DELEGATEE: Brokered Delegation Using Trusted Execution Environments

Sinisa Matetic and Moritz Schneider, *ETH Zurich;* Andrew Miller, *UIUC;*
Ari Juels, *Cornell Tech;* Srdjan Capkun, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity18/presentation/matetic

This paper is included in the Proceedings of the 27th USENIX Security Symposium.

August 15–17, 2018 • Baltimore, MD, USA

ISBN 978-1-931971-46-1

Open access to the Proceedings of the 27th USENIX Security Symposium is sponsored by USENIX.

DELEGATEE: Brokered Delegation Using Trusted Execution Environments

Sinisa Matetic
ETH Zurich

Moritz Schneider
ETH Zurich

Andrew Miller
UIUC

Ari Juels
Cornell Tech

Srdjan Capkun
ETH Zurich

Abstract

We introduce a new concept called *brokered delegation*. Brokered delegation allows users to flexibly delegate credentials and rights for a range of service providers to other users and third parties. We explore how brokered delegation can be implemented using novel trusted execution environments (TEEs). We introduce a system called DELEGATEE that enables users (Delegatees) to log into different online services using the credentials of other users (Owners). Credentials in DELEGATEE are never revealed to Delegatees and Owners can restrict access to their accounts using a range of rich, contextually dependent delegation policies.

DELEGATEE fundamentally shifts existing access control models for centralized online services. It does so by using TEEs to permit access delegation *at the user's discretion*. DELEGATEE thus effectively reduces mandatory access control (MAC) in this context to discretionary access control (DAC). The system demonstrates the significant potential for TEEs to create new forms of resource sharing around online services without the direct support from those services.

We present a full implementation of DELEGATEE using Intel SGX and demonstrate its use in four real-world applications: email access (SMTP/IMAP), restricted website access using a HTTPS proxy, e-banking/credit card, and a third-party payment system (PayPal).

1 Introduction

Delegation, the ability to share a portion of one's authority with another, is a well-studied concept in access control. However, delegation remains mostly unsupported in today's online services. Email provides no delegation support at all, for example, while other services, such as Facebook, support delegation in a limited and coarse-grained way. Facebook allows a user to delegate to a third-party application the authority to post to the user's wall, but not to impose a limit of three posts per day. In any case, the expression and enforcement of delegation policies lies entirely at the discretion of the services.

The ability to delegate access to existing online accounts and services, *safely and selectively*, could give rise to new forms of cooperation among users. Delegation may be useful for sharing digital content, such as access to streaming services like Netflix. Users may wish to delegate online tasks to remote workers, for example to reply to emails involving a particular topic or group. Delegation of access to financial services, such as Paypal, could enable broader access to banking.

Today, when delegation is needed in a way unsupported by the service, users must resort to credential sharing. This results in the *Delegatees* gaining full access to the *Owners'* accounts. Such delegation mostly works only in closed circles with high levels of mutual trust.

In this work, we argue that the emergence of trusted execution environments (TEEs), such as Intel Software Guard Extensions (SGX), has enabled an alternative way to achieve fine-grained delegation without trust between the Owner and Delegatee. We refer this new type of delegation — specifically with delegation restricted under a policy enforced by a TEE enclave holding the credential — as *brokered delegation*. Brokered delegation is a new and powerful tool that allows users to flexibly share and delegate access, without requiring the explicit support (or even knowledge) of the service providers.

To demonstrate the potential of brokered delegation, we design DELEGATEE, a system that provides brokered delegation for many existing web services according to complex contextual access-control policies. DELEGATEE also preserves the confidentiality of the managed credentials. We develop several application prototypes to demonstrate how brokered delegation can support new forms of resource sharing and give rise to new markets: secure outsourcing of personal and commercial micro-tasks, tokenization (i.e., creation of fungible, tradeable units), resale of resources and services, and new payment methods - all without changes to the legacy infrastructure. One of the key features of DELEGATEE is that it

requires no changes to the service managing the resource or to users' accounts.

We present two design variations for DELEGATEE. The first design encompasses a purely decentralized peer-to-peer (P2P) system in which a Delegatee that wants to use brokered credentials executes the secure enclave on her machine. The Owner of the credentials connects to the enclave and delivers the credentials along with the access control policy under which the Delegatee can access a specific service. The second design is based on a centralized broker service operated by a third party. In this architecture, an Owner can register credentials and an accompanying policy, authorizing use by a specific population of Delegatees. Both system designs provide a comprehensive solution for brokered delegation and can be used based on users' preferences.

DELEGATEE also demonstrates a broader insight about the security consequences of trusted hardware: TEEs can fundamentally subvert access-control policy enforcement in existing online services. Depending on the application, DELEGATEE can either enrich a target service or undermine its security policies (or both). For example, reselling limited access to a paid subscription service in regions where the service is unavailable undermines the service's security policy, while delegating access to office tools such as mail, calendar, etc. to administrative assistants can enrich the capabilities and usability of the service itself. Brokered delegation can also facilitate violations of web services' terms of use. Users may thereby circumvent *mandatory access control* (MAC) policies, reducing them to *discretionary access control* (DAC). The effect is similar to allowing use of `setuid` [28] in Unix irrespective of MAC policies [39, 36].

The fine-grained delegation offered by DELEGATEE can support new forms of meaningful cooperation among users, which existing online services do not provide. In this way DELEGATEE may be related to new technology-fueled resource-sharing models such as Airbnb and Uber, which have challenged legal and regulatory frameworks while creating and delivering appealing new services. We thus view DELEGATEE as a catalyst for such new contributions to the sharing economy.

In summary, we make the following **contributions**:

- *Brokered delegation:* We advance a new model for user-specified safe delegation of resources and services governed by fine-grained access control. Our approach involves credential outsourcing to trusted hardware.
- DELEGATEE*:* We present DELEGATEE, a system that realizes brokered delegation via Intel SGX. We present two implemented versions: One based on a hardened third party acting as a credential broker and the other as a peer-to-peer system where users directly store, manage, delegate, and use credentials.
- *Security analysis:* We show that both DELEGATEE versions provide security in a strong adversarial model, protecting against some compromised SGX platforms as well as the full software stack of victims' machines.
- *Prototype implementations:* We describe and implement four applications on top of DELEGATEE: Delegated email, PayPal, credit card/e-banking, and full website access through an HTTPS proxy. We run these with commercial services such as Gmail and PayPal using real user credentials. We document minimal performance overhead and the ability to support many concurrent users.
- *Impact on access control:* We show that TEEs can be used to circumvent MAC policies in online services and allow discretionary access control, enabling users to delegate rights and access at their discretion.

2 Motivation and Problem Statement

2.1 Motivation

There are two major motivations for our work: To demonstrate the many settings in which brokered delegation gives rise to new functionality, and to demonstrate how (for good or bad) trusted hardware TEEs can transform practically any mandatory access control policy in an online service into a discretionary one. Our four different application scenarios illustrate both motivations.

Mail/Office. Full or restricted delegation of a personal mailbox or other office tasks can be appealing for many reasons. These include a desire to delegate work to administrative assistants (e.g., read-only access, send mail only to a specific domain) or to allow limited access to law-enforcement authorities (e.g., read emails from a certain time window relevant to a court case). The first is especially valuable for virtual-assistant services, which outsource office tasks off-site [26]. Today, these services require users to completely share their credentials, a dangerous practice that discourages many potential users.

Payments. Virtually all payments, cash and cryptocurrencies excepted, happen through intermediaries. Users may naturally desire a richer array of choices of these intermediaries. Consider, for example, a payment system where the users pay using each others' bank accounts, credit cards, or third-party providers (e.g., PayPal). This can have large benefits in terms of cost-saving, business operations, and anonymity guarantees.

Imagine that a company wants to allow its employees to execute online purchases with the company credit card or PayPal, but restricted to a certain limit per expenditure and specific merchants. Currently, this cannot be done since access to the card details or PayPal credentials allows users to execute arbitrary payments. Companies therefore typically provide such information only to a few employees who then execute payments for the rest,

resulting in a highly inefficient process.

Delegation of payment credentials can also enable direct cost-savings for the end user. An example online system based on this premise is Sofort [25]. Sofort works as an internet payment middleman, with lower transaction fees than for credit cards. Sofort pays merchants for clients' online purchases and is repaid by clients via bank transfer. To guarantee repayment, Sofort requires users to share their e-banking credentials with the service, a practice that clearly raises security and privacy risks.

Finally, delegation of payments can benefit "underbanked" populations with limited access to online payment systems, by enabling them to leverage social ties (e.g., via brokered delegation to the bank accounts of friends, family, and peers).

Full Website Access. The most versatile form of delegation is delegation for arbitrary existing web services, which typically authenticate user accounts through password challenges and then cookies over HTTPS. This model includes access to users' social networking sites, video services, online media such as news and music, and general website content available only to registered users. One appealing example from the academic world is *Sci-Hub*. "The site bypasses publishers' paywalls using a collection of credentials (user IDs and passwords) belonging to educational institutions which have purchased access to the journals." Many anonymous academics from around the world donate their credentials voluntarily [9]. Some services, such as Netflix and various news sites, already offer users the ability to log in from different devices. Users can thus share their subscriptions by sharing credentials, but only in a dangerous all-or-nothing manner. More fine-grained, e.g., service-specific, and secure delegation could facilitate much broader sharing (for good and bad).

