TrustGlass: Human-Computer Trusted Paths with Augmented Reality Smart Glasses

Hélio Borges, Daniel Andrade, João Nuno Silva, Miguel Correia

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Lisboa, Portugal
{helio.borges, daniel.andrade, joao.n.silva, miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—Humans constantly interact with computing devices. Many times, these interactions involve sensitive information and/or sensitive commands, leading to the need for *trusted paths* between humans and computers. For computer-to-computer interactions, secure communication is not an issue thanks to cryptographic protocols such as Transport Layer Security (TLS) and the IP Security Protocol (IPSec). However, the same does not yet apply to interactions between humans and computers due to the inability of humans to execute non-trivial cryptographic algorithms. This results in interactions that are vulnerable to shoulder surfing, man-in-the-browser malware, and web page spoofing, among other attacks. This consequently puts at risk the use of sensitive services in uncontrolled environments.**

We present *TrustGlass*, a scheme that uses augmented reality smart glasses to solve this security problem. TrustGlass uses such glasses to extend human capabilities, to execute the cryptographic algorithms needed to establish a trusted path between the user and the trusted service, which can be executed in a Trusted Execution Environment (TEE). With this approach, we can guarantee that only the user equipped with the glasses is capable of interacting with the service. We implemented TrustGlass using commercial smart glasses and show experimental performance and usability results.

***Index Terms*—Trusted Path, Augmented Reality Smart Glasses, Human-Computer Interaction, TEE**

I. Introduction

Humans are continually engaging with computing devices such as computers, cell phones, Point-of-Sale (PoS) devices, and Automated Teller Machines (ATMs). In general, a person does these accesses without much thought of whether the interaction occurs with the real back-end *service* or with some malicious intermediary. Similarly, a person may not take into account the environment in which they are doing these accesses. This potentially false sense of security is to be expected, as people do not tend to suspect a service that operates in a seemingly correct manner or of their environment. Furthermore, computer-to-computer communication is generally considered secure thanks to established protocols such as Transport Layer Security (TLS) [1] and Internet Protocol Security (IPsec) [2].

In *human-to-computer communications*, users tend to expect that the data they insert or receive in/from a service will not be revealed. Similarly, users expect that the commands they issue on a service will not be modified. However, these expectations are sometimes not met, as the risks associated with using computing devices are still present. In particular, we highlight the risks associated with using computing services in unsafe environments, through unsafe devices, or unsafe servers. In these situations, users are at risk of attacks such as man-in-the-middle (where the attacker intercepts the communication and masquerades as one of the two entities [3]) and shoulder surfing (spying on the input of the victim). These attacks target the communication path between a user and a device/server, which is currently poorly protected.

If we look at the established protocols for secure computer-to-computer communications, we notice that the element they share is the use of *cryptography*. For computer-to-computer communications, cryptography is used mainly to ensure that the data is authentic, not modified, and not read by an adversary (authenticity, integrity, confidentiality). However, in human-to-computer communications, the use of cryptography is cumbersome, as a regular person is not able to perform cryptographic operations by themselves, using standard algorithms and key lengths [4].

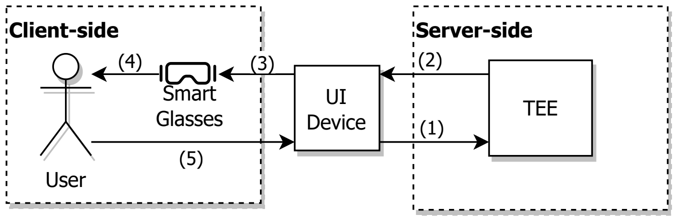


Fig. 1. Overview of the TrustGlass entities. The service can run in a local server, a cloud server, be an ATM, etc., which we designate *server*. The UI device can be a terminal (with screen and keyboard), and the TEE an SGX enclave.

In this paper, we present *TrustGlass*, a solution to secure human-to-computer communications (cf. Fig. 1). By using augmented reality smart glasses – *smart glasses* or *glasses* for short – [5], we equip the user with a tool that can bridge the gap between the server and the user's eyes, as well as perform cryptographic operations on their behalf. The glasses are trusted (they are not connected to any network or peripheral) and are considered to be an extension of the user's capacities. Using smart glasses, TrustGlass establishes a *trusted path* between the user and the service with which they are interacting. With this trusted path, TrustGlass gives the user the ability to both communicate securely with the remote service, regardless of the environment or malicious code in the server, and to independently recognize whether the path is trustworthy or not via the path's trust signals. Some examples of *applications* for our trusted paths would be ATM machines, PoS devices, enclaves inside untrusted servers, web or cloud applications accessed through a browser, and remote digital wallets (that store keys to access Blockchain assets).

We implemented TrustGlass[1] using a set of technologies. For the client side, we used Epson Moverio BT-35E smart glasses and an Android application. On the server side, services are isolated using Trusted Execution Environments (TEEs), more precisely, Intel Software Guard Extensions (SGX) enclaves [6]. The server-side uses the C/C++ Intel SGX trusted library. We evaluated TrustGlass using a mock ATM application. We performed usability experiments with 21 users.

The main contributions of this paper are 1) a trusted path architecture and protocol capable of securing communications directly between a user and a remote service; 2) a new approach for securing human-computer interactions; 3) and an experimental assessment of the solution.

II. Background

A. Trusted Paths

To define what a trusted path is, we first need to establish what a secure channel is. A secure channel, according to NIST [7], is a path between two devices that can ensure:

- *Confidentiality*: data should be accessible solely by authorized entities.
- *Integrity*: data has not been tampered with in an unauthorized manner.
- *Authenticity*: each message should be genuine and verifiable.
- *Freshness*: each response was recently generated.

On the other hand, the term *trusted path* is defined in RFC 4949 [3] as: *"A mechanism by which a user (...) can communicate directly with the security functions of the information system [(the service)] with the necessary confidence to support the system security policy."*

A trusted path ensures the same security properties that a secure channel does. Furthermore, it informs the user of the path's trustworthiness via a *trust indicator*, e.g., a LED or a small colored circle on the screen [8]–[10].

A trusted path should be impossible to be replicated by an attacker and has to span the entire communication path, from the user to the service [8]. The service is considered secure, i.e., part of the TCB [11], and can be, for example, an application or a service hosted on a trusted computer or running in a TEE. Trusted paths also have to be: 1) effective in allowing a user to recognize trusted data from untrusted data with minimal user interaction in the decision-making; and 2) not intrusive in terms of user experience [9].

