the public server. Finally, once the TLS handshake is completed, the TEE would possess the session key for the TLS session, while the parent instance would not.

5. **Final Request Transmission to Public Server:**

   Following TLS handshake completion, the TEE constructs the payload for the intended HTTP request, encrypts it with the session key, and passes it to the parent instance for relay to the public server. Conversely, the encrypted server response is routed back to the TEE for decryption and re-encryption with the established symmetric key before delivering it to the caller. This allows the caller to examine the server response after decrypting it with the established symmetric key, and determine the success or failure status of the intended request to the public server.

4.2 System Implementation

4.2.1 Solution Architecture

In the actual implementation for the proof of concept, several decisions were made.

Firstly, AWS Nitro Enclaves was chosen as the TEE due to its Python-based programming framework, facilitating seamless integration with the existing Python-based Dolphin code. This involved utilizing an m5.4xlarge EC2 instance with Nitro Enclaves enabled. However, one limitation of using cloud machines was their lack of support for certain Python libraries like minimodem, primarily due to limited support for sound drivers. Since the system required functionality such as decoding audio signals to byte streams and vice versa, a workaround was devised. In the implementation, the callee machine described in the system design section was divided into two parts: one part running on the local machine of the callee and the other on the EC2 instance. The program running on the EC2 instance consists of two sub components itself: the parent instance and the enclave instance. The parent instance maintains a websocket connection with the local machine, while the enclave instance executes the code within the TEE (Nitro Enclave). To enable bidirectional communication, the local machine and the EC2 instance were connected via websockets. The parent instance with a public IP address acts as the websocket server while the local instance executes as the client. Initially, the callee's local mobile phone would receive the call and transmit the signal via Bluetooth to the local machine. There, the audio signal would be decoded into a byte stream, which would then be sent to the EC2 machine as a websocket message. Upon receiving the message, the EC2 server would extract the data and communicate it to the application running in the Nitro Enclave. Similarly, when data needed to be passed to the caller, it would travel from the Enclave to the

parent instance, then from the parent instance to the local machine via websockets, eventually reaching the caller, mirroring the normal operation of the Dolphin setup. In terms of security, it does not matter whether the TLS connection with the intended public service (ex: Email) is constructed from the parent instance or the local computer of the callee. However, for latency concerns, TLS connection handling was assigned to the parent instance since each TLS handshake message requires communication with the Enclave anyway.

Secondly, for the proof of concept, the use case of sending emails was implemented, deemed significant for scenarios involving internet blackouts. However, the same concepts could be applied to other tasks such as posting on social media platforms like Twitter or accessing resources from the internet.

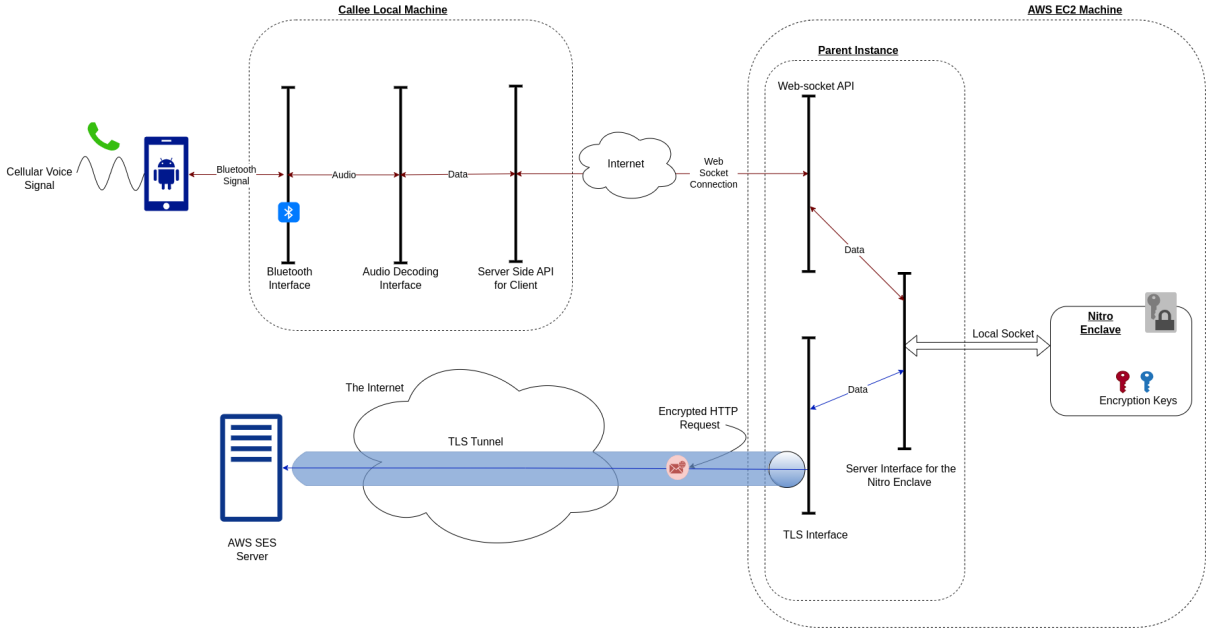


Figure 4.4: Server Side Implementation

Thirdly, AWS SES [75] was selected as the email sending client for various reasons. While considering options for sending emails, it was noted that native client-side libraries, such as the Python library called googleapiclient [76] for sending Gmails, couldn't be used due to the requirement of making requests from the untrusted parent instance. These libraries handle TLS session setup and offer only high-level APIs, rendering them unsuitable for this use case. Consequently, only HTTP/REST-based options were explored. Following Google's deprecation of APIs allowing sign-in using a Linux terminal or third-party Python libraries like smtplib with just email address and password credentials since May 30th, 2022 [77], the industry standard shifted to 2-step verification with OAuth2.0. Attempts to sign in using a general Gmail account via HTTP/REST APIs proved infeasible, as OAuth2.0-based sign-in led to a HTML page sign-in process [78]. The remaining options were either using a Google service account for sign-in [79] and email sending or leveraging an email client like AWS SES, offering a purely REST/HTTP-based approach to accomplish the task. Google service accounts are specialized accounts for application interaction with Google services programmatically,

eliminating the need for browser-based sign-in. Although the Google service account path could have been viable, the implementation was streamlined with AWS SES due to its ease of integration. Nonetheless, the Google service account-based approach could have been equally successful.

