SECURING MEDICAL SAAS SOLUTIONS USING A NOVEL END-TO-END ENCRYPTION PROTOCOL

Prototype

Slawik, Mathias, Technische Universität Berlin, Ernst-Reuter Platz 7, 10587 Berlin, Germany, mathias.slawik@tu-berlin.de
Ermakova, Tatiana, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany, tatiana.ermakova@tu-berlin.de
Repschläger, Jonas, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany, j.repschlaeger@tu-berlin.de
Küpper, Axel, Technische Universität Berlin, Ernst-Reuter Platz 7, 10587 Berlin, Germany, axel.kuepper@tu-berlin.de
Zarnekow, Rüdiger, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany, ruediger.zarnekow@ikm.tu-berlin.de

Abstract

E-Health solutions using the Internet provide many benefits for health centers; hosting such solutions in public Cloud Computing environments as Software-as-a-Service becomes increasingly popular. However, the deployment of e-health services in shared environments is restricted due to regulations prohibiting medical data access by illegitimate parties, such as cloud computing intermediaries. A pivotal requirement is therefore having security “end-to-end”, namely from a user agent to the server process; yet there is no viable approach for contemporary browser-based SaaS solutions. This paper outlines a blueprint for e-health solution architectures featuring an end-to-end security mechanism to prevent intermediary data access and therefore to ensure appropriate patient data privacy and security. This blueprint is instantiated based on a novel security protocol, the Trusted Cloud Transfer Protocol (TCTP) in the form of a prototype implementation. The evaluation of the prototype demonstrates its fulfilment of healthcare-specific security and privacy requirements, as well as low implementation efforts for similar architectures, and no measurable performance overhead in a practical benchmark.

Keywords: Cloud Computing, Healthcare, End-to-End Security, Medical Protocol
1 Introduction and Related Work

Despite the currently observable advances in healthcare information technologies and electronic health (e-health), delays of decisions about medical treatment or medical diagnostic procedures repetitively performed are still common causes of long waiting times for medical records from other healthcare providers being transferred by casual means. The healthcare industry demand for IS-solutions addressing this problem is growing. As a consequence, healthcare centers consider adopting new information technologies, such as cloud computing.

Public cloud Software-as-a-Service (SaaS) offerings, such as Salesforce Sales Cloud and Google Drive for Business, have been adopted among a considerable number of users in their respective application domain. In healthcare, they could enjoy a high adoption as well. Especially in terms of medical records sharing, cloud computing promises substantial improvements (e.g., Chen et al., 2012). Nevertheless, as argued by Li et al. (2012) and Chen et al. (2012) the affected parties could be concerned about patient data security and privacy. Thereupon, sensitive medical records are the subject of individuals’ protection plans issued by lawmakers and implemented by healthcare centers. In particular, the regulations require the prevention of patient data access by illegitimate parties, such as cloud computing intermediaries found within most public cloud offerings. There are a number of proposed countermeasures to address these issues in the related work as summarized by Abbas and Khan (2014). However, the end-to-end security of medical records communication with medical SaaS solutions we address in the present paper has not been the scope of research so far.

The presented research is carried out as part of the ongoing project TRESOR (TRESOR, 2014) with two German hospitals. The goal is to enable the German health care sector to use e-health SaaS applications in a way compliant to all regulatory provisions. Our research methodology is in line with the design science approach introduced by Hevner and Chatterjee (2010), and Peffers et al. (2008). Based on a review of related work and workshop sessions with German healthcare experts, where we revealed the essential business needs to be addressed by our approach, we developed an architecture blueprint and built a prototypical instantiation using state-of-the-art technologies. In the assessment part of our research, we discuss how the healthcare-specific needs are met, demonstrate the implementation efforts, and measure the performance and overhead. The findings of our work will be validated by two German health centers, which are an appropriate environment for the application of our work. As our blueprint is applicable to all RESTful applications, it is not limited to any application domain. However, applying the blueprint for medical SaaS solutions provides the basis for hosting them in shared cloud environments.

This paper is organized as follows: Section 2 presents the proposed architecture blueprint, outlines its application domain and goals, as well as its conceptual components which overcome current obstacles in the deployment of public cloud healthcare solutions. Section 3 shows the prototypical implementation of the blueprint, as well as its practical relevance through a thorough evaluation. We conclude this paper and outline limitations and future work in Section 4.