B. Trusted Execution Environments

The service has to be trusted, i.e., part of the TCB. Therefore, we isolate it from the rest of the server by running it in a TEE. A TEE is an isolated processing environment, created with the support of the processor, capable of ensuring the confidentiality and integrity of data from the host system (server in our case) [12]. An application or service loaded in the trusted environment provided by the TEE is designated a Trusted Application (TA), and is guaranteed to be isolated from the rest of the device's software, including privileged software like the Operating System (OS). There are several TEE implementations, e.g., Intel SGX [6], [13], AMD Secure Encrypted Virtualization (SEV) [14], and ARM TrustZone [15].

To obtain isolation, a TEE employs multiple mechanisms, such as: *Secure Boot*, to certify that only specified code is being loaded in the secure environment; *Remote Attestation*, so that TEE users can confirm its trustworthiness; *Secure Scheduling*, so it operates in parallel with the host OS, without impacting performance or security; *Inter-Environment Communication*, which allows a TEE to interact with the host system if needed; *Trusted Input/Output (I/O) Path*, allowing the TEE to interact with peripherals without the transmitted data being compromised; and *Secure Storage*, where data is stored with the guarantee of integrity, and optionally confidentiality.

Intel SGX [6], [13] is a prominent TEE implementation, available in a range of Intel processors. An SGX TEE is designated an enclave, a hardware-protected memory space in which the TA runs. The TA is part of a host application that runs in the commodity OS, alongside other applications. This host can request the TA to perform computations with its protected code and/or data.

C. Augmented Reality Smart Glasses

Augmented reality is a combination of real-life scenes with computer-generated images and sound in real time. This can be obtained with three main kinds of devices: 1) Head Mounted Displays (HMDs), where computer-generated imagery is displayed between the user's eyes and the real-world scene; 2) Handheld displays, such as smartphones or tablets, which combine the computer-generated imagery with a real-world recording in the device; and 3) Spatial Augmented Reality, which overlays the computer-generated imagery in the real–world via video-projectors and holograms, foregoing the need for user equipment [5], [16].

Not all *smart glasses* support augmented reality (for example, some do not have displays), but those that do fall into the HMD category. These devices take the form of regular glasses, but with displays embedded in or in front of the lenses. They are usually equipped with enough computational power to run an OS such as Android. This, when coupled with their graphic capabilities, means that these are wearable computers that can expand the capabilities of the human mind, vision, and possibly other senses. Smart glasses may have, for example, front-facing cameras, customizable buttons, and Bluetooth/Wi-Fi connectivity, depending on the model.

[1]https://github.com/hsven/trust-glass

III. TrustGlass

TrustGlass is our solution for creating trusted paths between users and the services they are accessing. TrustGlass leverages the use of smart glasses with support for augmented reality (HMDs with at least one front-facing camera). Services may be protected by running inside TEEs.

A. Threat Model and Assumptions

The TCB of TrustGlass consists exclusively of the TEE, the TrustGlass libraries, and the smart glasses. We do not trust any other element, such as the user's environment, the user interface (UI) device, the host device of the TEE, or anything in-between. Side-channel attacks (e.g., [17], [18]) and denial-of-service attacks are out-of-scope. We assume that neither the user nor the TEE are impersonated during the setup registration and setup phase (Section III-C), i.e., their public keys are correctly exchanged.

Cryptographic schemes: We assume the existence of an IND-CPA and IND-CTXT Authenticated Encryption with Associated Data (AEAD) scheme [19]. In practice, we consider this primitive to be the Advanced Encryption Standard in Galois/Counter-Mode (AES-GCM) algorithm [20]. This AEAD scheme is used to guarantee the confidentiality and integrity of the plaintext sent from the TEE to the user. It also guarantees authenticity, as the protocol ensures that only the user and the TEE have the session key. We also assume the existence of a key-exchange protocol secure against eavesdroppers and that ensures perfect forward secrecy. We use the Elliptic Curve Diffie-Hellman (ECDH) scheme with ephemeral secrets [21], [22]. Finally, we assume a Key Derivation Function (KDF), specifically HMAC-based KDF (HKDF) [23] with pseudo-random nonces, for generating session keys.

Functions: The AEAD scheme [24] provides the functions $Encrypt(K, N, P, A, IV) \rightarrow C, T$ and $Decrypt(K, N, C, IV) \rightarrow P$ or $FAIL$, where K is a secret key, N is a nonce, P is the plaintext to be encrypted, A is data included in the integrity check but not encrypted, IV is the initialization vector, C is the ciphertext, T is the authentication tag, and $FAIL$ is an error code that occurs when the input text is not authentic. HKDF provides the function $HKDF(K_{In}, S) \rightarrow K_{Out}$, where K_{In} is the input key, S is a unique salt, and K_{Out} is the resulting key. ECDH provides the function $ECDH(K_A^-, K_B^+) \rightarrow K_{A,B}$, where K_A^- is the private key of entity A, K_B^+ is the public key of entity B, and $K_{A,B}$ is the key shared between A and B.

B. Architecture and Overview

The objective of TrustGlass is to provide a *trusted path*, i.e., a secure human-to-computer communication protocol that ensures the four properties listed in Section II-A: Confidentiality, Authenticity, Integrity, and Freshness. It needs to provide these properties bidirectionally, between the user and the TEE. However, as explained later, the confidentiality and integrity guarantees provided in the two directions are different (lower in the user to TEE direction).

The solution involves four core entities (cf. Fig. 1): the user, their smart glasses, a UI device, and the TEE with which the user wishes to communicate. The UI device is any device with an input peripheral (e.g., keyboard or touch screen) and a visual output peripheral (screen), that can work as a terminal to the TEE. The UI device can be, for example, the computer where the TEE runs, or a terminal/smartphone that allows access to the TEE's server. The glasses work as the trust indicator of the trusted path, as they only show the data provided by the TEE.

The arrows in Fig. 1 represent the different steps of the TrustGlass operation loop. In step (1), the UI device sends a user message to the TEE. In step (2), the TEE sends back a response that the UI device renders as a *QR code* [25]. We opted to use QR codes because they serve our purposes well: they provide a machine-readable optical data format that is efficient and includes error correction (the data is acquired correctly even if there are errors in the reading).

The smart glasses scan the rendered QR code in step (3), decodes and decrypts the content, before displaying the resulting plaintext in step (4). The smart glasses are unable to send user input to the TEE so, instead, the TEE's messages prompt the user to input on their UI device, as represented in step (5).

This overview shows that communication is *asymmetric*: the TEE sends data to the user through the glasses, while the user sends data to the TEE without directly using the glasses. The way the data is protected in each direction is different. Protecting user input from an eavesdropper is not trivial, as it cannot be encrypted. The interaction can be divided into three phases that we explain below: *registration and long-term setup*, *trusted path establishment* and *user-TEE interaction*.