4.2.2 Attestation

Since the TEE selected for the implementation is AWS Nitro Enclaves, the attestation process is based on the same framework. As previously described, the attestation document contains PCR values and includes a private/public key pair based on AES encryption. In this implementation, the attestation document is generated inside the Nitro Enclave and sent to the client. Since the Nitro Enclave framework supports local validation, the client can verify the attestation locally using the necessary libraries. Attestation document verification process is two fold: verifying by using the root certificate for the AWS Nitro attestation for the legitimacy of the attestation document and then analyzing the PCR values for further checks. The most crucial PCR values for verification are PCR0 and PCR2. While PCR0 is used to verify the legitimacy of the attestation document, PCR2 can be used by the client to ensure that the callee is running exactly the code the caller expects within the enclave. This is achieved by checking the PCR2 value, which is essentially a measurement of the applications code running in the enclave. Essentially, if the build environment including the library versions are the same and the code that the enclave image is built on is similar, the PCR2 values calculated in two different virtual machines should be similar. The question arises regarding what the PCR2 value included in the attestation document should be compared against. It should be noted that the PCR2 value is generated during the enclave image creation for a specific code using the Nitro CLI [55]. The standard method involves generating the enclave image on an EC2 machine running on Amazon Linux OS [56], where the Nitro CLI tool is available for installation.

One method for the caller to calculate the PCR2 value is by creating the enclave image themselves (and deriving the PCR2 value as a byproduct) before the internet blackout, and then sharing the .eif enclave image with the callee. The enclave image can be easily generated in an EC2 machine or done locally on a Linux device [57]. If the solution is intended for larger use, the state of the EC2 instance could be mounted to a machine image [80] and made public along with the code repository including the PCR2 value of the enclave image. This process could also be undertaken by the author of this paper, although it would necessitate placing trust in the author to supply the accurate PCR2 value for the published Amazon Machine Image (AMI). Alternatively, one could spin up an EC2 machine using the published AMI to independently calculate the PCR2 value. Since we anticipate the caller to prepare in advance for an internet blackout by possessing the client-side code, either of these steps could be considered as part of the preparation. If the caller fails to pre-calculate the PCR2 value prior to the internet shutdown, they can perform only the PCR0-based and root certificate based legitimacy checks of the attestation document during the Dolphin-based communication and proceed with credential sharing if the validation succeeds. However, it is emphasized that without a pre-agreed PCR2 value, the caller cannot verify that the expected code is running in

the callee's TEE. Not having this pre-agreed PCR2 value could have significant consequences, as the caller would be unable to detect modifications in the TEE code that could result in malicious behavior, such as the callee altering the code to output plaintext credentials. Thus, pre-agreeing on the PCR2 value is crucial to ensure the integrity and expected operation of the TEE application, thereby preventing potential security breaches. The entire design hinges on verifying that the untrusted party adheres precisely to the protocol. If there's no way to guarantee that the enclave executing the code functions exactly as intended, then confidentiality is at risk, and sensitive information could be compromised.

It is essential to note that the attestation process, as verified by the PCR values and root certificate, does not indicate that the verification process is fully completed. Rather, it signifies that the callee is operating a server that executes the code expected by the client, and that the attestation document was produced using that same instance. However, it does not assure that the instance involved in the subsequent Diffie-Hellman (DH) key exchange, is the same instance that generated the received attestation document. The callee could theoretically use a legitimate Nitro Enclave solely for generating the attestation document without involving it in prior communication steps. To identify such attempts, it is crucial to verify whether the callee side private key generated during the DH key exchange (not the attestation document's private key) matches the private key generated by the instance that created the attestation document, in a previous step. If a match is found, it can be logically concluded that both keys belong to the same system. Since the resulting shared key is a result of the caller-side public key and the callee-side private key, by logical inference, the shared key should also be as unique as the callee-side private key. This point would be the deciding factor to verify the fact that the attestation document was generated by the same enclave application that participated in the communication process up to the attestation point.

To ensure this, the caller can generate a random text, encrypt it with the public key of the attestation document, then encrypt the resulting ciphertext using the shared key from the DH key exchange, and send it to the callee. Only the enclave that generated the attestation document possesses the private key corresponding to the attestation document's public key. Since the caller has validated that this instance runs the expected code, it means the callee cannot retrieve either the private key for the attestation document or the shared key generated by that application. Both keys remain confidential, as the code does not expose them. Thus, the double-encrypted ciphertext can only be decrypted by an application holding both keys: the shared key from the DH key exchange and the attestation document's private key.

Subsequently, the caller awaits the double decrypted ciphertext from the callee and ensures that it aligns with the original random text. It confirms that the caller has been communicating with a nitro enclave running the expected application code, which also generated the attestation document received by the caller. This process ensures that the shared key used for communication remains confidential, making it secure to proceed with credential sharing.

4.3 Possible Attacks and Defenses

All the attacks specified in the original Dolphin setup, such as DoS attacks, replay attacks, and attempts by censors to identify Dolphin clients, remain applicable to the new extension.

The exact defenses outlined in the original Dolphin paper are equally applicable, as the new extension does not alter the underlying communication protocol of the original Dolphin. The Dolphin-specific reliability layer for communication over the cellular channel, audio encoding and encryption methods used on top of the audio-encoded data remain unchanged. This section focuses on additional threats arising from the new threat model, which considers the callee as a colluding entity, that could display a range of behaviors from semi-honest to malicious intent.