2 Architecture Blueprint

This section presents our blueprint for medical solution architectures, which employs end-to-end encryption for securing them. After outlining the application domain of architectures build upon this blueprint we derive the goals of the blueprint. This section concludes by presenting the components of the blueprint in detail. The blueprint provides a foundation for the implemented prototype, which is presented in Section 3.
2.1 Application Domain

As almost all SaaS cloud applications implement a RESTful design, we consider our blueprint to be applicable for instantiating architectures of medical hypermedia applications, more specifically, RESTful (Representational State Transfer) HTTP applications, which are deployed as SaaS in a shared environment, such as a Platform-as-a-Service (PaaS) solution. The application characteristics include:

**RESTful design:** Besides utilizing HTTP as a transfer protocol, the applications should follow the conceptual framework given in the work by Fielding (2000), i.e., the interaction with these applications through user agents should represent operations on resourceful abstractions of health data, e.g., patient records, treatment journals, or medication plans. By following the RESTful design principle, our architecture blueprint can make generalized assumptions about application characteristics, such as HTTP operation semantics, and therefore provides a generalized solution for securing such applications.

**Deployment in a shared environment:** The application of end-to-end encryption carries security benefits for any solution architecture, e.g., by preventing the access to communication plaintext by intermediaries. Applying such a security measure is especially meaningful within shared environments, such as public cloud offerings, as those are associated with extensive ramifications of security breaches of those intermediaries, where a potentially large number of tenants would be affected. Some offerings do also have a high visibility, such as Amazon EC2 or Microsoft Azure, making these offerings and the solutions deployed on them a worthwhile target for security attacks.

**Communication through HTTP management proxies:** If a user agent and an origin server would be connected directly, transport layer protocols, such as Transport Layer Security (TLS) could provide end-to-end security (Stallings, 2003, pp. 530-548). However, most cloud computing environments include intermediary management proxies, such as load balancers, reverse proxies, and caching cloud optimizers. Furthermore, many organizations opt to filter outgoing internet traffic to impose restrictions on their employees’ internet access, introducing yet another intermediary. To carry out their tasks, proxies need to have access to communication plaintext, which is only possible if they act as TLS server connection ends. This violates the “Need-to-know-Principle”, as those intermediaries do not need to have access to the HTTP entity-body to carry out their functions, yet can access it at any time.

2.2 Requirements for Medical Hypermedia Applications

With our blueprint as a foundation for medical hypermedia applications, we aim to satisfy the following requirements, derived from the interviews with German healthcare experts:

**R1 - End-to-end confidentiality of transmitted health data:** Within medical RESTful applications, health data is contained in the entity bodies of HTTP requests and responses. Considering this, confidentiality of health data is achieved, if unauthorized intermediaries are prevented from accessing those entities. For instance, the entities could be encrypted whereas the parties negotiating encryption keys involve the client, e.g., the workstation from which health data is accessed, and the origin server, i.e., the (possibly virtualized) server where these records are stored; otherwise, e.g., when just encrypting hop-to-hop, for example by establishing a TLS connection from the client to a cloud proxy, the confidentiality would not be “end-to-end”.

**R2 - End-to-end integrity of transmitted health data:** On the way through HTTP management proxies, health data should remain accurate and consistent. Possible violations include altering, reordering, and replaying previous data.

**R3 - Customer control over encryption keys:** As emphasized by our experts, other procedures, e.g., provider-issued encryption keys, would be unacceptable from a practical security viewpoint, and would also violate legal provisions.
R4 - Low implementation efforts and performance overheads: To ensure the practical applicability of the blueprint, the blueprint components should impose a low overhead for encryption and authentication, possibly relying on hardware encryption functionality, such as AES-NI (Stallings, 2003, pp. 139-167). Furthermore, the effort for implementing such architectures should be taken into consideration, for example, by reusing existing components, extensive modularization, or reliance on established and mature technologies.

2.3 Blueprint Components

This section describes the components which enable architectures based on this blueprint to accomplish the exemplified requirements. There are two main components in this blueprint: The security technology enabling end-to-end HTTP entity body encryption, and software on the client and server implementing this security technology. These are shown in Figure 1, along with a short description of their role:

![Figure 1. The blueprint components.](image)

As the blueprint is applicable to all RESTful applications, these components should be fully compatible to HTTP, i.e., cloud intermediaries should be able to process exchanged messages, and the client and server security software should also be able to differentiate between regular and encrypted HTTP traffic. The following subsections particularize the requirements which the blueprint components should meet.