C. Registration and Long-Term Setup

This initial registration and setup phase is represented in Fig. 2. The figure decomposes the TrustGlass server-side into three software components: 1) a *Server App* that runs outside the TEE and that is needed because the TEE is passive, i.e., runs only when called; 2) the *Service* that runs inside the TEE and provide its functionality, whatever it is; 3) the *TrustGlass library* that runs the TrustGlass server-side code, inside the TEE.

This phase exchanges public keys between the TEE and the glasses, then establishes a shared key using the ECDH scheme. Unlike TrustGlass' normal operation, this setup requires a connection between the smart glasses and the server (e.g., a TLS connection through a network or USB cable).

The process works as follows. The user U prompts the smart glasses SG to connect to the TEE T, providing their ID, ID_U. The smart glasses start by establishing a connection to the TEE via the server app. Next, the smart glasses generate a pair of Elliptic Curve (EC) asymmetric keys and then send the public key K_{SG}^+ and ID_U to the TEE. The TEE generates its own pair of EC asymmetric keys and uses both its private key K_T^- and the received K_{SG}^+ to generate the long-term shared secret key $K_{SG,T}$ using ECDH. This key is stored by the TEE, before sending its public key K_T^+ and ID ID_T to the smart glasses. The smart glasses then generate their copy of $K_{SG,T}$. At the end, the glasses and the service delete their asymmetric key pairs, as they have no further purpose.

D. Trusted Path Establishment

This phase creates a trusted path (a session) between the user and the TEE (see Fig. 3). It is performed whenever the user wants to use the service, any number of times. The phase begins with the user connecting to the TEE on the UI device and introducing ID_U. With this ID, the TEE selects the long-term shared secret key, $K_{SG,T}$. The TEE then generates a nonce, which it uses, along with $K_{SG,T}$, to derive the shared key for the session $Ksession_{SG,T}$, using HKDF. Then, the TEE sends the user the nonce and ID ID_T through the UI device. Once the UI device receives this message, essentially a number, it renders it as a QR code for the smart glasses to scan. The glasses scan the QR code, retrieve $K_{SG,T}$ with the help of ID_T, and use the received nonce to derive their copy of $Ksession_{SG,T}$.

The session is established and data can be exchanged with confidentiality and integrity ensured. However, the user and the service are not mutually authenticated. The user authenticates the service when the service shows knowledge of $Ksession_{SG,T}$. The TEE must send to the glasses a message encrypted with this key, that can then be decrypted with AES-GCM, as explained in the next section. If the session key each endpoint had was different, then the trusted path was not established. How the service authenticates the user is explained at the end of the next section.

E. User-TEE Interaction

With the trusted path established and both endpoints authenticated, the user can now exchange messages with the TEE. This phase is represented in Fig. 4.

TEE to user: For the TEE to send a message to the user, it follows these steps: 1) the TEE prepares the message; 2) to prevent replay attacks, a freshness token is included; 3) a random Initialization Vector (IV) is generated; 4) the message is encrypted with AES-GCM, $Ksession_{SG,T}$, and IV; 5) the ciphered message and the IV are sent to the UI device.

When the UI device receives the message, it renders it as a QR code to be scanned by the glasses. An example of this interaction, in which the glasses display the result of scanning a QR code, is represented in Fig. 5. The glasses are responsible for ensuring that the message received is trustworthy before displaying it to the user. After scanning the image with its front-facing cameras, it decodes it into text, decrypts and checks its integrity with AES-GCM and $Ksession_{SG,T}$, and checks if the freshness token is recent. If the message passes these steps without error, the plaintext result is rendered in the glasses for the user to read. Otherwise, an error message is displayed.

The plaintext shown depends on the application. One option is data that allows the user to send their next input to the service securely, as explained next.

User to TEE: Protecting user input to be sent to the TEE is not trivial, as users insert input in an untrusted UI (e.g., type it on an untrusted keyboard) and cannot encrypt data in their heads (we assume no interface for users to give data to the glasses for encryption). TrustGlass supports two input mechanisms with different security guarantees:

- *Randomized Strings (RS):* the TrustGlass library generates strings of any desired size and provides the user with the (bijective) mapping between those strings and some meaningful input (e.g., `QWert123ty means Confirm, WEqwet123 means Cancel`).
- *Randomized Keyboard (RK):* the TrustGlass library creates a random bijective mapping of keys to characters (e.g., the digit `7` should be typed as `5`). Fig. 5 shows an RK example.

More rigorously, for both mechanisms RS and RK:

1) The TEE sends the glasses two sets: the input set S_i and the encoded set S_e, and a bijection between them $f : S_i \rightarrow S_e$, everything encrypted using AEAD and $Ksession_{SG,T}$. We assume that every bijection is random and used only once, so there is indistinguishability;
2) The user inserts in the input device (e.g., a keyboard) a sequence of encoded inputs $i_1, i_2, ...$ (possibly only one) with $\forall i_i : i_i \in S_e$. This is the step that may be eavesdropped or tampered by the adversary;
3) The TEE receives the input and accepts it if it contains only strings in S_e ($\forall i_i : i_i \in S_e$); if accepted, it uses the bijection to obtain the input the user wanted to provide ($\forall i : inp_i = f^{-1}(i_i)$;

RS: RS might be (but is not) implemented by sending the user options obtained by encrypting the real inputs with the AEAD scheme and some random key K used only once (and that does not leave the TEE), then converting the strings to a typeable format (e.g., base 64). This would provide confidentiality and integrity (without K, it is not possible to obtain or generate the input). However, that solution would be constrained to the number of bits of the input sent, n_b, being a multiple of the size of an AES block ($n_b = k \times 128bits$). Therefore, we generalize it to sequences of characters of arbitrary length n_l, with charsets with an arbitrary number of characters n_c. This leads to the probability that the adversary guesses an input that is valid to be $P_g = n_o/n_c^{n_l}$, where n_o is the number of valid options. For example, with 2 options and 10 characters in base 64, $P_g = 2/(64^{10}) = 1.9 \times 10^{-9}$. Therefore, the confidentiality guarantees are identical to those provided by the scheme based on AEAD; integrity depends on the number of characters sent, but can be as high as needed.