Sharing Economy. The examples above involve an Owner delegating credentials to known Delegatees, e.g., friends or colleagues. However, Owners can also offer access to their services on an open market to a wide range of potentially pseudonymous or anonymous Delegatees. This would result in a shared economy in which Owners sell time-limited and restricted access to their accounts in return for other services or financial compensation. For example, users could sell access to Netflix accounts on an open market. They could also sell space in their social networking accounts to advertisers; e.g., a user could sell the ability to post in her name, enabling an advertiser to target her social network. The right to post could be restricted to a certain volume and type of content to prevent abuse by advertisers.

2.2 Problem Statement

If service providers regularly offered richly featured native delegation options, there would be no need for **brokered delegation**. Most do not, however, usually for business or regulatory reasons. Our work aims to change this situation fundamentally — DELEGATEE empowers users to delegate their authority, making use of any existing internet service, such that:

- The Owner's credentials remain confidential.
- The Owner can restrict access to her account, e.g., in terms of time, duration of access, no. of reads/writes etc.
- The system logs the actions of Owners and Delegatees so that post-hoc attribution of their behaviors is possible (as a means of resolving disputes).
- The system minimizes the ability of a service to distinguish between access by the Delegatee and that of the legitimate Owner, thus, preventing delegation. (As we shall discuss, this is not achievable for all services.)

2.3 Why the Problem is Hard

DELEGATEE leverages SGX to implement functionality that without SGX or equivalent mechanisms would be infeasible or impossible to achieve. Consider our delegated payment scenarios involving PayPal, credit card or e-banking. Such delegation would be easy to support on the back end; e.g., PayPal could offer a delegation API.

Without back end support, however, there are only two possible implementation strategies. The first is that the Owner remains online and mediates requests, which forecloses on the possibility of private transactions or her inability to provide continuous service availability.

The second is that the Owner provides the Delegatee with a digital resource for unmediated access to the target resource. This, however, would require black-box obfuscation to construct a functionality that establishes a TLS connection, authenticates a user with a concealed password, and supports a series of policy-constrained transactions. General virtual black-box (VBB) obfuscation is known to be impossible [5]. It is unclear whether indistinguishability obfuscation ($i\mathcal{O}$), whose realization remains an open problem [12], could achieve this functionality. $i\mathcal{O}$, would in any case require circuit complexity well beyond the bounds of feasible deployment. It would also be subject to replay attacks unless the functionality could somehow change or revoke the credential atomically with permissible operations. In summary, SGX is required to solve our problem as stated, and even with SGX, as we now explain, solution remains challenging.

3 DELEGATEE

The main idea behind the DELEGATEE system is to send the Owner's credentials (passwords, etc.) to a Trusted Execution Environment (TEE) that implements the delegation policy. The Delegatee communicates with the resource (web service) indirectly, using the TEE as a proxy. In this section, we briefly introduce background on TEEs, then present the DELEGATEE system design.

3.1 TEEs and Intel SGX Background

Modern TEE environments, most notably ARM TrustZone [3, 42] and Intel SGX [13, 1], enable isolated code execution within a user's system. Intel introduced SGX in the 6th generation of its CPUs as an instruction set architecture extension. Like TrustZone, an older TEE that permits execution of code in a "secure world" and is used widely in mobile devices, SGX permits isolated execution of the code in what is referred to as secure *enclaves*. In TrustZone, transition to the secure world involves a complete context switch. In contrast, the SGX's secure enclaves only have user-level privileges, with `ocall`/`ecall` interfaces [20] used to switch control between the enclaves and the OS. The SGX architecture enables the app developer to create multiple enclaves for security-critical code, protecting it from malicious applications [43], a compromised OS, virtual machine manager [11], or BIOS [24], and even insecure hardware [16] on the same system. Additionally, SGX includes a key feature unavailable in TrustZone, called *attestation*.

In summary, the main protective mechanisms supported by SGX are: runtime isolation [33], `ocall`/`ecall` interfaces [20], sealing [2], and attestation [22, 13]. We relegate further details below; for in-depth treatment of SGX, see [13, 21].

Readers familiar with Intel SGX can skip the rest of this subsection. The main protection mechanisms of SGX, in more detail, are:

Attestation. Attestation is the process of verifying that enclave code has been properly initialized. We distinguish between two types:

- In *local attestation*, a prover enclave requests a statement containing measurements of its initialization sequence, enclave code, and issuer key. Another enclave on the same platform can verify this statement using a shared key created by the processor.
- In *remote attestation* the verifier may reside on another platform. A system service called Quoting Enclave signs the local attestation statement for remote verification. The verifier checks the signature with the help of an online attestation service run by Intel. The signing key used by the Quoting Enclave is based on a group signature scheme called EPID (Enhanced Privacy ID) which supports two modes of attestation: fully anonymous and linkable attestation using pseudonyms [22, 13].

Runtime isolation. As mentioned, the SGX security architecture guarantees enclave *isolation*, using protective mechanisms enforced in the processor, from all software running outside of the enclave. The control-flow integrity of the enclave is preserved and the state is not observable. The code and data of an enclave are stored in a protected memory area called Enclave Page Cache (EPC) that resides in Processor Reserved Memory (PRM) [33].

Sealing and Memory encryption. Enclaves can save confidential data across executions trough sealing, a process for encrypting and authenticating enclave data for persistent storage [2] controlled by the untrusted OS. Each enclave is provided with a sealing key, private to the executing platform and the enclave. The sealing key is derived from a Fuse Key (unique to the platform, not known to Intel) and an Identity Key (either Enclave Identity or Signing Identity). Additionally, all runtime enclave memory is encrypted and cannot be accessed by the OS as described above. In Section 4 we consider an attacker that cannot break the SGX hardware protection mechanism but can have all SGX keys used to, e.g., decrypt seals or the extracted memory content.

Ocall/Ecall. The interface between the trusted enclave and the untrusted application is implemented using `ocalls` and `ecalls`, calls from the trusted to the untrusted part, and vice-versa, respectively. During an `ocall`/`ecall` all arguments are copied to trusted/untrusted memory and then executed in order to maintain a clear partition of trusted and untrusted parts. These interfaces are defined and implemented by Intel in the Intel SGX SDK [20].

3.2 System Design

We explore the DELEGATEE design space through two system architectures: a purely decentralized *P2P system*, and what we call a *Centrally Brokered* system, in which a third party runs the enclaves. Both architectures involve three distinct classes of parties: credential *Owner(s)* A, *Delegatee(s)* B, and *service(s)* G. Additionally, the system distinguishes 2 data types: *credential(s)* C and *access control policy(ies)* P. Owners and Delegatees are generically referred to as *users*.

The system supports a potentially large population of credential Owners $A_1 ... A_n$ (henceforth referred to as Owners) and Delegatees $B_1 ... B_n$. In general, the Owner A_i has access to a service G_k. The Delegatee B_j does not have access to the service, but she can get access by using credentials C_x of the Owner A_i. However, the Owner A_i does not want to reveal the credentials to the Delegatee B_j. The Owner A_i wants her credentials to remain confidential and used only by an authorized Delegatee. Additionally, the Owner wants to restrict access to the services that she enjoys (i.e. G_k) according to an access control policy P_{ijxk} specific to this delegation relationship. P_{ijxk} defines an policy involving Owner $A_{\mathbf{i}}$, Delegatee $B_{\mathbf{j}}$, credentials $C_{\mathbf{x}}$, and service $G_{\mathbf{k}}$. The type and structure of the access control policy depends on the service that the Owner delegates. Definition and enforcement of the policies are described in Section 3.4.

P2P system architecture. In our peer-to-peer system, there is no need for a central management entity to mediate between the Owners and the Delegatees. A Delegatee

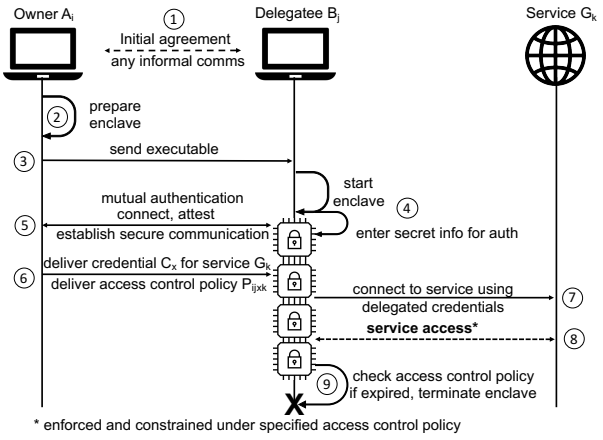


Figure 1: DELEGATEE's P2P system architecture

can directly coordinate with the Owner to gain access to a specific service from group G. In order to execute this setup, a Delegatee from party B has to have a Intel SGX supported machine. The steps to execute secure credential delegation, also given in Figure 1, are:

(**1**) The Owner A_i agrees directly with the Delegatee B_j for which specific service (G_k) access will be granted using her credentials (C_x). The agreement is done at the users discretion and through any available channel such as online messaging, email, phone call etc. Additionally, users need to establish a method for authentication upon enclave start (e.g. pre-shared key, certificates). This step can be executed in an any informal communication channel that the users consider appropriate. However, the emphasis should be on the confidentiality of the channel (e.g., chat over a coffee).

(**2**) (optional[1]) After that, A_i prepares the enclave.

(**3**) (optional[1]) Owner A_i sends the executable to B_j.

(**4**) The Delegatee B_j starts the enclave and enters the secret information (shared secret exchanged during the initial agreement) to the enclave needed for mutual authentication and secure connection establishment.