1.
   - **Attack Scenario:** This scenario represents one of the most basic attacks that the callee could try. Here, the callee connects their mobile phone to a local machine running Dolphin-like code without TEE integration. This fraudulent setup attempts to extract the credentials and message body sent by the caller. From the caller's perspective, there would be no discernible difference, as they would still receive the caller's public key for symmetric key calculation. Given that the deceptive application on the callee's side lacks TEE integration, the user at the callee end can effortlessly decrypt the transmitted content using the readily available final encryption key.
   - **Defense Strategy:** This attack would fail at the attestation document verification stage, as mentioned in section 4.2. Without a TEE, the necessary documentation cannot be generated, thwarting the attacker's efforts.

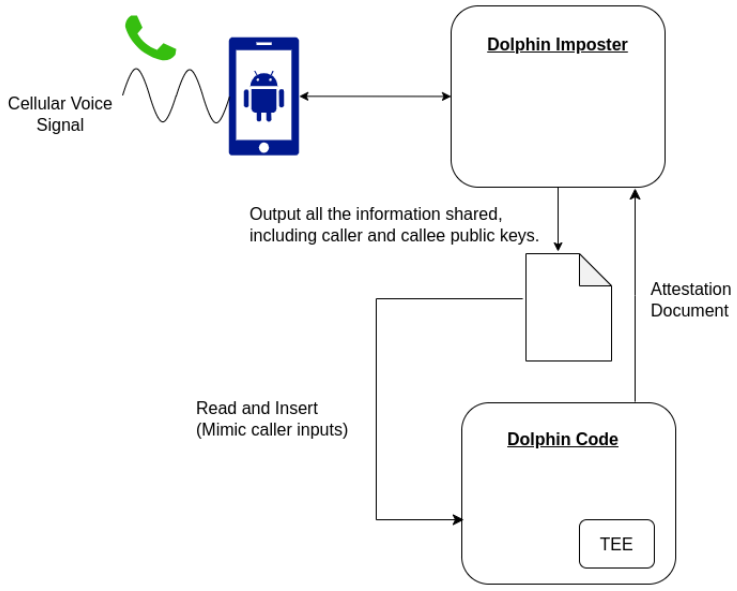


Figure 4.5: Attack with Parallel Applications

2.
   - **Attack Scenario:** In this scenario, the callee utilizes both the genuine Dolphin code with the new extensions, running within a genuine TEE, and a dolphin-like code without TEE integration. The latter communicates with the actual dolphin client, pretending to be the application that is expected to run at callee side. The

callee manually feeds data into the genuine Dolphin code to mimic communication from the caller, ensuring minimal latency. Additionally, the callee provides the same public key as the caller to the genuine Dolphin code. When the caller requests the attestation document, following the procedure outlined in the High-Level Solution Overview, the callee obtains the attestation document from the genuine code and forwards it to the caller. Refer to Figure 6.1 for further details.

- **Defense Strategy:** The attestation document should include some pre-agreed data encrypted with the symmetric key used for encryption/decryption between the caller and callee. Despite the callee employing the caller's public key, the genuine Dolphin code and fraudulent dolphin-like code are unlikely to generate identical server-side encryption keys. In the context of key exchange using Diffie-Hellman (DH), it is improbable that two applications calculate the same server-side private key, due to the inherent properties of the DH algorithm. This is because each party generates their own private key randomly, typically using large prime numbers. In the implementation, the private keys are randomly generated from a key space of 2^{256} possible keys [81]. As a result, the probability of two parties independently generating the same private key is extremely low. Therefore, when verifying attestation documents, the caller can reasonably expect that any attempts to mimic communication using fraudulent dolphin-like code would be detected. This is because the server-side encryption keys generated by the genuine Dolphin code and the fraudulent code are expected to differ.

3.
- **Attack Scenario:** The callee, while running a Dolphin instance with an integrated TEE, makes slight modifications to the code to log caller credentials and request body to the Enclave console output in plain text.
- **Defense Strategy:** In the attestation document, a hash of the code running in the Enclave is included. With the modified code, the hash output would differ from the expected hash when the caller validates the attestation document. Consequently, the caller can terminate the communication before sharing credentials. This defense relies on the assumption, as outlined in section 3.2, that the attestation document contains a property to validate the integrity of the TEE running code, and the hash function is designed such that any change in the code results in a different hash.

4.
- **Attack Scenario:** The user at the callee's end manipulates the TEE to extract information about its runtime, seeking to access caller credentials.
- **Defense Strategy:** As per assumptions 3.2.3 and 3.3.4, fraudulent activities of this nature are either deemed improbable or, if feasible, detectable during the attestation phase. Essentially, we rely on the TEEs to uphold the security assurances they offer.

5.
- **Attack Scenario:** As the code running in the parent instance cannot be attested, the callee has free rein to execute any actions therein. The callee may masquerade

as a genuine participant up until the moment the caller concludes transmitting their data to the TEE application. At that juncture, during the TLS handshake orchestration, the parent instance could divert communication away from the intended public server to a mock server hosted locally. Through this subterfuge, the callee could attempt to intercept caller credentials.

- **Defense Strategy:** Within the TLS handshake process, following the server hello, the client validates the digital certificate provided by the server. This validation logic is integrated into the TEE application code. To bolster security measures, it's conceivable to pre-download digital certificates from commonly accessed services and embed them as a resource within the TEE application. Consequently, the received certificate can be cross-referenced against these pre-downloaded certificates in addition to standard validation procedures. Given the assumptions outlined in section 3.2, wherein flawless digital certificate forgery is deemed implausible and detectable, this defense remains effective. Moreover, any attempt to introduce forged digital certificates into the TEE application would necessitate altering the TEE program, thereby generating a distinct hash from the one expected by the genuine Dolphin solution, thereby detectable during the attestation validation phase.