RK: RK provides lower guarantees. Confidentiality is provided if the input has just a few characters, as the characters sent are opaque to an eavesdropper (indistinguishable from any other). However, in general, it is ensured only if the input is not long enough for frequency attacks or other cryptanalysis methods to be viable. Integrity is not provided if the input can be an arbitrary sequence of characters. However, if the service expects a specific input, such as a PIN code, then integrity can be stated as a probability as for RS: the probability that the

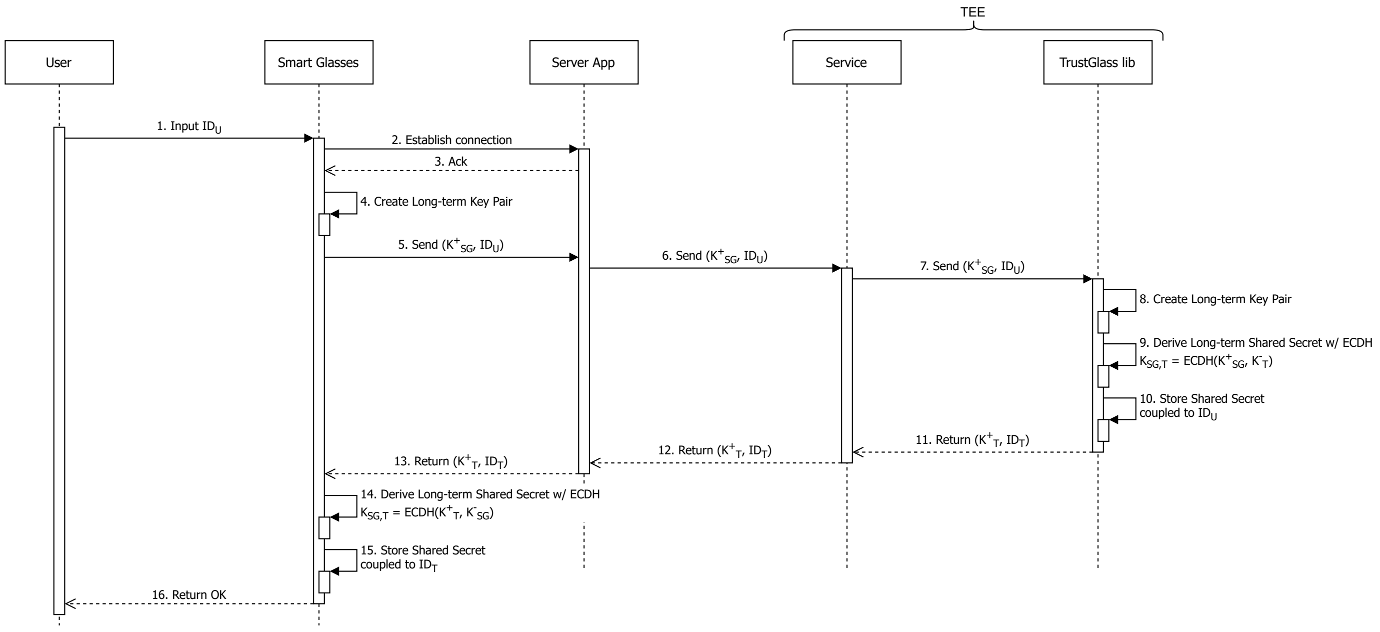

TEE
User
Smart Glasses
Server App
Service
TrustGlass lib
1. Input ID_U
2. Establish connection
3. Ack
4. Create Long-term Key Pair
5. Send (K^+_{SG}, ID_U)
6. Send (K^+_{SG}, ID_U)
7. Send (K^+_{SG}, ID_U)
8. Create Long-term Key Pair
9. Derive Long-term Shared Secret w/ ECDH
$K_{SG,T}$ = ECDH(K^+_{SG}, K^-_T)
10. Store Shared Secret coupled to ID_U
11. Return (K^+_T, ID_T)
12. Return (K^+_T, ID_T)
13. Return (K^+_T, ID_T)
14. Derive Long-term Shared Secret w/ ECDH
$K_{SG,T}$ = ECDH(K^+_T, K^-_{SG})
15. Store Shared Secret coupled to ID_T
16. Return OK

Fig. 2. UML sequence diagram of the registration and long-term setup.

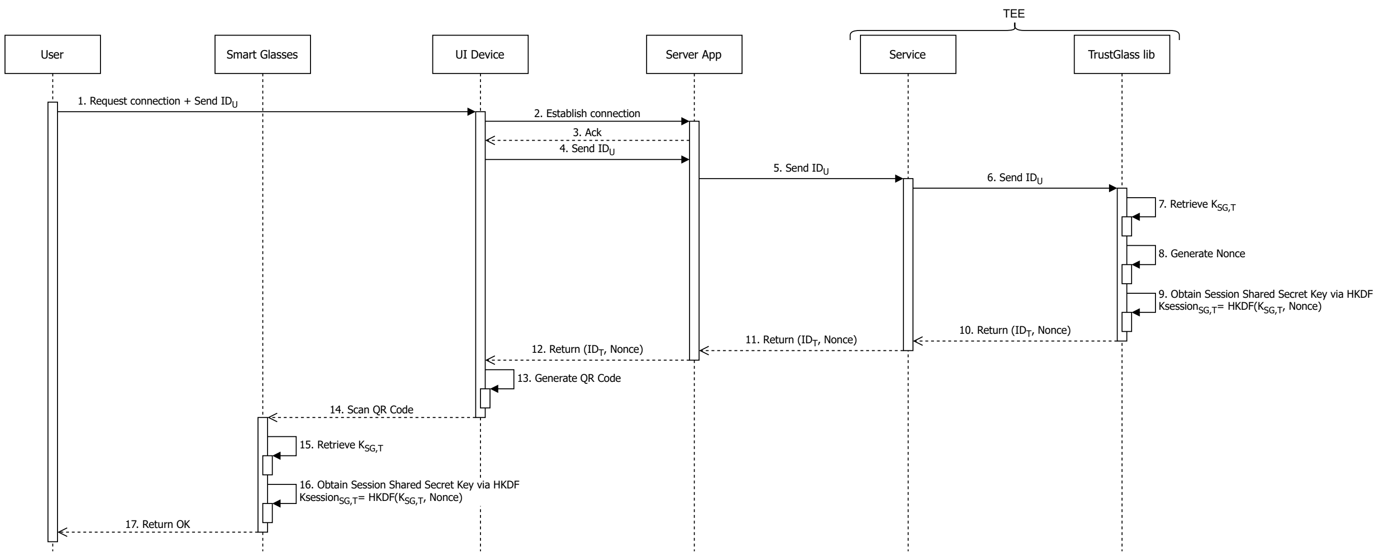

TEE
User
Smart Glasses
UI Device
Server App
Service
TrustGlass lib
1. Request connection + Send ID_U
2. Establish connection
3. Ack
4. Send ID_U
5. Send ID_U
6. Send ID_U
7. Retrieve $K_{SG,T}$
8. Generate Nonce
9. Obtain Session Shared Secret Key via HKDF
$Ksession_{SG,T}$= HKDF($K_{SG,T}$, Nonce)
10. Return (ID_T, Nonce)
11. Return (ID_T, Nonce)
12. Return (ID_T, Nonce)
13. Generate QR Code
14. Scan QR Code
15. Retrieve $K_{SG,T}$
16. Obtain Session Shared Secret Key via HKDF
$Ksession_{SG,T}$= HKDF($K_{SG,T}$, Nonce)
17. Return OK

Fig. 3. UML sequence diagram of the trusted path establishment.

adversary guesses a valid input becomes $P_g = 1/n_c^{n_l}$. With the PIN example, with one valid sequence, the 10 digits and a 4-digit PIN, $P_g = 1/10^4 = 0.0001$.