(**5**) After the Delegatee B_j starts the enclave, the Owner A_i connects to the enclave, attests it to verify that it is the correct code with respect to the requested service delegation, and subsequently uses the secret information to authenticate and create a secure communication channel.

(**6**) The A_i sends credentials C_x for the service G_k with the access control policy P_{ijxk} using the secure channel.

(**7**) The Delegatee B_j now uses the enclave as a proxy to connect to the service G_k using the delegated credentials.

(**8**) The scope of usage is strictly limited by the defined policy and therefore Delegatee B_j cannot use the parts of the service not allowed by the Owner A_i.

(**9**) If the access control policy has a time limit, the Delegatee B_j's access to the service is terminated after the time has passed, unless the Owner A_i extends the policy. The enclave restarts do not change this fact, requiring the connection from the Owner A_i to the enclave to deliver the information again. The enclave is stateless, meaning that any interruption, restart or termination after the initial start and the delivery of confidential information is going to result in service abortion.

Authentication mechanisms. The agreement between the users and their mutual identification and authentication is of utmost importance. The Owner needs to be certain that the enclave used to access a specific service with her credentials is running on the machine of the intended Delegatee. Attestation only gives us proof that the enclave is executing the presumed code, but without any information under whose control the machine is. To allow mutual authentication between the Owner and the Delegatee, a separate authentication method is needed.

Several authentication mechanisms are possible. First, the parties could use an out-of-band confidential and authenticated channel to exchange a shared secret key. After the enclave start, the Delegatee enters this pre-shared key into the enclave. The Owner uses the same key to establish a TLS (PSK mode) session with the enclave. If an attacker attempts to establish an impostor or man-in-the-middle session with the Owner, the keys will mismatch. As an alternative, we could use a trusted PKI so that the Owner obtains Delegatee's public key certificate, later used to establish a TLS session. This requires the Delegatee to provide her private and public keys to the enclave. Our design is agnostic to the used authentication method while the prototype uses the first option.

Centrally Brokered system architecture. Alternatively to the P2P configuration, the Centrally Brokered system consists of a central server that mediates all transactions and communication between the involved parties and also serves as a management entity. The server has a trusted execution environment (SGX enclaves) that performs security-critical operations. Thus, the system can be attested to verify the running code and authenticated to verify the service provider. In this case, the Owners and the Delegatees do not need to have SGX. Steps needed to execute secure delegation follow Figure 2:

(**1**) Both the Owners (A_1...A_n) and the Delegatees (B_1...B_n) need to register with the system to acquire unique login information (username and password) for access. After registration, both Owners and Delegatees can execute credential delegation for service access.

(**2**) The Owners A_1...A_n now establish a secure channel to the system (using the ordinary web PKI) and start stor-

[1]Enclaves used for the credential delegation can also be downloaded from a trusted source. Each different service requires implementation of specific enclaves due to access complexity. The Owner and the Delegatee can verify the enclave trustworthiness with attestation.

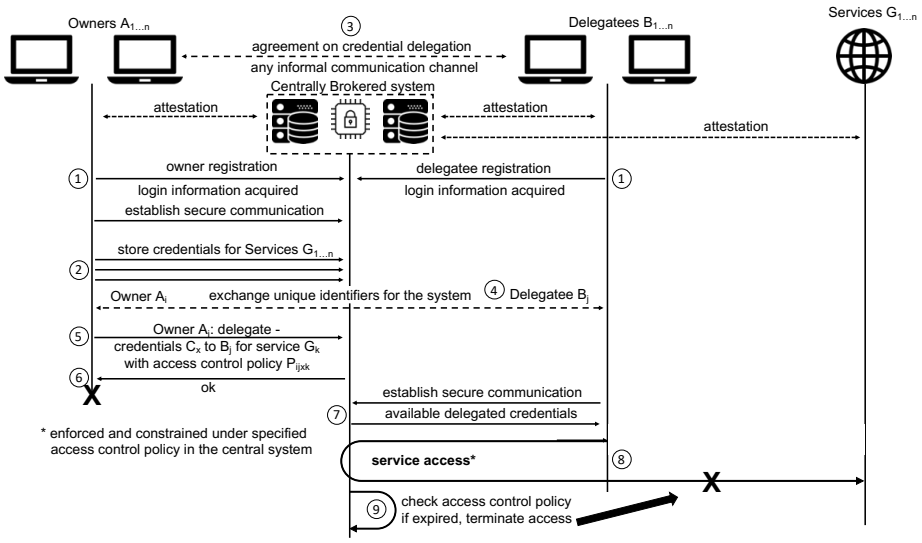


Figure 2: Centrally Brokered system architecture for credential delegation with DELEGATEE

ing the credentials $C_1...C_n$ for specific services $G_1...G_n$. The variety of credentials that can be stored depends on the supported services (see Section 5 for details).

(**3**) The Owners $A_1...A_n$ may agree directly with the Delegatees $B_1...B_n$ for which specific service (G_k) the Owner will grant access using her credentials (C_x). The agreement is done at the users discretion through any available out-of-band channel and is limited by the implemented technical capabilities of the system (i.e., for supported use cases implemented by DELEGATEE).

(**4**) During the agreement, users exchange their unique identifiers (i.e. system username) so that the Owner from party A knows whom to authorize from party B.

(**5**) The Owner A_i establishes a secure channel to the system, specifies for which credentials (C_x) she wants to perform the delegation, for which service (G_k) and to whom (username of B_j), while she additionally specifies the access control policy P_{ijxk} that restricts usage.

(**6**) After receiving the confirmation, A_i disconnects.

(**7**) The Delegatee B_j now establishes a secure channel to the system and can immediately see that she has been delegated credentials for a certain service. The credentials are hidden for the Delegatee B_j. If the Delegatee wants to access the service G_k, she may proceed.

(**8**) The access to the service is always proxied through the central broker with no direct communication between the Delegatee and the service. Any attempt to circumvent this results in protocol termination (e.g., if the user clicks an external link outside the proxied service).

(**9**) After the defined access control policy expires (e.g. if it is time limited) the Delegatee B_j loses access and the credentials are no longer delegated.

Interoperability. In Section 5 we describe the implementation of DELEGATEE. Our prototype implementation is based on the Centrally Brokered architecture, since this is the most plausible deployment scenario, although we discuss how the P2P model applies to each supported application. The implemented enclaves have two operation modes that can be chosen and set prior to the execution. In case of the Centrally Brokered system, the enclave retrieves important data regarding services, credentials, and access control from the management enclave, while in the P2P system, the enclave awaits the connection from its issuer to receive all information.

3.3 Usage with and without anonymity

DELEGATEE supports both identity-based (non-anonymous) and anonymous use models, as follows.

Identity-based model. An identity-based model follows directly from the model and examples given above. Here, the users know each other in some way, have a communication channel and can mutually identify each other. The Owner directly delegates her credentials to a specific Delegatee. Common use case examples include family sharing, delegation among friends and colleagues, etc.

Anonymous model. As DELEGATEE conceals an Owner's credentials, it naturally preserves her anonymity, even in the P2P model where the Delegatee operates the enclave executing DELEGATEE. However, the agreement is necessary in order to specify details for the delegation relationship. An Owner and Delegatee may negotiate and perform credential delegation without direct interaction. For example, a bulletin board (available on the Centrally Brokered system) might allow Owners to publicly list services they are willing to delegate, specifying accompanying access control policies and costs (or offer of free service). Owners may identify themselves with pseudonyms, e.g., onion addresses.

In the P2P model, the bulletin board can be hosted on a third-party website, while the protocol runs through Tor Hidden Services, thereby ensuring privacy protection for both the Owner and Delegatee.

3.4 Policy Creation and Enforcement

Securely enforcing defined policies presents a challenge on its own. We aim to prevent all attackers from modifying the policies or circumventing the enforcement by applying a combination of allowed actions in order to reach a desirable state. While the security analysis (Section 4) ensures that the owner-provided access control policy is respected, the burden remains on the Owner to choose an appropriate access control policy in the first place. An Owner who wants to delegate restricted access for a specific service needs to be able to define all allowed actions through a rich access control policy, denoted as P_{ijxk}. For increased security, we prefer the white-listing of operations based on the least-privileges in order to prevent unwanted access and usage of the delegated account. Unfortunately, a general model for a wide variety of different services cannot be used. For every specific service category, and sometimes even for every specific service provider in the same category, a new policy must be created that resembles the exact capabilities and actions which a fully allowed user may invoke. We discuss the limitation of policies in Section 7.

Policies in DELEGATEE. We designed and implemented policies for all scenarios defined in Section 2.1, namely, for mail, payments, and full website access.

In mail, DELEGATEE relies on the IMAP and SMTP protocols which are standardized and well defined. Inside the enclave we parse all incoming and outgoing request (to and from the Delegatee) and compare them against the defined access policy. Consider a concrete scenario: the organizer of a conference wishes to delegate her email account to an assistant to respond to logistical questions from attendees. The Delegatee should be granted read access to only *subset* of the organizer's email (e.g., defined by a regular expression query like `(*#Usenix18*)`). The organizer might also wish to enforce restrictions on message sending. Rather than sending to any possible email address, the assistant may only be allowed to *reply* to emails and deleting emails should be prevented. In general, for the inbox requests the Delegatee can be limited based on criteria such as date, time, sender, subject or content of the email. In outgoing requests, the limitation is set on the subject or content, and the intended recipient(s). Additionally, the Owner can rate-limit emails sent within a time interval, applying a spam and abuse filter for outgoing messages.