6.
- **Attack Scenario:** The adversary, a user on the callee side, attempts to gain information by observing the traffic between the parent instance and the public server, or between the callee and the caller. Additionally, they might try to break the encryption through brute-force methods.
- **Defense Strategy:** Both communications are encrypted using robust, industry-standard encryption mechanisms. The encryption algorithms employ sufficiently strong key lengths and cryptographic protocols, making them resistant to brute-force attacks. Therefore, the risk posed by this type of attack can be confidently disregarded.

7.
- **Attack Scenario:** The attacker exploits side-channel vulnerabilities in the TEE hardware or software to gain access to sensitive information processed within the enclave, including caller credentials.
- **Defense Strategy:** As stated in section 3.3.3, this study assumes that the TEE is resilient against all forms of side-channel attacks, rendering this attack scenario infeasible. Furthermore, the solution could be enhanced by implementing countermeasures specifically designed to protect against TEE-based side-channel attacks. However, it is important to note that these countermeasures are outside the scope of this study.

8.
- **Attack Scenario:** An callee exploits side-channel vulnerabilities in the TLS tunnel between the TEE and the public server, targeting the connection established through the parent instance.

- **Defense Strategy:** As outlined in section 3.3.4, this study assumes the TLS tunnel to be robust against side-channel attacks, thereby negating this attack scenario. Although specific defenses against side-channel attacks are available, they are not within the scope of this study.

CHAPTER 5

Experiements and Results

5.1 Experiment Setup

5.1.1 Setting up the Elastic Compute Cloud (EC2) Machine, Amazon Simple Email Service (SES) Configurations

To conduct our experiments, it is necessary to use an EC2 machine that supports Nitro Enclaves. The Amazon m5.4xlarge instance is part of the m5 category, which includes support for Nitro Enclaves. It is important to enable Nitro Enclaves when launching the EC2 instance to ensure that Nitro Enclaves are included in the hardware configuration.

The setup of the Amazon m5.4xlarge EC2 instance for our experiments involved several detailed steps to ensure that all the required libraries and configurations are correctly set up.

1. **Initialization of the EC2 Instance:**

   The following libraries need to be installed to set up the environment. Refer to Appendix 1 for the exact commands.

   - Install the AWS Nitro Enclaves Command Line Interface (CLI).
   - Install Docker.
   - Configure the Nitro Enclaves allocator by editing its configuration file to set the memory allocation as 1024 MB and CPU count to 2. The memory allocation has to be large or equal to the minimum memory the generated enclave image needs to run which is 856 MB for the current version of the code. The number of virtual CPUs (vCPUs) needed depends on whether the parent instance has multi-threading enabled or not. If multi-threading is enabled, then at least 1 vCPU should be left for the parent instance. Another requirement is that the number of vCPUs specified should be an even number. The m5.4xlarge EC2 instance has 16 vCPUs. Therefore, to fulfill all requirements, number of vCPUs we declare should be an even number between 1 and 15.

2. **Rebooting the Instance:**

- Reboot the instance to apply the newly made changes.

3. **Starting the Nitro Enclaves Allocator Service:**
   - Start and enable the Nitro Enclaves allocator service to ensure it runs at startup:

     ```
     sudo systemctl start nitro-enclaves-allocator.service
     sudo systemctl enable nitro-enclaves-allocator.service
     ```

   - Install additional CLI packages required for AWS Nitro Enclaves development:

     ```
     sudo yum install -y aws-nitro-enclaves-cli-devel
     ```

4. **Download the codebase to the EC2 machine**.
5. **Building and Deploying the Enclave:**
   - Move to the server_aws folder.
   - Decide a local port for the enclave to communicate with the parents instance. Include it in the docker-compose.
   - Construct the Docker image required for the enclave:

     ```
     docker build -t server -f Dockerfile .
     ```

   - Build the Nitro Enclave Image File (EIF) using the Docker image:

     ```
     nitro-cli build-enclave --docker-uri server
       --output-file server.eif
     ```

6. **Retrieve the PCR2 value from the output of the enclave build process.**
7. **Run the enclave instance**.

   ```
   nitro-cli run-enclave --eif-path server.eif --cpu-count 2
       --memory 1024 --enclave-cid {enclave-cid}
   ```

8. **Configure Simple Email Service (SES):**
    - Go to Amazon Simple Email Service from the AWS console.
    - Verify the sender and receiver emails.
    - Go to Simple Mail Transfer Protocol (SMTP) settings and generate SMTP credentials.
    - Note down SMTP user name and SMTP password.

5.1.2 Running the Experiments

These experiments were conducted in order to measure the latency introduced by the new setup. During the experiments, latency measurements were collected to evaluate the additional communication overhead introduced by the new Dolphin setup in comparison to the original configuration. To isolate the evaluation to the new extension's components, the original Dolphin setup was excluded from the test. Consequently, the caller actions were simulated on the callee side. Depending on the exact stage within the callee side where this simulation of caller actions was implemented, two primary pathways were identified for consideration.

Main Experiment: Mocking Caller Actions at the Callee's Local Instance

Steps:

1. **Setup and Run the Parent Instance**
    - Modify the security group associated with the EC2 instance to create an inbound rule. This rule should accept TCP traffic on port 8080, ensuring necessary communication capabilities. This enables a local device to establish a websocket connection with the websocket server that runs at the parent instance.
    - Move to client_aws sub folder.
    - Install necessary libraries.

      ```
      ./setup.sh
      ```

    - Run the web-socket server.

      ```
      ./run_client.sh {enclave-cid} {server-port}
      ```

2. **Setup and Run the WebSocket Client**
    - Download the codebase to a local device, not the EC2 machine.
    - Move to the local_setup subfolder.
    - Change the environment variables of the .env file.

```
PCR2={PCR2 value noted down}
IP_ADDRESS={Public IP of the EC2}
ROUNDS={Number of rounds}
LENGTH={Email body length in characters}
ENABLE_PRINTS={Enable for demonstrative logs, but note
                it may cause a performance hit}
```

- Install necessary libraries.

  ```
  ./setup.sh
  ```

- Run the web-socket client.

  ```
  ./run_client.sh
  ```

This experiment is considered the primary one because it resembles the complete end-to-end solution more, compared to other variant. As a result, it provides a more realistic estimation of latency values. The setup involves the local instance of the callee and the EC2 machine operated by the callee. The duration for one iteration was measured from the moment the callee sends the first WebSocket message to the EC2 instance, up to the point when the callee's local instance receives the HTTP response for the email sent by the caller, encrypted with the shared key.