User Authentication: The service can authenticate the user by providing an RK to request the user's password, then comparing it with a stored password. An alternative is to authenticate only the glasses (not the user directly), using the RS mechanism to present the user with a string that they have to introduce back; if the string is correctly inserted, the service has a high probably to assume that the user is using glasses that have access to $Ksession_{SG,T}$, so were registered with $K_{SG,T}$. If the smart glasses require user authentication to be used, then this alternative option would also authenticate the user.

F. TrustGlass Implementation

Our TrustGlass prototype has two parts: an enclave trusted library (executed in the TEE) and a smart glasses application (executed in the glasses). Both parts of the system implement the three cryptographic schemes: AEAD, ECDH, and HKDF. However, for AEAD, the TEE library implements only AEAD encryption, and the glasses implement only AEAD decryption. The server-side TrustGlass library was implemented in C and

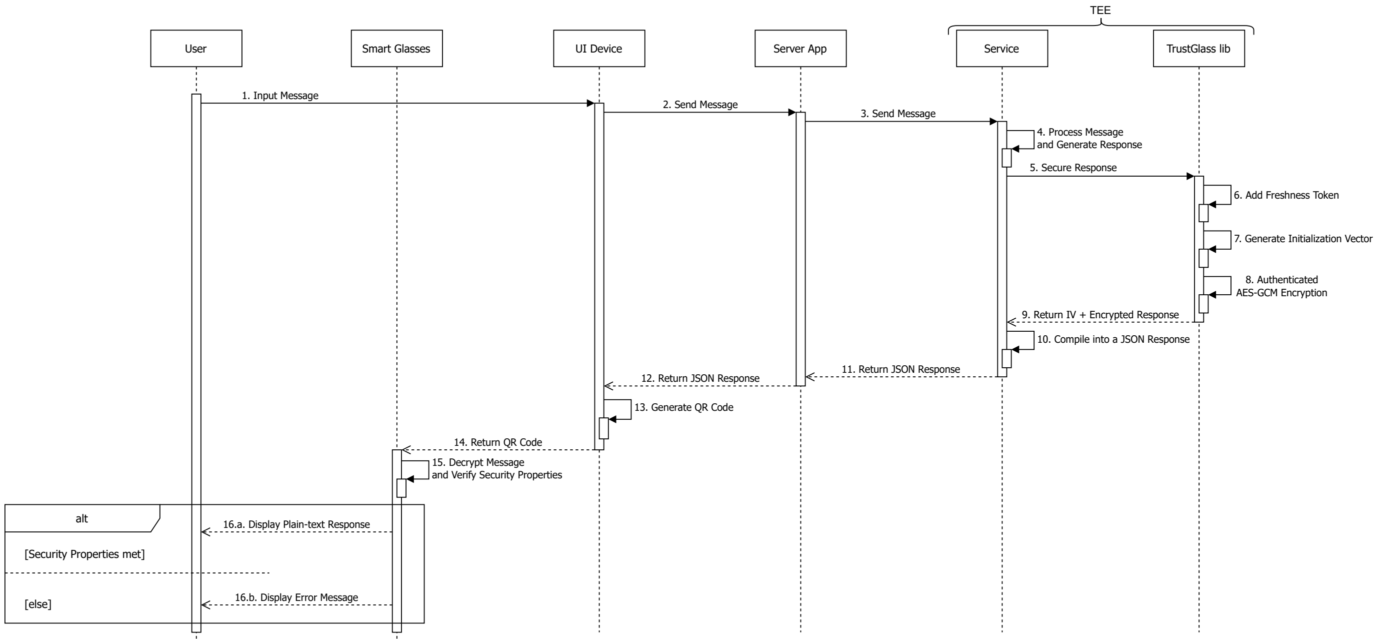
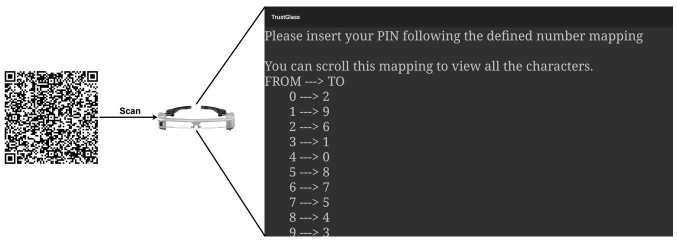

TEE
User
Smart Glasses
UI Device
Server App
Service
TrustGlass lib
1. Input Message
2. Send Message
3. Send Message
4. Process Message and Generate Response
5. Secure Response
6. Add Freshness Token
7. Generate Initialization Vector
8. Authenticated AES-GCM Encryption
9. Return IV + Encrypted Response
10. Compile into a JSON Response
11. Return JSON Response
12. Return JSON Response
13. Generate QR Code
14. Return QR Code
15. Decrypt Message and Verify Security Properties
alt
16.a. Display Plain-text Response
[Security Properties met]
[else]
16.b. Display Error Message

Fig. 4. UML sequence diagram of the message exchange phase.



Please insert your PIN following the defined number mapping

You can scroll this mapping to view all the characters.
FROM ---> TO
0 ---> 2
1 ---> 9
2 ---> 6
3 ---> 1
4 ---> 0
5 ---> 8
6 ---> 7
7 ---> 5
8 ---> 4
9 ---> 3

Fig. 5. Example result of a QR code scan, as seen by the smart glasses user.

C++, using the Intel SGX SDK[2] and the Intel SGX SSL[3] library [13].