In payments, the main restriction is on limiting the allowed amount per transaction or the total amount using the delegated credential for either a credit card or any other third party payment service. Additionally, the DELEGATEE can enforce restrictions on the source, limiting the Delegatee to perform payments only on specific sites or identified merchants/services, and white-listed geographical locations based on the IP address.

In the full website access, DELEGATEE implements limiting the use of login credentials to specific sites (e.g., the Owner can have the same credentials for two different services. However, full access is only achieved to the site allowed by the policy). As work in progress, the policies are expanded to restrict specific actions on sites after the login, including, clicks on various links, loading of specific site content or access to the account settings.

Our prototype implements delegation policies targeted at particular services, directly in `C++`. These policies rely on the mechanisms explained above; the Owner only needs to configure the value of the policy attributes (e.g., time limit, max amount, regular expression, etc.). In principle, the credential Owners could describe their own delegation policy in a general programming language. In Section 8 we mention existing and generic ways to extend our general functionality regarding access control. However, specifying the policies is difficult to do correctly. We envision that a likely deployment scenario is a curated "app store", to which entrepreneurs or power users submit useful policies they develop. These policies are then evaluated by experts and users. Web services are constantly updated, and the interfaces change over time, requiring delegation scenarios to be continuously maintained as well. In Section 7 we discuss further challenges if the services seek to actively prevent delegation.

4 Security Analysis

Brokered delegation provides a new usage pattern for potentially any existing online service. It, therefore, provides new security challenges as well, arising especially because each new service requires a customized delegation mechanism. In this section we describe the main security properties that DELEGATEE is designed to ensure across all applications:

(**a**) *First and foremost, the Owner's access credentials remain confidential.*

(**b**) *The use of the delegated credentials is defined by the access control policy which will not be violated.*

(**c**) *Use of the credentials should only be granted to the intended Delegatee, as authorized by the Owner.*

The DELEGATEE system is designed to provide these security guarantees even against a strong attacker model. We assume that an attacker neither corrupts the full software stack of the Owner's and Delegatee's machines (unless the Delegatee is the attacker), nor the online service, as existing web authentication mechanisms rely on them anyway. However, we consider an attacker that controls

everything else (i.e., including the standard Dolev-Yao adversary [14] that can read and manipulate network traffic between parties). The two architectures we develop, P2P and Centrally Brokered, differ mainly in where the enclave is hosted (respectively, on the Delegatee's own device or at an independent third-party). Although we rely on a TEE, our system is designed to tolerate vulnerabilities in the SGX enclaves as long as the software stack on the machine running the enclave is also not compromised. Below we discuss several attacker configurations and the design decisions made to mitigate them. It is of utmost importance to note that our system is designed in a way that breaking the SGX protection mechanism on an arbitrary enclave will not subvert our system. The attacker would need to break the exact enclave running DELEGATEE, bypass the authentication mechanism, and compromise the full software stack on the same machine to violate the security properties. Additionally, we consider side-channel attacks to be out of scope of this work.

4.1 Security through trusted enclaves

We first describe how these properties are ensured assuming the TEE enclaves are secure, even if the software stack of the Centrally Brokered system is compromised.

In the Centrally Brokered architecture, the TEE guarantees security properties (a) and (b) even if the central broker and the Delegatee are otherwise corrupted. The Owner only transmits her credential after validating the attestation that the enclave is running the correct code and if the authentication is successful. The mechanism for authenticating the Delegatee to the broker also lies inside the enclave, in the broker's API enclave. This means that property (c) is guaranteed even if the broker's full software stack is compromised since all security-critical operations are performed inside the enclave.

In the P2P architecture, even if the Delegatee's software stack is corrupted, the Owner's credentials are kept confidential. In Step (5), the Owner receives a TEE attestation before communicating further over the TLS channel, and validates it against the DELEGATEE enclave executable. Since the Owner only sends her credentials along this channel directly to the enclave, it is never exposed to the Delegatee's host machine, thereby ensuring property (a). The only way the Delegatee can make use of the credentials is by providing commands as input to the enclave (all access is proxied through the enclave), where they are processed according to the access control policy P_{ijxk}, ensuring the enforcement of (b). Since the TEE is hosted locally by the Delegatee (that also has to authenticate to the Owner using the agreed shared secret), then property (c) is ensured against an external attacker if he cannot steal the shared secret; against a rogue Delegatee, this property is not meaningful anyway.

We note that in either architecture, the code running in the enclave must use the credentials in application-specific ways. We stress that in our proposed system, the owner-provided access control policy P_{ijxk} for service G_k is a configuration parameter given as input to one of the supported application specific enclaves. Hence the proof burden is on us to show that properties (b) and (c) hold for any policy P_{ijxk}. We refer the reader to Section 3.4 and Section 5 for a detailed explanation. To summarize, each application makes use of the credentials only to authenticate with the corresponding service.

Finally, we note that Denial-of-Service attacks is outside the scope of our security guarantees since an external (network) adversary can always drop messages.

4.2 Robustness to compromised enclaves

Our system relies on the TEE to provide security against a compromised Delegatee or the broker service. However, DELEGATEE is also designed to provide defense in depth where possible, such that even a partial compromise of the TEE does not impact security (as long as the host machine is also not compromised). In particular, we consider an attacker that can recover the internal keys (e.g. sealing, memory encryption, etc.) of the Intel SGX. This strong attacker would be able to decrypt any sealed persistent storage or encrypted memory pages and create false attestations. As of the time of writing, there have not been any such attacks on Intel SGX keys. Regardless, by arguing security against this attacker model we reduce the harm if such a vulnerability should be found.

We address these concerns by designing our protocol so that all communication channels are authenticated end-to-end, even when communicating with an attested SGX enclave. To illustrate, first consider the P2P architecture. Our authentication mechanism defends against such an attacker by requiring authentication input from the Delegatee before establishing the TLS endpoint in the enclave. Notice that by Step (5), the Owner A_i opens the TLS endpoint to the Delegatee's enclave, over a potentially insecure channel. At this point, if the attacker can forge an enclave attestation, then the TLS channel may actually be an impostor channel. However, by authenticating the TLS channel against the pre-shared secret initially established with the Delegatee, the Owner would detect and invalidate such an impostor channel. This authentication occurs before the Owner ever transmits the credentials, ensuring desired property (a). Furthermore, the enclave software is guaranteed to be the correct DELEGATEE executable transmitted by the Owner, as long as the Delegatee's host OS is uncorrupted. This ensures that properties (b) and (c) hold as well. However if a rogue Delegatee colludes with an attacker that can forge TEE attestations, or if the Delegatee's software stack is fully compromised, then the credentials would be forfeit.

Our Centrally Brokered architecture is also designed with end-to-end authentication to mitigate against a potentially compromised TEE. The Delegatee and the Owner each establish authenticated TLS channels to the central broker (authenticating the broker's enclaves using the typical certificate PKI), and only communicate to the broker over this channel. Hence all three security properties are ensured as long as the service's own software stack is not accessible to the attacker, regardless of any forged TEE attestations the attacker may produce or if the attacker can guess the SGX keys used by the enclave.

We also avoid the use of persistent encrypted storage in the P2P model, thus, preventing potential rollback attacks [32], which may otherwise occur if the enclave's sealing keys can be derived by the attacker. Our DELEGATEE enclaves, therefore, do not provide any means to resume a previously-established delegation session if the processor is power cycled. Instead, their state is restarted from scratch. In the Centrally Brokered system, we do presume that the attacker has no presence on the full software stack, thus, for continuous operation of the system we make use of the persistent encrypted storage. If the attacker model would be expanded to allow attacker presence on the software stack, methods and techniques described in [32] could be applied to prevent rollback.

For ease of exposition, we have only discussed the highlights of our security design. A systematic security analysis can be found in the online (eprint) version of our paper at `https://eprint.iacr.org/2018/160`.

4.3 Other Security Properties

Mandatory Logging. A well-chosen policy should ideally prevent any misuse from occurring. To be prudent, we would also like to ensure support for forensic investigation in the case that an incorrect policy is abused. We propose that all the requests and responses exchanged between the service provider and the Delegatee are securely logged using a timestamped statement signed by the enclave, for a possible later review. For example, in the payment scenario, if the Delegatee uses the Owner's credit card, the following events are registered: time and date, the website and the amount of the executed payment. As another example, in the mail scenario, if the Delegatee manages to evade the abuse filter and send offensive emails, these messages should be logged. We imagine such logs may be used later on to prove that the Delegatee herself performed some action and indemnify the credential Owner. This discourages the Delegatees to perform any actions that could harm the Owner. To detect suppression of log entries, we could make use of a hardware monotonic counter. The enclave could additionally require a "receipt" from an independent backup service that replicates the log entry, before continuing.

Delegatee protection. So far our security analysis has only focused on protecting the Owner. Security for the Delegatees may be important too. For example, if a Delegatee wishes to use the Owner's payment account to purchase a sensitive item, they may not wish for the transaction details to be disclosed to the Owner. A delegation policy supporting the Delegatee could, in this case, offer a way to automatically delete payment transaction logs (if this is possible at all using the payment service).