The communication mediums involved in this experiment include the WebSocket connection between the callee's local instance and the EC2 instance, the TLS-based communication between the EC2's parent instance and the email server, and the local socket connection between the parent instance and the Nitro Enclave. This setup captures nearly all the new communication hops introduced by the extension, excluding the relay of the attestation document back to the client, the transmission of the double encrypted ciphertext from the caller to the server, and the caller receiving the corresponding response from the server. These excluded steps would involve the original Dolphin-based message passing via mobile phones. Since the conducted tests exclude the aforementioned messages, the results only provide a lower bound on the additional overhead introduced by the new extension.

Latency tests were performed with messages of three different sizes: 1024, 5000, and 10000 characters. For each message size, 25 end-to-end communication rounds were conducted, and the average time was calculated. The message sizes were configured using the .env file. After each test run, both the parent and enclave instances were restarted as they are programmed to self-terminate after the communication concludes.

Secondary Experiment: Mocking caller actions at the parent instance of the EC2.

Steps:

1. **Setup and Run the Parent Instance**
   - Move to client_aws sub folder.

- Change the environment variables of the .env file.

```
PCR2={PCR2 value noted down}
ROUNDS={Number of rounds}
LENGTH={Email body length in characters}
ENABLE_PRINTS={Enable for demonstrative logs, but
                note it may cause a performance hit}
```

- Install necessary libraries.

```
./setup.sh
```

- Run the web-socket server.

```
./run_client.sh {enclave-cid} {server-port}
```

As described in the design architecture section 4, which abstracts the TEE's basic functionalities rather than focusing on one specific modern TEE, even though AWS Nitro Enclaves were used for the implementation of the proposed extension, the ideal application is if the TEE is included in the callee's local instance. To estimate the overhead time added by such a solution, the WebSocket setup was excluded, and the caller actions were simulated in the parent instance of the EC2 instance.

The communication mediums tested in this experiment include the TLS-based communication between the EC2's parent instance and the email server, and the local socket connection between the parent instance and the Nitro Enclave. This setup captures the same concepts intended to be assessed by the main experiment but focuses on a scenario where the TEE is integrated into the local device of the callee.

Latency tests for the secondary experiment were conducted with the same message sizes as the main experiment: 1024, 5000, and 10000 characters. For each message size, 25 end-to-end communication rounds were conducted, and the average latency time was calculated. As with the main experiment, the message sizes were configured using the .env file, and both the parent and enclave instances were restarted after each run.

5.2 Results

The results for the main experiment are demonstrated in Fig. 7.1. The x-axis represents the message size in characters, while the y-axis represents the average time taken. Since multiple tests were conducted, minimum and maximum error margins are also identified and represented in the chart in blue color. The increase in duration time as the message size grows can be attributed to several key factors. Firstly, as the size of the message increases, the time required for encryption and decryption also increases, thereby contributing to longer overall communication times. Additionally, larger message sizes typically result in more data packets being sent, which can lead to increased latency as these packets traverse the network. It should also be noted that the rate at which the average time increases is not linear. When the message

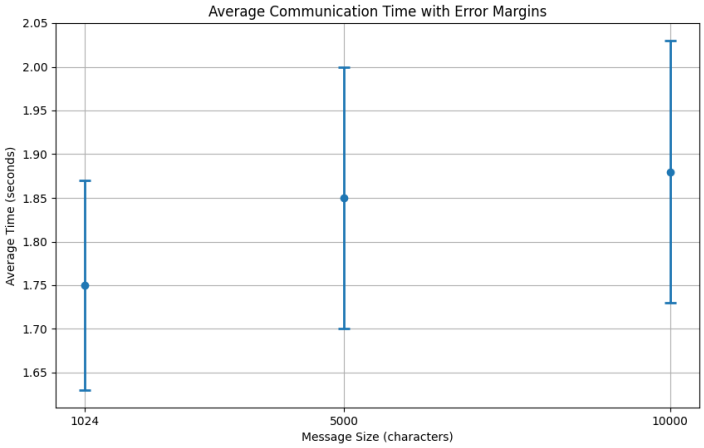


Figure 5.1: Average Latency of Additional Communication Rounds in the Main Experiment for Various Message Sizes in the Extended Dolphin Setup

size nearly multiplied by 10, from 1024 characters to 10000 characters, the average execution time only increased by approximately 7.5%. The time consumed in the communication process is significantly impacted by several factors. These include setting up the WebSocket channel, setting up the TLS connection, internet-based communication between the parent instance and the SES server, internet-based communication when sending messages between the WebSocket server and the local WebSocket client, and sending significantly large files such as the attestation document back and forth, which are relatively larger than the message sizes. Each of these steps plays a bigger role in contributing to the overall latency compared to the message sizes. These results demonstrate that while there is an increase in communication time with larger message sizes, the overall impact remains minimal.

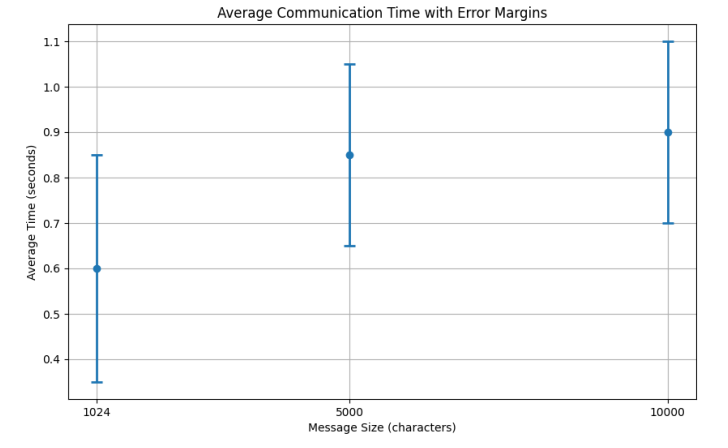


Figure 5.2: Average Latency of Additional Communication Rounds in the Secondary Experiment for Various Message Sizes in the Extended Dolphin Setup

Figure 7.2 illustrates the results of the secondary experiment. The x-axis denotes the message size in characters, while the y-axis indicates the average communication time in seconds. The blue bars represent the minimum and maximum error margins, highlighting the variability observed across multiple test runs. The data shows a clear trend: communication time increases with the size of the messages.