The smart glasses part is an Android application written in Java that runs in Epson Moverio BT-35E glasses, coupled with an Epson BO-IC400 controller (the cheapest commercially available option that provides the functionality we need). For QR code scanning, it leverages the scanning API of ML-Kit[4]; for cryptography, it mainly uses functions from Java's default libraries, except for the Bouncy Castle API[5] for the HKDF operation. The main task of this application is to scan QR codes, check security properties, decrypt messages, and then render the plain text to the glasses' lenses.

IV. Evaluation

This section presents the evaluation of TrustGlass. We aim to answer three questions: 1) Does TrustGlass ensure the four security properties? 2) How does TrustGlass perform? 3) Is TrustGlass viable in a real-world scenario?

[2]https://github.com/intel/linux-sgx
[3]https://github.com/intel/intel-sgx-ssl
[4]https://developers.google.com/ml-kit
[5]https://www.bouncycastle.org/

TABLE I
TrustGlass analyzed against the STRIDE methodology.

Attack Category	**Associated Property**	**Property Ensured?**
Spoofing	Authenticity, Freshness	Yes
Tampering	Integrity	Yes
Repudiation	Non-repudiation	Out-of-scope
Information Disclosure	Confidentiality	Yes
Denial of Service	Availability	Out-of-scope
Escalation of Privilege	Authorization	Out-of-scope

We evaluate the security of TrustGlass using the STRIDE model [26]. For performance, we present several micro-benchmarks. Usability is evaluated using an empirical method, with real users (21 participants), and a demonstration application that emulates an ATM, with a questionnaire inspired by the widely adopted System Usability Scale (SUS) questionnaire [27].

A. Security Evaluation

STRIDE allows us to analyze TrustGlass against a set of attack categories. As shown in Table I, TrustGlass ensures the core security properties of trusted paths (confidentiality, integrity and authenticity).

Confidentiality in the TEE to user direction is ensured by encrypting the data with an IND-CPA AEAD scheme, with the shared key $Ksession_{SG,T}$, between the smart glasses and the TEE. This key guarantees that the data sent can only be accessed by the glasses and the user after the TEE sends it. Under the established assumption of no impersonation during setup (cf. Section III-A) and the security of the ECDH scheme, the long-term shared secret key $K_{SG,T}$ established in the setup phase is not disclosed to adversaries. As such, an attacker would not be able to obtain valid session keys to decrypt the TEE-sent data. In relation to confidentiality in the other direction (user to TEE), if the input mechanism RS is used, the input data is confidential because it is encoded in a way that is indistinguishable from any other data (the encoding/bijection is random and changed every time).

Integrity in the TEE to user direction is ensured by encrypting data with an IND-CTXT AEAD scheme, with $Ksession_{SG,T}$, between the smart glasses and the TEE. Integrity in the user to TEE direction is ensured when RS is used because the adversary has no knowledge of the bijection, so it cannot provide a valid input with probability $P_g = n_o/n_c^{n_l}$. This probability can be as high as needed by increasing the number of characters introduced n_l. RK provides the same level of integrity if the TEE expects a specific input, as explained in Section III-E.

Authenticity is ensured due to long-term secret $K_{SG,T}$. During the setup, both entities use ECDH to establish the long-term shared secret, binding it to the ID of the other. With this, there is an implicit authentication of both entities when the trusted path is established, as they need to have the same long-term shared secret to establish a shared session key.

Freshness is achieved in the TEE to user direction by adding a freshness token, given the integrity and authenticity of the communication proved above. This freshness token allows the glasses to verify whether a message is recent, guaranteeing that there is no replay attack. Freshness for the user to TEE direction follows the same principles as was mentioned for integrity. If the integrity of user input is ensured, then the fact that the user was authenticated and can input according to the TEE's instructions means that only the expected user could be sending the messages.

Non-repudiation, Availability and Authorization are not ensured by TrustGlass, but they are also not goals.

B. Performance Evaluation

We evaluated the performance of TrustGlass in terms of several micro-benchmarks. Measurements were obtained in a brightly lit environment, and QR codes were displayed on a 22-inch display, as shown in Fig. 6. The results depend on the lighting of the environment and the noise of the image.

Regarding the number of characters that can be sent to the glasses, the bottleneck is the number of characters that can be encoded in a QR code. In the conditions we had, we managed to use QR code version 40, which is the version that allows us to send the most amount of data [25]. That version uses a 177×177 grid and low error correction level. With this version, we managed to encode 2127 8-bit characters. As the freshness token used in the prototype is a counter, this limit decreases slightly over time as the counter may require more bits.

We measured the scanning and decryption speed of QR codes, across three code sizes (77, 733 and 2733 characters). For each size, we scanned 20 QR codes consecutively, for a total of 60 scans. We obtained, on average, a 274 millisecond wait time between a successful QR code scan and its content being displayed to the user, regardless of the code's size.

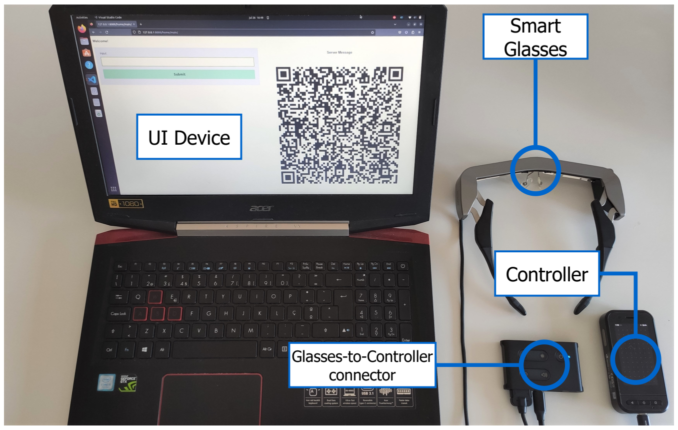


Fig. 6. Usability test setup.

For each size, we measured the maximum distance the glasses could stay away from the screen and still reliably scan a code. For 77 and 733 character codes, the glasses could reliably scan at around 60 and 53 centimeters, respectively. However, for 2733 characters, the distance was reduced to 25 centimeters, due to the limited image quality of the glasses' camera.

C. Usability Tests Setup

To test the usability of TrustGlass, we tested our ATM demonstration application with a group of 21 participants (17 men and 4 women). Of these participants, 18 were between 18-24 years old, a participant was between 25-39 years old, and 2 participants were between 40 and 59 years old. Ten participants use prescription glasses regularly; the others do not. Regarding their formal educational background, there were 9 participants with a Bachelor's degree or equivalent, 7 participants with complete high school, 4 with a Master's degree, and 1 with middle school-level education. 18 participants completely agreed that they are used to modern technology, while the 3 remaining participants were evenly divided between slight disagreement and slight agreement with the statement. When it comes to being used to mixed reality, 16 participants responded between neutral and not being at all used to the technology, while only 5 responded with being either slightly or completely adept with it. The experiment was carried out in Oeiras, Portugal, on a university campus.