Enclave type	Core	`mbedtls`	Total
API	4.0 (7.3%)	51.0 (92.7%)	55.0
Mail	1.9 (3.6%)	51.0 (96.4%)	52.9
Paypal	2.6 (4.9%)	51.0 (95.1%)	53.6
CreditCard	2.5 (4.7%)	51.0 (95.3%)	53.5
HTTPS Proxy	2.7 (5.0%)	51.0 (95.0%)	53.7

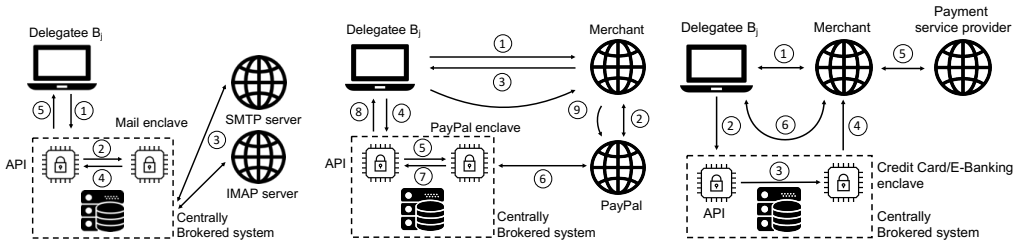
Table 1: TCB of DELEGATEE in LoC (thousands).

5 Prototype Implementation

In this section we describe our prototype implementation for the selected use cases mentioned troughout the paper. All enclaves rely on the OS to handle incoming and outgoing TCP connections while the SSL endpoints reside in the trusted enclaves. We use the `mbedtls` library developed by ARM [29], which also comprises the bulk of our trusted-computing-base (TCB). The interface between the OS and the enclaves consists of one `ecall` and ten `ocalls`, all of which are needed by the SSL library to use the OS's capability to handle the TCP connections. The small number of calls and the small TCB, as shown in Table 1, facilitate code verification and reduce the surface area that may be affected by vulnerabilities.

To demonstrate our use cases, we implemented four service specific enclaves for delegated use of mail, PayPal, credit card/e-banking, and full website access through an HTTPS proxy. Additionally, a fifth management enclave is used to authenticate the users and store credentials, implemented as a RESTful API, further referred as the API. The API enclave is not used in the P2P system since it is not needed. Only service specific enclaves are deployed on the Delegatee's machine. Additionally, we implemented a browser extension that communicates directly with the Centrally Brokered system and allows ease-of-use for the delegated credentials by the Delegatee (page parsing, detection of forms, choosing delegated credentials, etc.). All communication between the users, the enclaves and the browser extension is done using TLS with replay protection. We refer the reader to Appendix A for prototype screenshots of chosen examples. In these implementation details we presume that the Owner A_i and Delegatee B_j already registered to the system and that the Owner authorized the Delegatee by storing the credentials C_x and defining the access policy P_{ijxk} for a specific service. Thus, the Owner A_i is not shown in the figures.

Multithreading in Intel SGX. Intel SGX does not sup-



(a) Mail model (b) PayPal transaction model (c) Credit card / e-banking model

Figure 3: Architecture overview for the Centrally Brokered system

port traditional multithreading within an enclave. Additional threads cannot be started by an enclave, instead multiple threads of the untrusted app can simultaneously perform an `ecall`, resulting in parallel enclave execution. The amount of concurrency is specified during compilation of the enclave and is limited by the number of logical cores in the processor.

Additional Authentication. In Section 7, we discuss limitations concerning the modern authentication challenges and DELEGATEE. Our implementation supports one advanced authentication method involving *CAPTCHA*. In case of website login or PayPal, a captcha may be required as an additional authentication step. We successfully overcome this issue by extracting the secret image, presenting it to the Delegatee through browser extension generated pop-up, allowing her to solve it and continue with executing the desired operation. We refer the reader to Appendix A for prototype screenshots.

5.1 Mail/Office

Delegation of email accounts under a specific access policy, one of the DELEGATEE motivated applications, is implemented in the mail enclave. IMAP and SMTP clients are implemented to allow a Delegatee B_j to read and send emails using the delegated credentials C_x. Below we describe the architecture depicted in Figure 3a:

(**1**) The Delegatee B_j wants to use some credentials C_x that have been delegated by A_i. B_j connects securely to the centralized API using her username and password (for P2P model the communication is established as described in Section 3.2, with both methods supported). She then requests to perform some action using C_x.

(**2**) The API verifies that the Delegatee has access to C_x and then forwards the request, C_x and the corresponding policy P_{ijxk} to the mail enclave.

(**3**) The mail enclave connects to either the SMTP server (for sending mail) or the IMAP server (for receiving mail) and executes the requested operation.

(**4**) P_{ijxk} gets applied to the response from the external servers (IMAP) or to the outgoing requests (SMTP) and the resulting response gets forwarded to the API.

(**5**) The API delivers the final response to B_j.

5.2 Payments

PayPal. PayPal does not want to endorse giving away your credentials or automating the payments as this could compromise their security. Thus it is non-trivial to automate a PayPal payment and there is no public API. We must emulate a browser inside our enclave that accurately simulates a real user. Normally the payment process relies on a javascript library but running a javascript interpreter in Intel SGX would bloat the TCB, and create potential vulnerabilities associated with running an unmeasured, externally provided script inside an enclave. We instead use the no javascript fallback mechanism from PayPal. Our implemented emulated browser follows redirects, fills known forms, and handles cookies until the final confirmation page is reached. The enclave then returns a confirmation `id` to the issuer that is used by the merchant to finalize the payment. Our implementation was tested using PayPal's sandbox and real-world environment, executing a real payment. Our browser extension simplifies the use of delegated PayPal credentials by adding a DELEGATEE checkout button next to the original PayPal checkout button if the Delegatee is logged in to our system and has some delegated credentials. Upon clicking on the DELEGATEE checkout the Delegatee can choose one of the available PayPal credentials delegated to her and then the automated payment process starts (please see Appendix A for screenshots). After that, no further user interaction is needed and the Delegatee will be forwarded to the confirmation page of the merchant if the payment succeeds. Below we describe the architecture depicted in Figure 3b:

(**1**) The Delegatee B_j wants to buy something from a merchant using credentials C_x delegated by A_i. B_j connects to the merchant and asks for a PayPal payment.

(**2**) The merchant uses PayPal API to create a payment.

(**3**) The payment is then forwarded to B_j.

(**4**) B_j connects securely to the centralized API enclave using her username and password (for P2P model the communication methods are described in Section 3.2).

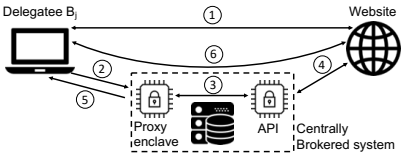


Figure 4: Login model

She then requests to pay with PayPal using C_x.

(**5**) The API enclave verifies that the user can access to C_x and then forwards the request, C_x and the corresponding policy P_{ijxk} to the PayPal enclave.

(**6**) The PayPal enclave connects to PayPal and pays the payment with C_x if it is allowed by the policy P_{ijxk}. The PayPal service responds with a confirmation number.

(**7**) The confirmation number is forwarded to the API.

(**8**) The API delivers the confirmation number to B_j.

(**9**) B_j forwards the confirmation number to the merchant and then the PayPal payment is finalized by the PayPal API using the received confirmation number.

Credit card/e-banking. Payments are similar to PayPal payments: upon checkout on the merchant's website, the browser extension is triggered if the payment form is available. The Delegatee chooses any delegated credentials she is authorized to use. The enclave fills the form with the credentials received either from the centralized API or directly from A_i using the P2P model. Our implementation was tested without any service provider that would finalize the transaction. Figure 3c shows the detailed architecture and the steps follow bellow:

(**1**) The Delegatee B_j wants to buy something from a merchant using some credentials C_x containing credit card or e-banking information that have been delegated by A_i. B_j connects to the website and the browser extension renders a second button beside the normal credit card and e-banking credentials submit button.

(**2**) Upon clicking the injected button, the browser extension requests a payment with C_x from the API.

(**3**) The API verifies that the user has access to C_x and then forwards the request, C_x and the corresponding policy P_{ijxk} to the credit card/e-banking enclave.

(**4**) The enclave fills C_x into the request while taking the policy P_{ijxk} into account and forwards it to the merchant.

(**5**) Finalization is done by the payment service provider.

(**6**) Response is routed through the enclaves to B_j.

5.3 Full Website Access

HTTPS Proxy. For secure browsing we implemented a HTTPS proxy enclave. We want to proxy selected websites and if a user leaves the website, he also leaves the proxy. We implemented this by using cookies to set the correct host name. The user sends any request to the proxy and he sets a cookie with the host name he wants to visit through the proxy. The enclave then parses the request, replaces the host name and sends it on to the real website. The response is also modified by the enclave so that the host name points to the proxy again. All links in the response are left unmodified so all relative links point to the proxy but all absolute links direct to a different website. The website certificates are checked against the statically compiled root certificate list in the enclave.

Login. To log into a service using delegated credentials we leverage similar technologies as in the HTTPS proxy and we thus only extended the proxy enclave to support delegated authentication for websites. Analogous to the HTTPS proxy we use cookies to specify the Delegatee's *session token* and which credentials C_x she wants to use. The enclave then asks the API whether the Delegatee with the specified *session token* is allowed to use C_x. If everything checks out, the API responds with the details of C_x and P_{ijxk} and the proxy enclave fills the login form before forwarding it to the website. As websites *session tokens* are usually stored in cookies, we encrypt all cookies forwarded to and from the website in order to prevent session stealing by an adversarial Delegatee. We use the browser extension in the same way as in the PayPal example: a button is rendered next to the original login. Figure 4 depicts the architecture and the detailed steps:

(**1**) The Delegatee B_j wants to log into a website using some credentials C_x that have been delegated by A_i. B_j connects to the website and the browser extension renders a second button beside the normal login button.