Comparing the results of the secondary experiment with the main experiment, it is evident that the latency values for similar message sizes are significantly lower. For instance, sending a 5000-character email takes over 1.8 seconds in the main experiment, whereas it takes less than 0.9 seconds in the secondary experiment, which is less than 50% of the corresponding value in the WebSocket-based setup. This improved performance is expected, as the secondary setup avoids internet communication via WebSockets, with all communications being internal and local except for interactions with the SES servers. These findings suggest that the performance of the proposed solution can be markedly enhanced if a local TEE, such as Intel SGX, is employed and the callee-side code is adapted to be compatible with that particular TEE framework, instead of using AWS Nitro Enclaves.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

The analysis in section 4.3, which considered all potential attacks by the callee and the robustness of the proposed solution, demonstrates that the new extension indeed provides confidentiality guarantees to the caller. It shows that the callee cannot practically breach confidentiality, and attempts to deviate from the protocol, such as not executing the code in a TEE or modifying code to retrieve caller information, can be detected via the attestation process. Therefore, the goals of ensuring confidentiality of caller credentials and data, and verifying the trust are achieved under the stated assumptions. Although AWS Nitro Enclaves were used for the concrete implementation, the concept outlined in section 4.1 is agnostic of any specific TEE implementation that is commercially available. While the current implementation relies on AWS Nitro Enclaves for ease of implementation and Python support compatible with the Dolphin code base, the ideal scenario is to transition to more traditional TEEs like Intel SGX. If the solution is made compatible with a locally integrated TEE of the callee, instead of a cloud device, the latencies would significantly reduce as all communication within the callee side would be internal to one device. The secondary experiments, which involved mocking the caller actions within the parent instance of the cloud device, support this claim. Given these points, it could be argued that the goal in section 3.2.2, which aims to achieve no considerable performance impact, has also been achieved. Therefore, it can be concluded that under the assumptions defined in section 3.3, all the goals declared in section 3.2 have been successfully achieved.

6.2 Future Work

Future work could focus on making the solution compatible with traditional TEEs such as Intel SGX, enhancing practicality and efficiency by reducing delays. Another significant improvement would be developing robust defenses within the solution's code against all forms of side-channel attacks, ensuring comprehensive protection beyond the capabilities of the TEE alone.

Additionally, a crucial enhancement could involve developing an enclave measurement that calculates a hash dependent only on the user code and associated libraries, excluding the build environment. This streamlined approach would simplify the attestation verification process for the caller, eliminating the need to precalculate values like PCR2 in AWS Nitro based frameworks.

Bibliography

[1] "Sri lanka restricts access to social media amid protests." `https://www.dw.com/en/sri-lanka-restricts-access-to-social-media-platforms-amid-protests/a-61343824`. → [p1]

[2] "2019 internet blackout in iran." `https://en.wikipedia.org/wiki/2019_Internet_blackout_in_Iran`, January 2024. → [p1], [p6]

[3] Africa News, "Ethiopia restores internet access after shutdown for exams," June 2017. http://www.africanews.com/2017/06/08/ethiopia-restores-internet-access-after-shutdown-for-exams/. → [p1], [p6]

[4] The Guardian, "Iraq shuts down the internet to stop pupils cheating in exams," May 2016. https://www.theguardian.com/technology/2016/may/18/iraq-shuts-down-internet-to-stop-pupils-cheating-in-exams. → [p1], [p6]

[5] Editorial, "Internet shutdown in the republic of the congo on election day," Mar. 2021. https://netblocks.org/reports/internet-shutdown-in-the-republic-of-the-congo-on-election-day-xAGR398z. → [p1], [p6]

[6] A. for Submission, "Anonymized for submission. private communication with a humanitarian field worker in syria.." September 2017. → [p1], [p6]

[7] NDTV, *Internet services restricted in 13 districts of Haryana, India*, November 2017. https://tinyurl.com/y5646kz9. → [p1], [p6], [p11], [p21]

[8] Tor Project, *Tor at the Heart: Bridges and Pluggable Transports*. https://blog.torproject.org/tor-heart-bridges-and-pluggable-transports/. → [p1], [p5], [p6]

[9] *Hyphanet, The original Freenet.* https://www.hyphanet.org/index.html. → [p1], [p5]

[10] A. Pradeep *et al.*, "Moby: A blackout-resistant anonymity network for mobile devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, pp. 247–267, July 2022. https://doi.org/10.56553/popets-2022-0071. → [p1], [p8]

[11] A. Lerner *et al.*, "Rangzen: Anonymously getting the word out in a blackout," *ArXiv.org*, December 2016. arxiv.org/abs/1612.03371. → [p1], [p8]

[12] *Bridgefy: Offline Messaging App & SDK.* https://bridgefy.me/. → [p1], [p9]

[13] P. Kumar Sharma *et al.*, "Dolphin: A cellular voice based internet shutdown resistance system," *Proceedings on Privacy Enhancing Technologies*, vol. 2023, pp. 589–607, January 2023. https://doi.org/10.56553/popets-2023-0034. → [p1], [p9], [p22]

[14] S. Li and N. Hopper, "Mailet: Instant social networking under censorship," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, pp. 175–192, April 1 2016. → [p2], [p12]