The test was structured to take 10 to 15 minutes, including filling out questionnaires, and each participant was asked to follow a short script of steps that simulate a simple balance check and a money withdrawal operation in an ATM. The test was supported by a demonstration application that included these features, implementing an enclave application that uses TrustGlass in its workflow, and the glasses to host the TrustGlass client-side app. The test setup can be seen in Fig. 6.

Before the test, each participant was asked to complete a consent form. The participant was then told that they would play the role of an ATM user, and an explanation of how to use the system was given.

After the test, each participant was asked to complete a questionnaire. This questionnaire asked for demographic data and then some questions regarding the system and protocol. Our goal with the questionnaire was to evaluate the usability of the protocol itself. This means that we focused on evaluating the mechanisms employed in TrustGlass, such as the RS and RK mechanisms, and less so on the demonstration application. As such, we decided on four aspects of the protocol which we wished to evaluate with the questionnaire:

- Effectiveness: whether the tasks were doable;
- Efficiency: how difficult were the tasks are;
- Satisfaction: how comfortable were the tasks for the participant;
- Security: whether the tasks felt secure to the participant.

Our questions were based on the widely-adopted SUS questionnaire [27], taking care to select questions relevant to our goals. Aside from these questions, we asked the following more specific questions:

- I found the presented text in the glasses comfortable to read.
- I found it simple to write text according to the system's instructions.
- I feel safer accessing sensitive data when using the system.
- I found the system was clear in its intention of improving security in communicating with the machine.

D. *Usability Results*

Regarding how much time it took to complete the test tasks, we expected the participants to take about five minutes if no issues occurred. The results confirmed our expectations, with an average of 5:25 minutes and a standard deviation of 3:23 minutes. Each step (interaction between scanning a QR code and completing the presented instruction) took on average 1:05 minutes.

Regarding the usability of the system, there was a mildly positive reception. Participants found the system to be moderately simple, consistent, and not too complex. More importantly, they felt more secure in interacting with the system.

To analyze our results, we chose to calculate confidence intervals (CI) for each question, then check where that interval is positioned compared to a neutral response (i.e., a 3 in the Likert scale [28]). Since some of the questions are negative-sounding, a good evaluation in these require the confidence interval to be lower than 3, while the opposite occurs for positive-sounding questions.

Fig. 7 presents the results obtained, with each question categorized. Starting with the effectiveness category, the results are positive. The participants found that the system was not too inconsistent (mean of 1.86 and SD of 0.57) and also that the different functions were decently well integrated (mean of 4.14 and SD of 0.73). However, the participants did have a more neutral and varied response on whether they would need technical assistance (mean of 2.48 and SD of 1.21).

The efficiency/difficulty of the tasks received a very neutral response. Both questions of this category, whether the participant found the system cumbersome and whether they were

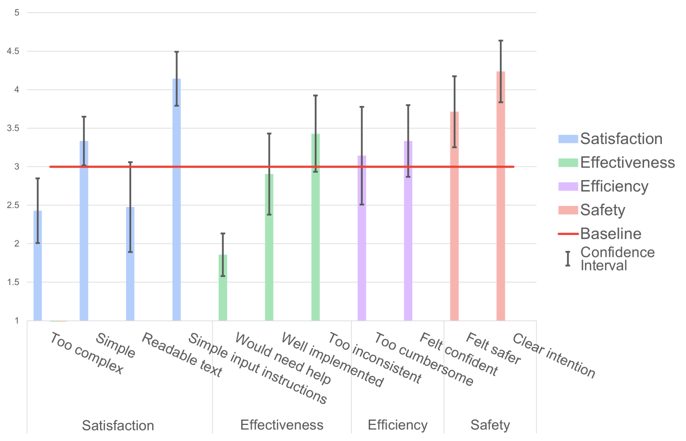


Fig. 7. Average results of user answers to the questionnaire. 1 = Strongly Disagree, 5 = Strongly Agree.

confident in their actions, had responses averaging around 3 (mean of 2.9 and 3.43 respectively), and an SD of approximately 1 for both.

For the satisfaction category, we selected four questions that show a slight positive response with respect to complexity, but a more neutral response regarding legibility and comprehension of the instruction. Participants found the system not too unnecessarily complex (mean of 2.42, SD of 0.87) and found it slightly simple to use (mean of 3.33, SD of 0.66). However, when answering if the text was comfortable to read on the glasses' lenses, there was a wide spread of answers, resulting in a mean of 3.14 and an SD of 1.31. The participants also responded rather differently regarding whether they found it easy to follow the presented instructions (mean of 3.33, SD of 0.97).

Last but not least, regarding the feeling of safety when using the system, there is a good, positive reception to it. When asked if the person felt safer accessing sensitive data using the system, the response was slightly positive, although with a rather high range of answers (mean of 3.71, SD of 0.95). When asked if the system was clear in its intention of securing communications, however, the results are more positive and less spread out (mean 4.24, SD 0.83).

Overall, these results paint a positive picture of the system: 1) It provides a sense of security to the users. 2) Using the system is doable, thanks to its consistency and low complexity. 3) However, there is some difficulty in using the system, even if some practice could mitigate this feeling. 4) Readability can be problematic.

V. Related Work

There is some work on trusted paths [8], [9], [29], and some of the most recent considers TEEs [30], [31]. However, none of these works consider the use of smart glasses or other COST components to support encryption and the trusted path. On the contrary, they assume the use of hardware or virtualized devices for user interaction (e.g., additional displays and keypads). This is actually what inspired this work.

Andrabi et al. [32] studied visual cryptography used in smart glasses to decipher singular numbers and letters on a display. They follow a traditional visual cryptography scheme [33], but with digital shares. A UI device presents one share of a pair, while the user's smart glasses present the other, allowing the user to visually overlap them to reconstruct the original message. Lantz et al. [34] also investigated the use of visual cryptography, specifically to read one-time authentication codes, but proposed a visual obfuscation scheme as an alternative. These works do not provide a trusted path; only confidentiality in service to user communication. There is also no benefit of visual cryptography over TrustGlass' use of symmetric cryptography (AES-GCM) and QR codes.