(**2**) Upon clicking this button, the browser extension changes the URL pointing to the proxy and appends cookies, specifying the credentials B_j wants to use.

(**3**) The proxy asks the API for C_x. The API checks if B_j has the rights to use C_x and then forwards C_x.

(**4**) The proxy enclave fills in the username and password into the login request and proceeds to send it to the website and receives the response.

(**5**) The proxy rewrites the header of the response to encrypt cookies and then forwards it to B_j.

(**6**) All subsequent connections have to go through the proxy where the policy P_{ijxk} can be enforced.

6 Performance analysis

In this section we show that the overhead imposed by our solution stays within reasonable bounds. The performance testing was done using two i7-7700 machines with 16 GB RAM, connected via the internet and local network. We can serve around 100 users concurrently even running on consumer grade hardware.

Table 2-a shows an overhead of around 50ms for a full SSL handshake using `mbedtls` inside an enclave. The handshake involves three exchanged messages, thus at

Type	Test case	Mean (± std)
a) SSL handshake	`openssl`	52.12ms (± 3.62)
	`mbedtls`	57.14ms (± 3.37)
	`mbedtls` in SGX	105.22ms (± 4.23)
b) Mail	direct	1.12s (± 0.27)
	mail enclave	1.19s (± 0.22)
	API/mail enclave	1.45s (± 0.25)
c) PayPal	direct	25.92s (± 6.83)
	direct, no js	29.96s (± 8.51)
	PayPal enclave	27.00s (± 4.35)

Table 2: Latency for *a*) SSL handshakes, *b*) receiving e-mails in inbox, and *c*) executing PayPal transactions. Sample: 1000.

Target (site)	Test case	Mean (± std)
small (2.6KB)	direct	5.0ms (± 2.7)
	proxy enclave	64.3ms (± 2.5)
medium (411KB)	direct	12.2ms (± 1.2)
	proxy enclave	76.8ms (± 3.3)
big (15.7MB)	direct	202.6ms (±19.9)
	proxy enclave	432.2ms (±16.0)

Table 3: HTTPS proxy latency with various page sizes.

least three `ocalls`/`ecalls`, all of which have to copy buffers. In our measurements we recorded 19 `ocalls` during a request to the enclave. Overhead for `ocalls` and `ecalls` is measured and analyzed in [40] and is significant for copying buffers from the untrusted memory.

The mail enclave incurs minimal overhead (Table 2-b) with the extra handshake to the IMAP server (P2P system). In our test we retrieve all emails from the account inbox. In the Centrally Brokered system an additional handshake with the API is leading to a higher delay.

The PayPal example does not seem to suffer from any delay added by our implementation (Table 2-c). Note that we performed tests using the sandbox environment, provided by PayPal itself for testing integration with their services. This environment is feature-complete but slow as it is only functionality-oriented. Most time falls in waiting for the PayPal servers. As the enclave uses the fallback mechanism to execute PayPal transactions without JavaScript, we measured both variants: one allowing JavaScript and one blocking it. We also conducted tests in the real PayPal environment using the Centrally Brokered system, executing a real payment and buying an item online with a merchant supporting PayPal. However, due to the *CAPTCHA* protective mechanism involving the Delegatees' actions, it is not feasible to measure performance, since it depends on the user input.

Table 3 shows that the proxy adds the biggest latency overhead compared to normal browsing but still below 0.1 seconds for small to medium websites, a response time limit for seamless user interaction [35]. A part of the high delay stems from the enclave waiting for the whole web server response before forwarding it to the Delegatee. Message parsing, the additional handshake, and the fact that all communication has to cross the `ocall`/`ecall` interface twice also adds to the overhead. A full HTTPS proxy enclave is in the works to reduce the waiting time and support all client connections.

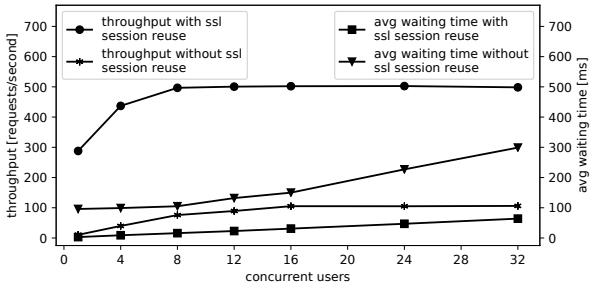


Figure 5: Concurrency shown in throughput and average waiting time. Each user tries to send 100 requests.

We have also tested video streaming through our proxy, supporting DELEGATEE's streaming service examples (i.e., Netflix). We modeled streaming as a client that requests some video from a webserver. Therefore the performance of video-streaming through DELEGATEE is analogous to the ordinary HTTPS proxy use case. There was no additional overhead compared to normal streaming for a single user, e.g. as in the P2P model (the standard deviation is larger than the initial waiting time, for both the normal streaming and the proxied one). The streaming service was tested on the Centrally Brokered system where the delegatee connects to the proxy from the internet. This test was only done for a single user streaming at one point in time due to hardware and bandwidth limitations. As in the previous test, the overhead is negligible once the streaming starts while the initiation depends on the current latency.

Our multithreaded implementation was tested using 8 threads. Incoming connections are kept in a queue and served by the enclave threads, thus reaching maximum throughput with 8 concurrent users sending requests, as shown in Figure 5. The average waiting time stays constant until the same 8 user threshold, increasing linearly as new requests get queued. Our implementation supports SSL session reuse which significantly improves the throughput and lowers the waiting time. Without session reuse we can accommodate maximum 100 req/sec for 32 concurrent users, while with session reuse this grows to 500. Numbers could vary depending on the chosen cipher suite (ECDHE-RSA-AES256-GCM-SHA384).

7 Discussion & Limitations

In this section we explore limitations of DELEGATEE, mainly focusing on how brokered delegation faces technical challenges in the authentication process, as well as business and regulatory challenges arising from users controlling their own resources in a more flexible and fine-grained way than service providers intend.

Authentication challenges. Authentication in modern web services is complex. It can involve not just passwords but additional factors such as personal questions, email challenges, phone challenges, and "two-step authentication" apps such as Authy and Google Authenticator. Some of these can be supported with DELEGATEE, such as, email challenges or 2FA apps that could run inside the enclave as well, while for some, e.g. phone challenges, DELEGATEE cannot overcome the challenge.

Contextual factors often additionally come into play, such as the IP address, time of day, and nature of service requests. Financial services, e.g., PayPal, have particularly sophisticated fraud detection regimes; e.g., ordering unusual products with Paypal may trigger a fraud alert. Consequently, a single credential in the form of a password may not suffice to delegate a resource or service via DELEGATEE. In Section 5 we outline a solution for an additional authentication method in form of a CAPTCHA, that is required by some online services.

To illustrate, consider a scenario in the P2P mode where an Owner Alice (an inhabitant of the U.S.) delegates a password to DELEGATEE and allows her PayPal account to be rented. Suppose then that Delegatee Bob, in Nigeria, rents Alice's PayPal account in a prescribed way and attempts to execute a transaction. Paypal will see an unusual request coming from an IP address in a country with a different risk profile than the U.S., and potentially one that Alice has never visited. Bob's transaction request is likely to be suspicious. PayPal may then deny the transaction or request additional confirmation, e.g., via e-mail, to proceed. If Alice is unavailable or denies the transaction - which she may fail to recognize as originating with her delegation - the transaction will fail.

For future production deployment of DELEGATEE, we will address these complications in several ways:

• *Application-specific delegation:* Authentication systems vary considerably across applications and service providers. Each DELEGATEE application will include configuration not just for the APIs of a given target Service, but also its authentication policies.

• *Delegation of multiple credentials:* For services that require multiple credentials, DELEGATEE may require more than a password from an Owner. For example, two-step authentication apps can be executed within the enclaved DELEGATEE application and set up by an Owner as an additional authentication factor. Similarly, an Owner may delegate her email to the enclave to respond to email-based authentication challenges. The SGX platform performing the delegation may be situated in the same country or region as the Owner. Finally, an Owner can perform a set of legitimate transactions through DELEGATEE in order to confirm that required credentials are present and to white-list the platform with the authentication system of the target service.

• *Failure modes:* Periodic delegation failures are inevitable, just as legitimate users' transactions fail sporadically due to false positives in the fraud-detection systems. As DELEGATEE is not intended for mission-critical uses, it could include graceful failure modes.

Authentication collisions. Attempts at simultaneous use of a resource may fail, as many web services do not support multiple concurrent sessions for a given account. For example, if Alice has delegated use of her bank account to Bob, then she may be unable to use it herself while Bob (or DELEGATEE, to be precise) is logged in. Such collisions can be treated by invoking failure modes like those for basic authentication failures. Other policies are possible, however. For example, Owner Alice may set a policy that only delegates her resource at times when she is unlikely to use it. A small enhancement to DELEGATEE can also enable Alice to preempt the session of a Delegatee if desired.