[15] J. Lind, O. Naor, I. Eyal, F. Kelbert, P. Pietzuch, and E. G. Sirer, "Teechain," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019. → [p2], [p13]

[16] S. Sasy and I. Goldberg, "Consensgx: Scaling anonymous communications networks with trusted execution environments," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 331–349, 2019. → [p2], [p13]

[17] M. Sabt *et al.*, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015. `https://hal.archives-ouvertes.fr/hal-01246364/file/trustcom_2015_tee_what_it_is_what_it_is_not.pdf`, `https://doi.org/10.1109/Trustcom.2015.738`. → [p2], [p13]

[18] "Trusted execution environment," Feb. 23 2021. `en.wikipedia.org/wiki/Trusted_execution_environment`. → [p2], [p13]

[19] D. Goldschlag, M. Reed, and P. Syverson, *Onion Routing*, 1999. → [p5]

[20] *I2P Official Website.* https://geti2p.net/en/. → [p5]

[21] "Garlic routing." Retrieved April 22, 2024, from `https://geti2p.net/en/docs/how/garlic-routing`. → [p5]

[22] L. H. Newman, "Russia takes a big step toward internet isolation," January 2020. https://www.wired.com/story/russia-internet-control-disconnect-censorship/. → [p6]

[23] "Internet censorship (part 2): The technology of information control." https://townsendcenter.berkeley.edu/blog/internet-censorship-part-2-technology-information-control. → [p6]

[24] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg, "Sok: Making sense of censorship resistance systems," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 37–61, 2016. → [p6]

[25] Tor Project, *Obfs4.* https://support.torproject.org/glossary/obfs4/. → [p6]

[26] Tor Project, *Snowflake.* https://snowflake.torproject.org/. → [p6]

[27] Tor Project, *How to Use the Meek Pluggable Transport.* https://blog.torproject.org/how-use-meek-pluggable-transport/. → [p6]

[28] "Decoy routing: Toward unblockable internet communication." Rhttp://www.freehaven.net/anonbib/cache/foci11-decoy.pdf. → [p6]

[29] C. B. Ian Goldberg, *Slitheen: Perfectly Imitated Decoy Routing through Traffic Replacement.* In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016. https://cypherpunks.ca/ iang/pubs/slitheen-ccs16.pdf. → [p6]

[30] D. Ellard and et al., "Rebound: Decoy routing on asymmetric routes via error messages," in *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pp. 91–99, 2015. → [p7]

[31] T. Guardian, "Record number of countries enforced internet shutdowns in 2022 – report," February 28 2023. https://www.theguardian.com/global-development/2023/feb/28/internet-shutdowns-record-number-countries-2022-report. → [p7]

[32] Wired, "Welcome to the era of internet blackouts," 2023. https://www.wired.com/story/cloudflare-internet-blackouts-report/. → [p7]

[33] S. A. Network, "#keepiton organising the global effort to end internet shutdowns." https://solidarityaction.network/media/keep-it-on.pdf. → [p7]

[34] A. Now, "Internet shutdowns in eastern europe and central asia: Attacks on human rights." https://www.accessnow.org/press-release/keepiton-internet-shutdowns-2022-eeca/. → [p7]

[35] A. Now, "Internet shutdowns in mena in 2022: Continued abuses and impunity." https://www.accessnow.org/press-release/keepiton-internet-shutdowns-2022-mena/. → [p7]

[36] "Cloudflare." https://www.cloudflare.com/en-gb/. → [p7]

[37] M. R. Albrecht, J. Blasco, R. B. Jensen, and L. Mareková, "Mesh messaging in large-scale protests: Breaking bridgefy," in *Cryptographers' Track at the RSA Conference*, pp. 375–398, Springer, 2021. → [p9]

[38] M. R. Albrecht *et al.*, "Breaking bridgefy, again: Adopting libsignal is not enough," 2022. www.research-collection.ethz.ch/handle/20.500.11850/580228. → [p9]

[39] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," *Journal of Cryptology*, vol. 33, no. 4, pp. 1914–1983, 2020. https://doi.org/10.1007/s00145-020-09360-1. → [p9]

[40] Business World, *Internet shutdown in response to mega public gathering in India*, October 2020. https://tinyurl.com/yxa9e32u. → [p11], [p20]

[41] Forbes, *Russian Authorities "Secretly" Shutdown Moscow's Mobile Internet: Report*, August 2019. https://tinyurl.com/47pnarm2. → [p11], [p21]

[42] Telenor, *Myanmar shuts down Internet but allows cellular services to function*, May 2020. https://tinyurl.com/up2ytfo. → [p11], [p21]

[43] C. K. LaDue, V. V. Sapozhnykov, and K. S. Fienberg, "A data modem for gsm voice channel," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 4, pp. 2205–2218, 2008. → [p11]

[44] M. Perić, P. Milićević, Z. Banjac, and B. M. Todorović, "An experiment with real-time data transmission over global scale mobile voice channel," in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, pp. 239–242, IEEE, 2015. → [p11]

[45] B. T. Ali, G. Baudoin, and O. Venard, "Data transmission over mobile voice channel based on m-fsk modulation," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 4416–4421, IEEE, 2013. → [p11]

[46] A. Beimel, "Secret-sharing schemes: A survey," in *Coding and Cryptology - Third International Workshop, IWCC 2011*, (Qingdao, China), Proceedings, May 30-June 3 2011. → [p12]

[47] E. Hoogerwerf, D. Van Tetering, A. Bay, and Z. Erkin, "Efficient joint number generation for secure multi-party computation," in *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021* (S. D. C. di Vimercati and P. Samarati, eds.), pp. 436–443, SciTePress, 2021. → [p12]

[48] J. Ménétrey, C. Göttel, A. Khurshid, M. Pasin, P. Felber, V. Schiavoni, and S. Raza, "Attestation mechanisms for trusted execution environments demystified," *arXiv preprint arXiv:2206.03780*, 2022. https://arxiv.org/pdf/2206.03780.pdf. → [p13], [p14]