EyeDecrypt [35] takes advantage of smart glasses to perform cryptographic operations. Data is encrypted in the service, encoded as dataglyphs, then scanned and decrypted by the user's smart glasses. Their encoding scheme allows data to be deciphered in blocks. Something similar is done in Ubic [36], but using QR codes instead. These works are closer to ours, but focus on displaying information to the user, guaranteeing confidentiality in the service to user direction, not on bidirectional communication. We present and analyze the RS and RK schemes to provide security when the user inputs data and its security guarantees. Moreover, we test the usability of a prototype of our scheme with real users, whereas these works are more theoretical and focus on the encryption schemes.

VI. Conclusion

This paper presented TrustGlass, a trusted path scheme that connects a user to a remote trusted application using AR smart glasses on the client side. The smart glasses have the purpose of augmenting the user with cryptographic capabilities, as well as removing the gap between the user's eyes and the display to which the data is transmitted. The TEE secures the remote application, being responsible for encrypting data sent to the smart glasses and displayed to the user, and preparing the instructions on how the user should input their next command to the TEE securely.

We tested TrustGlass' usability via an ATM demonstration application with 21 participants, which highlighted some usability issues, but also showcased that the protocol itself could successfully be applied to protect user-server data exchanges. We also assessed its security using STRIDE, confirming that TrustGlass ensures the properties of a trusted path.

Acknowledgment We warmly thank Alfredo Ferreira for discussions about this research. This work was developed within the scope of the project nr. 51 "BLOCKCHAIN.PT - Agenda Descentralizar Portugal com Blockchain", financed by European Funds, namely "Recovery and Resilience Plan - Component 5: Agendas Mobilizadoras para a Inovação Empresarial", included in the NextGenerationEU funding program. It was also supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UIDB/50021/2020 and 2021.07440.BD and by Universidade de Lisboa (ULisboa), Instituto Superior Técnico (IST), and Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento (INESC-ID).

References

[1] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[2] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071, Feb. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6071

[3] R. W. Shirey, "Internet Security Glossary, Version 2," RFC 4949, Aug. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4949

[4] E. B. Barker and A. L. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," U.S. Department of Commerce, Gaithersburg, MD, Tech. Rep. NIST Special Publication 800-131A Revision 2, 2019.

[5] J. Carmigniani and B. Furht, "Augmented reality: an overview," in *Handbook of augmented reality*. Springer, 2011, pp. 3–46.

[6] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, Jun. 2013.

[7] National Institute of Standards and Technology, "Recommendation for Cryptographic Key Generation," U.S. Department of Commerce, Gaithersburg, MD, Tech. Rep. NIST Special Publication 800-133 Revision 2, 2020.

[8] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune, "Building verifiable trusted path on commodity x86 computers," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 616–630.

[9] Z. Ye, S. Smith, and D. Anthony, "Trusted paths for browsers," *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 153–186, 2005.

[10] T. Tong and D. Evans, "Guardroid: A trusted path for password entry," *Proceedings of Mobile Security Technologies (MoST)*, 2013.

[11] National Computer Security Center, "Trusted computer systems evaluation criteria," Aug. 1983.

[12] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 57–64.

[13] V. Costan and S. Devadas, "Intel SGX explained," *Cryptology ePrint Archive, Paper 2016/086*, 2016.

[14] D. Kaplan, J. Powell, and T. Woller, *AMD Memory Encryption*, AMD, Apr. 2016, release version 2021-10-18.

[15] *ARM Security Technology – Building a Secure System using TrustZone Technology*, Arm Limited, Apr. 2009, issue C.

[16] N. M. Kumar, N. K. Singh, and V. Peddiny, "Wearable smart glass: Features, applications, current progress and challenges," in *2nd International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2018, pp. 577–582.

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

[18] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution," in *IEEE European Symposium on Security and Privacy*, Aug. 2019.

[19] P. Rogaway, "Authenticated-Encryption with Associated-Data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 98–107.

[20] J. A. Salowey, D. McGrew, and A. Choudhury, "AES Galois Counter Mode (GCM) Cipher Suites for TLS," RFC 5288, Aug. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5288

[21] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[22] D. Boneh and I. E. Shparlinski, "On the unpredictability of bits of the elliptic curve Diffie-Hellman scheme," in *Annual International Cryptology Conference*. Springer, 2001, pp. 201–212.

[23] D. H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, May 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5869

[24] D. McGrew, “An interface and algorithms for authenticated encryption,” RFC 5116, Jan. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5116

[25] International Organization for Standards and International Elecrotechnical Comission, “ISO/IEC 18004:2000 - Information Technology - Automatic Identification and Data Capture Techniques - Bar Code Symbology - QR Code,” 2000.

[26] R. Khan, K. McLaughlin, D. Laverty, and S. Sezer, “Stride-based threat modeling for cyber-physical systems,” in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, 2017, pp. 1–6.

[27] J. Brooke, “SUS: A quick and dirty usability scale,” in *Usability Evaluation in Industry*. London, England: CRC Press, 1996, vol. 189, no. 194, pp. 4–7.

[28] R. Likert, “A technique for the measurement of attitudes.” *Archives of Psychology*, vol. 140, pp. 1–55, 1932.

[29] B. Pfitzmann, J. Riordan, C. Stuble, M. Waidner, and A. Weber, “The PERSEUS system architecture,” Technical Report RZ 3335, Tech. Rep., 2001.

[30] H. Liang, M. Li, Y. Chen, L. Jiang, Z. Xie, and T. Yang, “Establishing trusted I/O paths for SGX client systems with Aurora,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1589–1600, 2019.

[31] S. Weiser and M. Werner, “SGXIO: Generic trusted I/O path for Intel SGX,” in *Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 261–268.

[32] S. J. Andrabi, M. K. Reiter, and C. Sturton, “Usability of augmented reality for revealing secret messages to users but not their devices,” in *11th Symposium On Usable Privacy and Security (SOUPS)*, 2015, pp. 89–102.

[33] M. Naor and A. Shamir, “Visual cryptography,” in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1994, pp. 1–12.

[34] P. Lantz, B. Johansson, M. Hell, and B. Smeets, “Visual Cryptography and Obfuscation: A use-case for decrypting and deobfuscating information using Augmented Reality,” in *Financial Cryptography and Data Security: FC 2015 International Workshops, BITCOIN, WAHC, and Wearable*. Springer, 2015, pp. 261–273.

[35] A. G. Forte, J. A. Garay, T. Jim, and Y. Vahlis, “Eyedecrypt: Private interactions in plain sight,” in *Proceedings of the 9th International Conference on Security and Cryptography for Networks*. Springer, 2014, pp. 255–276.

[36] M. Simkin, D. Schröder, A. Bulling, and M. Fritz, “Ubic: Bridging the gap between digital cryptography and the physical world,” in *19th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2014, pp. 56–75.