Usability, Deployment and Service Prevention. Throughout the paper we have presented multiple use-cases and implemented prototypes that support delegation of different services. The usability of these services by potential Delegatees is as if they were using the original service as its Owner. However, the usability of the DELEGATEE in general depends on the supported use-cases. A limitation of our system is that for each and every use-case a specific module (that matches the capabilities and technical challenges) has to be implemented. Until now, we have not found a way in order to develop a generic module that could support a wide variety of services. For example, interpreted languages, such as Javascript, remain an open problem since by executing unmeasured code in an enclave running the interpreter we cannot guarantee the security properties of DELEGATEE. In addition to that, almost all services (even the ones from the same category) have different user mechanisms, UI and control. Thus, a specific policy needs to be created that matches these controls in order to allow Owners to specify how their service could be used by potential Delegatees. Due to the complexity, for now, the policies have to be created beforehand along with the implemented delegation scenario, while the end-user involvement is limited to configuring parameters, out of a set of given policy characteristics.

If all service operators would share a unique set of API calls that could cover the full functionality of their services, then the deployment of DELEGATEE would be feasible for almost all service categories. This would also allow for the creation of more general and richer access control policies that could be created by the end-users of the service as well, possibly overcoming the initially dis-

cussed complexity of complete policies that require serious engineering and evaluation of each specific use-case scenario.

However, it is hard to imagine that the service operators would view the above even as a viable option. In many cases, DELEGATEE allows the creation of secondary markets (see the last paragraph of the section) and poses a threat to the revenue stream of the original service operator. Additionally, DELEGATEE reduces the operators' ability to control and track their users since virtually, the number of users could grow but they would be seen only through the increased activity of users registered to the original service. Thus, most service operators would try to deny service access if executed through this form of delegation. As already mentioned, IP geofencing, pattern matching of actions and service usage, 2FA, along with the already existing fraud-detection mechanisms may endanger the functionality of our system. We have addressed several of them, however, future work involves investigation into further improvements that could make the distinction between the Owner and any Delegatee less possible.

Scalability. Scalability for all other supported services except video streaming is generally not a constraint. It comes down to running a proxy which can be adapted in terms of processing power (adding more enclaves horizontally) like any other service provider, while the bandwidth requirements remain moderate. However, in the case of video streaming in the centralized approach, the limitation is in the number of running connections since all video material is re-routed through the proxy. Namely, the proxy would need to have extremely high bandwidth, processing power and be scalable almost as the video service provider itself. We did not perform scalability tests to see how many users in parallel we could support for the video streaming example. This would require server grade hardware which we do not possess and any reported results would be meaningless. However, for the P2P model, since the enclave resides on the Delegatees themselves, a single Owner can support multiple delegation of his, e.g. Netflix account (at least based on the limit of Netflix itself – 2 or 4 devices based on the subscription). The streaming is done directly to the Delegatee, and the access will be valid until the policy expires.

Secondary markets. Brokered delegation could give rise to offerings that compete directly with those of the very platforms hosting the delegated resources.

Facebook users could sell opportunities for "sponsored post" - unsolicited advertisements sent to their networks of friends or shown on their walls, as discussed above. Facebook users would then compete with Facebook itself in selling ads. Similarly, users could rent use of their Netflix account. Account sharing is already common within families and close friend circles. Brokered delegation could enable broad reselling and foster competition with direct sales of the subscription service.

Such secondary markets would in many cases violate providers' existing terms of service and might resemble markets for underground sales of virtual goods [27, 44]. Those underground markets have met with two responses, sometimes used in tandem: (1) providers aim to detect facilitators of secondary markets and penalize or ban them, and (2) providers themselves seek to capture the revenue streams generated by secondary markets; e.g., online role-playing game providers have offered virtual goods for sale through their own shops [31]. DELEGATEE could provoke similar responses.

Peer-to-peer cryptocurrency-for-fiat exchanges is another setting that can benefit from DELEGATEE. Today, websites like LocalBitcoins.com receive Bitcoin deposits and hold them in escrow. Then they match-make and allow a buyer and a seller to negotiate a e-banking transfer. When the receiver gets the bank transfer, they instruct the LocalBitcoins service to complete the payment from the escrowed funds. If the receiver raises a dispute, then the service must investigate and ultimately determine whether to release the funds. However, such services naturally have limited investigative ability. They may call the user's bank, or ask both parties for evidence (i.e., screenshots). Neither option is satisfactory; the latter is prone to forgery, while the former may inadvertently draw suspicion to the user's bank account. Credential delegation provides an alternative, simplifying this business model and implementing a secure intermediary that guarantees execution and fair exchange.

8 Related Work

TEEs are widely used today. ARM TrustZone, for example, is commonly used to protect data on mobile devices, e.g., biometric templates and encryption keys in iOS devices [4]. Intel SGX has been proposed for a number of applications, including confidential map-reduce tasks [37], trustworthy data feeds for blockchain oracles [45] and retrofitting of legacy applications [7], secure payment channels [30], etc. With DELEGATEE we extend this line of work with a new class of applications based on credential delegation.

Delegation of authority has been an important focus in access control security. Two mechanisms are commonly used. First, the credential Owner can interact with an authentication service to mint new credentials or tokens (representations of capabilities) for the Delegatee (e.g., Active Directory, Kerberos, and Oauth [17, 18]). Second, using chains of cryptographic assertions or certificates (as in X.509 or SPKI/SDSI), which can be digitally signed and communicated without interacting with

a central server [10, 15, 34, 8, 6]. In either case, the delegation mechanism must be supported by the resource (or a reference monitor guarding the resource). Our system is different in that we use a trusted enclave-based proxy that stores the user's credentials and is transparent to the resource. It is, therefore, used to retrofit delegation for existing web services, without requiring additional effort (or even explicit support) from the provider.

Many web services like Facebook, and Twitter, support delegation for third-party applications typically using OAuth or OpenID (e.g., a user may delegate to a Facebook app the authority to read her friends-list but not to post new messages on her behalf). However, this delegation is not very expressive. The authority to post on a users Facebook wall is all-or-nothing, for example; we cannot express restrictions such as no more than 1 post per day. Much of the research literature has focused on flexible languages for specifying and reasoning about delegation policies [10, 6, 38, 19]. Our approach is complementary, as our enclave-based proxy can be used to apply more expressive policies to existing services.

Without support for fine-grained delegation, users sometimes resort to sharing passwords with each other or with third parties [38]. For example, to use the financial dashboard service Mint.com, users often need to share their bank account passwords with the service [41].

Delegation based on TEEs promises a more secure alternative to this status quo. Credential delegation using SGX was first explored in [45] to support "oracle" queries. Use of SGX for credential management was also proposed in [23]; there the goal was validation and resale of credentials for criminal purposes. More recent work involves the delegation of private keys for cryptocurrencies in order to secure a payment channel [30]. DELEGATEE is much more general than these prior works, as it supports delegation of credentials for any desired goal.

9 Conclusion

In this paper we propose a new concept called brokered delegation, using TEEs to enable flexible delegation of credentials and access rights to internet services. We explored two design spaces, the decentralized P2P mode as well as a more pragmatic Centrally Brokered mode. Our implementation and experiments show that Delegatee in either mode can be applied to several real-world applications with minimal overhead, while preserving security against a strong attacker. Delegatee therefore has potential to enable delegation for any existing services, even without support from the service itself. This raises significant questions for future work: Can we enable robust delegation even against services that act to prevent it? Or can services defend against unwanted delegation? Lastly, given secure delegation, how would the economy of online services change?

References

[1] Intel Software Guard Extensions, Reference Number: 332680-002, 2015. https://software.intel.com/sites/default/files/332680-002.pdf.

[2] ALEXANDER, B. Introduction to Intel SGX Sealing, 2016. https://software.intel.com/en-us/blogs/2016/05/04/introduction-to-intel-sgx-sealing.

[3] ALVES, T., AND FELTON, D. TrustZone: Integrated Hardware and Software Security-Enabling Trusted Computing in Embedded Systems, 2004.

[4] APPLE. iOS Security. Whitepaper, https://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2017.

[5] BARAK, B., GOLDREICH, O., IMPAGLIAZZO, R., RUDICH, S., SAHAI, A., VADHAN, S., AND YANG, K. On the (Im)Possibility of Obfuscating Programs. In *Annual International Cryptology Conference* (2001), Springer, pp. 1–18.

[6] BAUER, L., SCHNEIDER, M. A., AND FELTEN, E. W. A General and Flexible Access-Control System for the Web. In *USENIX Security Symposium* (2002), pp. 93–108.

[7] BAUMANN, A., PEINADO, M., AND HUNT, G. Shielding Applications From An Untrusted Cloud With Haven. *ACM Transactions on Computer Systems (TOCS) 33*, 3 (2015), 8.

[8] BIRGISSON, A., POLITZ, J. G., ERLINGSSON, U., TALY, A., VRABLE, M., AND LENTCZNER, M. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *NDSS* (2014).

[9] BOHANNON, J. Who's downloading pirated papers? Everyone, 2016.

[10] BORISOV, N., AND BREWER, E. A. Active Certificates: A Framework for Delegation. In *NDSS* (2002).

[11] CHECKOWAY, S., AND SHACHAM, H. Iago attacks: Why the system call API is a bad untrusted RPC interface. *ACM 41* (2013).

[12] CORON, J.-S., LEE, M. S., LEPOINT, T., AND TIBOUCHI, M. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *IACR International Workshop on Public Key Cryptography* (2017), Springer, pp. 41–58.

[13] COSTAN, V., AND DEVADAS, S. Intel SGX explained. In *Cryptology ePrint Archive* (2016).