[49] "Intel software guard extensions (sgx)." https://01.org/intel-softwareguard-extensions. → [p13]

[50] "Trustzone." https://developer.arm.com/ip-products/security-ip/trustzone. → [p13]

[51] A. W. Services, "Nitro enclaves architecture diagram." `https://aws.amazon.com/ec2/nitro/nitro-enclaves/`. → [p15], [p55]

[52] A. W. Services, *The Security Design of the AWS Nitro System*. `https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf`. → [p14]

[53] A. Name, "Privacy preserving deep learning with aws nitro enclaves." https://towardsdatascience.com/privacy-preserving-deep-learning-with-aws-nitro-enclaves-74c72a17f857. → [p15]

[54] "Amazon ec2." `https://aws.amazon.com/ec2/`. → [p15]

[55] "Nitro enclaves command line interface." `https://docs.aws.amazon.com/enclaves/latest/user/nitro-enclave-cli.html`. → [p15], [p32]

[56] "Amazon linux 2023." `https://aws.amazon.com/linux/amazon-linux-2023/`. → [p15], [p32]

[57] "Ubuntu 20.04 - how to install nitro cli from github sources." `https://github.com/aws/aws-nitro-enclaves-cli/blob/main/docs/ubuntu_20.04_how_to_install_nitro_cli_from_github_sources.md`. → [p15], [p32]

[58] Amazon Web Services, "Verifying the nitro enclave image signature." https://docs.aws.amazon.com/enclaves/latest/user/verify-root.html. → [p15]

[59] A. W. Services, "Setting up attestation for nitro enclaves." https://docs.aws.amazon.com/enclaves/latest/user/set-up-attestation.html. → [p15]

[60] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing." `https://csrc.nist.gov/csrc/media/events/physical-security-testing-workshop/documents/papers/physecpaper19.pdf`. → [p16]

[61] R. Press, "Side-channel attacks explained: Everything you need to know." `https://www.rambus.com/blogs/side-channel-attacks/`. → [p16]

[62] W. Contributors, "Side-channel attack." `https://en.wikipedia.org/wiki/Side-channel_attack`. → [p16]

[63] U. of California Sen Diego, "Cse 127 lecture 6: Side-channel attacks." `https://cseweb.ucsd.edu/classes/wi22/cse127-a/scribenotes/6-sidechannels-notes.pdf`. → [p16]

[64] M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon, "Algebraic side-channel attacks on the aes: Why time also matters in dpa," in *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems (CHES 2009)*, pp. 97–111, 2009. → [p16]

[65] H. Kario, "Everlasting robot: the marvin attack." `https://eprint.iacr.org/2023/1442.pdf`. → [p16]

[66] K. C. Mubashwir Alam, "Making your program oblivious: A comparative study for side-channel-safe confidential computing," in *IEEE CLOUD 2023*, pp. 97–111, 2023. → [p16], [p17]

[67] W. Wang, "Side channel risks in hardware trusted execution environments (tees)." `https://heartever.github.io/files/SideChannelRisks.pdf`. → [p16]

[68] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, p. 991–1008, 2018. → [p16]

[69] A. Muñoz, R. Ríos, R. Román, and J. López, "A survey on the (in)security of trusted execution environments." `https://www.sciencedirect.com/science/article/pii/S0167404823000901`. → [p16]

[70] D. Page, "Partitioned cache architecture as a side-channel defence mechanism." `https://eprint.iacr.org/2005/280.pdf`. → [p17]

[71] Intel, "Guidelines for mitigating timing side channels against cryptographic implementations." `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html`. → [p17]

[72] D. Das, S. Maity, S. B. Nasir, S. Ghosh, A. Raychowdhury, and S. Sen, "High efficiency power side-channel attack immunity using noise injection in attenuated signature domain." `https://arxiv.org/pdf/1703.10328`. → [p17]

[73] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors." `https://eprint.iacr.org/2018/418.pdf`. → [p17]

[74] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A survey of microarchitectural side-channel vulnerabilities, attacks and defenses in cryptography." `https://arxiv.org/abs/2103.14244`. → [p17]

[75] "Amazon simple email service (ses)," 2009. https://aws.amazon.com/ses/. → [p31]

[76] "Google api python client." `https://github.com/googleapis/google-api-python-client`. → [p31]

[77] "Less secure apps: Your google account termination date of may 30th, 2022." `https://support.google.com/accounts/thread/154305711/less-secure-apps-your-google-account-termination-date-of-may-30th-2022`. → [p31]

[78] "Using oauth 2.0 for web server applications." `https://developers.google.com/identity/protocols/oauth2/web-server#httprest_1`. → [p31]

[79] "Using oauth 2.0 for server to server applications." `https://developers.google.com/identity/protocols/oauth2/service-account#httprest`. → [p31]

[80] "Amazon machine images (ami)." `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html`. → [p32]

[81] "coincurve." `https://pypi.org/project/coincurve/5.0.0/`. → [p35]

List of Figures

Appendix 1

- **Installation of Essential Packages in the EC2 Machine:**
  - Execute a comprehensive update of the instance's packages:

    ```
    sudo yum update -y
    ```

  - Install the AWS Nitro Enclaves Command Line Interface (CLI):

    ```
    sudo yum install -y aws-nitro-enclaves-cli
    ```

  - Install Docker:

    ```
    sudo yum install -y docker
    ```

  - Initiate the Docker service to support container operations:

    ```
    sudo service docker start
    ```

  - Set user-privileges for Docker:

    ```
    sudo usermod -aG ne ec2-user
    sudo usermod -aG docker ec2-user
    ```

  - Configure the Nitro Enclaves allocator by editing its configuration file to set the memory allocation as 1024 MB and CPU count to 2.

    ```
    sudo vim /etc/nitro_enclaves/allocator.yaml
    ```

Labor omnia vincit (Work conquers all) – Virgil