[14] DOLEV, D., AND YAO, A. On the security of public key protocols. *IEEE Transactions on information theory* (1983).

[15] GASSER, M., AND MCDERMOTT, E. An architecture for practical delegation in a distributed system. In *IEEE Computer Society Symposium on Security and Privacy* (1990), IEEE, pp. 20–30.

[16] HALDERMAN, J. A., SCHOEN, S. D., HENINGER, N., CLARKSON, W., PAUL, W., CALANDRINO, J. A., FELDMAN, A. J., APPELBAUM, J., AND FELTEN, E. W. Lest we remember: Cold-boot Attacks on Encryption Keys. *Communications of the ACM 52*, 5 (2009), 91–98.

[17] HAMMER-LAHAV, E. The Oauth 1.0 Protocol. RFC 5849, https://tools.ietf.org/html/rfc5849, 2010.

[18] HARDT, D. The OAuth 2.0 Authorization Framework. RFC 6749, https://tools.ietf.org/html/rfc6749, 2012.

[19] HOWELL, J., AND KOTZ, D. An Access-Control Calculus for Spanning Administrative Domains. *Dartmouth College* (1999).

[20] INTEL. SGX SDK, 2016. https://software.intel.com/en-us/sgx-sdk.

[21] INTEL. Intel Software Guard Extensions - Developer Zone. Website, https://software.intel.com/en-us/sgx, 2017.

[22] JOHNSON, S., SCARLATA, V., ROZAS, C., BRICKELL, E., AND MCKEEN, F. Intel SGX: EPID provisioning and attestation services, 2016. https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation.

[23] JUELS, A., KOSBA, A., AND SHI, E. The Ring of Gyges: Investigating the Future of Criminal Smart Contracts. In *CCS* (2016).

[24] KAUER, B. OSLO: Improving the Security of Trusted Computing. In *USENIX Security Symposium* (2007), pp. 229–237.

[25] KLARNA. Sofort - Einfach und direkt bezahlen. Website, `https://www.klarna.com/sofort/`, 2017.

[26] LAURINAVICIUS, T. What Successful Entrepreneurs Outsource to a Virtual Assistant. Entrepreneur, `https://www.entrepreneur.com/article/300195`, 2017.

[27] LEHDONVIRTA, V., AND CASTRONOVA, E. *Virtual economies: Design and analysis*. MIT Press, 2014.

[28] LEVIN, T., PADILLA, S. J., AND IRVINE, C. E. A Formal Model for UNIX Setuid. In *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on* (1989), IEEE, pp. 73–83.

[29] LIMITED, A. mbedTLS (formerly known as PolarSSL), 2015. `https://tls.mbed.org/`.

[30] LIND, J., EYAL, I., KELBERT, F., NAOR, O., PIETZUCH, P., AND SIRER, E. G. Teechain: Scalable Blockchain Payments using Trusted Execution Environments. *arXiv preprint arXiv:1707.05454* (2017).

[31] MACK, C. Virtual Goods and Mainstream Game Companies. AdWeek, `http://www.adweek.com/digital/mainstream-games-companies-virtual-goods/`, 2009.

[32] MATETIC, S., AHMED, M., KOSTIAINEN, K., DHAR, A., SOMMER, D., GERVAIS, A., JUELS, A., AND CAPKUN, S. ROTE: Rollback Protection for Trusted Execution. *26th Usenix Security Symposium* (2017).

[33] MCKEEN, F., ALEXANDROVICH, I., BERENZON, A., ROZAS, C. V., SHAFI, H., SHANBHOGUE, V., AND SAVAGAONKAR, U. R. Innovative instructions and software model for isolated execution. In *HASP@ ISCA* (2013).

[34] NEUMAN, B. C. Proxy-based authorization and accounting for distributed systems. In *In Proceedings the 13th IEEE Conference on Distributed Computing Systems* (1993), pp. 283–291.

[35] NIELSEN, J. Website Response Times. Nielsen Norman Group, `https://www.nngroup.com/articles/website-response-times/`, 2012.

[36] SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. Role-based Access Control Models. *Computer 29*, 2 (1996), 38–47.

[37] SCHUSTER, F., COSTA, M., FOURNET, C., GKANTSIDIS, C., PEINADO, M., MAINAR-RUIZ, G., AND RUSSINOVICH, M. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *IEEE S&P* (2015).

[38] SINGH, S., CABRAAL, A., DEMOSTHENOUS, C., ASTBRINK, G., AND FURLONG, M. Password sharing: implications for security design based on social practice. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2007), ACM, pp. 895–904.

[39] THOMAS, T. A mandatory access control mechanism for the Unix file system. In *Aerospace Computer Security Applications Conference, 1988., Fourth* (1988), IEEE, pp. 173–177.

[40] WEISSE, O., BERTACCO, V., AND AUSTIN, T. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (2017), ACM, pp. 81–93.

[41] WESTON, L. Why banks want you to drop Mint, other 'aggregators'. Reuters, `https://www.reuters.com/article/us-column-weston-banks/why-banks-want-you-to-drop-mint`, 2015.

[42] WINTER, J. Trusted Computing Building Blocks for Embedded Linux-based ARM Trustzone Platforms. In *Proceedings of the 3rd ACM workshop on Scalable Trusted Computing* (2008).

[43] WOJTCZUK, R., AND RUTKOWSKA, J. Attacking SMM memory via Intel CPU cache poisoning. *Invisible Things Lab* (2009).

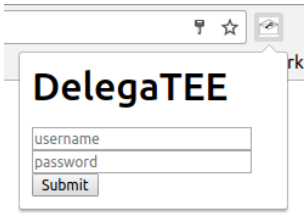
[44] XIE, Y. The Underground Market For In-Game Virtual Goods. TechCrunch, `https://techcrunch.com/2016/01/20/virtual-goods-real-fraud/`, 2016.

[45] ZHANG, F., CECCHETTI, E., CROMAN, K., JUELS, A., AND SHI, E. Town Crier: An Authenticated Data Feed for Smart Contracts. In *CCS* (2016).

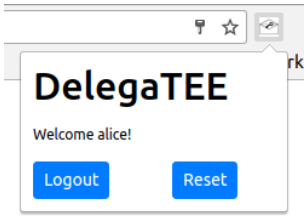
A DELEGATEE Prototype Demo

In this section we show prototype screenshots when a Delegatee, Alice, is buying something or logging in to a website using DELEGATEE. First, Bob enters his credentials into DELEGATEE and delegates them to Alice. Alice then logs into the browser extension (Figure 6a, Figure 6b) and the new button appears next to the PayPal checkout button (Figure 6c), the credit card/e-banking checkout button (Figure 6d) or the login button (Figure 6e). After clicking the DELEGATEE button, Alice is presented with a list of delegated credentials to choose from (Figure 6f). Upon selecting some credentials, the enclave takes over and completes the transaction and Alice is redirected to the confirmation page. If a CAPTCHA has to be solved to continue with the transaction, the user is asked to solve (Figure 6g).

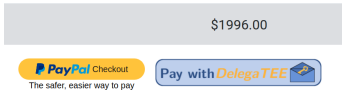
Receiving and sending emails using delegated credentials can be done with our mail client for DELEGATEE. It allows to view the inbox and read single mails of the delegated mail account (Figure 7a). Sending emails is also supported (Figure 7b).



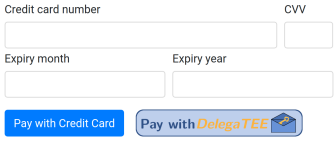
(a) Browser extension: Login



(b) Browser extension: Welcome

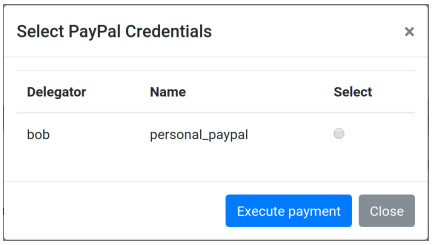


(c) Extra button rendered next to the PayPal checkout button



(d) Extra button rendered next to the credit card checkout button

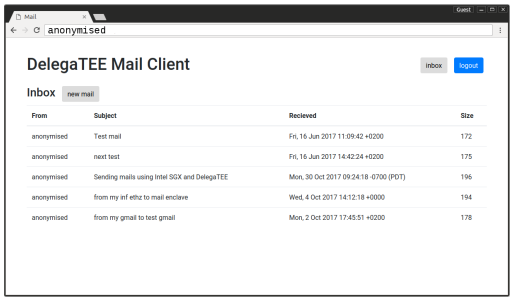
(e) Extra button rendered next to the login button



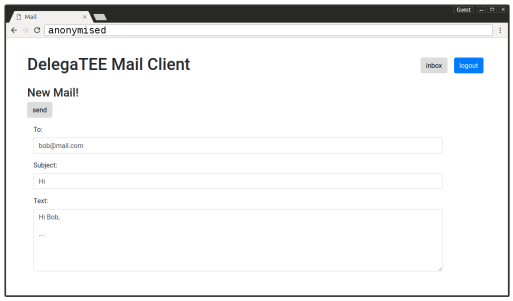
(f) Delegated credentials selection.

(g) The Delegatee is asked to solve CAPTCHA

Figure 6: Demo of a payment/login process using DELEGATEE. The buttons and the dialog get injected to the website by the browser extension.



(a) Receiving mail



(b) Sending mail

Figure 7: DELEGATEE mail client example. All links and other details have been anonymized for review.