Security technology. The security technology included within architectures based on this blueprint should have the following functionality to meet the requirements listed in the preceding section:

- Encryption of HTTP entity bodies, with strong encryption algorithms;
- Secure authentication of encrypted HTTP entity bodies, preventing the alteration of encrypted data;
- Message-flow protection, i.e., detection capabilities for replayed or reordered encrypted data;
- Supporting partial HTTP processing. If the technology would only operate on complete entities, this would prevent media streaming applications such as telemedicine, and HTTP comet technologies such as HTTP server push.
Client & server security software. The client and server security software should handle the en- and decryption of HTTP entity bodies. Client security software can be instantiated as browser add-ons, HTTP client library extensions, or local HTTP proxies. Server software can be either integrated into application frameworks, server middleware, HTTP servers, or remote HTTP proxies. Furthermore, it should support the following operations:

- Secure on-line exchange of encryption keys used by the security technology to encrypt and authenticate HTTP entity bodies, possibly using strong algorithms, such as Elliptic Curve Cryptography.
- Negotiation of key exchange and encryption algorithms. This enables independent development of multiple client and server security software with differing capabilities.
- Discovery of origin server encryption capabilities by user agents preventing unnecessary round trips, e.g., when sending encrypted content to a server, which lacks the server security software.
- Specifying differentiated encryption, i.e., encrypting sensitive operations, such as updating patient records, while transferring interface assets, such as style sheets or icons, in plaintext, therefore lowering the overall encryption overhead.

3 Prototypical Architecture Instantiation and Evaluation

In this section, we show an example instantiation of the blueprint using the novel Trusted Cloud Transfer Protocol (TCTP) (Slawik, 2013) as end-to-end encryption security technology. We chose TCTP, as it is the only end-to-end encryption security technology which meets all of the requirements set in the preceding section. After a short description of TCTP, the details of the prototype implementation are explicated, and the evaluation results how the prototype meets the goals of the architecture blueprint are provided.

3.1 The Trusted Cloud Transfer Protocol (TCTP)

As argued by Slawik (2013), established security technologies for HTTP end-to-end encryption have diverse shortcomings, which his proposal, the Trusted Cloud Transfer Protocol (TCTP), surmounts. Moreover, TCTP is the only end-to-end encryption based technology which meets all of the requirements set in the preceding section. The main concept of TCTP is the notion of HTTP Application Layer Encryption Channels (HALECs), which encapsulate the mature and well-tested Transport Layer Security (TLS) protocol, so that HTTP entity-bodies are fragmented into encrypted and authenticated TLS records. The fragmentation permits partial HTTP processing, while the notion of a channel with an associated TLS session state enables the discovery of replayed and reordered HTTP messages through the TLS HMAC (Stallings, 2003, pp. 372-376) mechanism.

Another concept of TCTP is the encapsulation of the TLS handshaking protocols, which are used to set up those HALECs, thus enabling the secure on-line exchange of encryption keys, and the negotiation of encryption algorithms. This secure exchange can be performed using strong encryption algorithms, such as Elliptic Curve Diffie-Hellman (Stallings, 2003, pp. 304-308). At last, TCTP includes a discovery mechanism, where origin servers are able to state which resources are protected by TCTP, and how to create HALECs to access those resources.

3.2 The Prototype Architecture

The prototype architecture consists of the following components: A user agent on a client communicating with a local TCTP proxy. This proxy carries out TCTP discovery, handshake, and
entity-body encryption and was built specifically for this paper. It is a flexible, customizable HTTP reverse/forward proxy written in Ruby using the Eventmachine Ruby gem. This proxy communicates with the Redhat OpenShift platform, on which a PaaS container (a “gear”) is deployed. This container runs a RESTful HTTP application, the TRESOR Demo application, which imitates a contemporary cloud-based patient data management system. This application was extended through the TCTP Rack middleware by Slawik (2013), which fulfills the server role of TCTP. Every OpenShift compute node features a cloud intermediary, which is an HTTP reverse proxy. While TCTP, the TCTP proxy, and the middleware are specific instantiations of blueprint components, RedHat OpenShift and the TRESOR Demo application are selected for illustrative purposes of contemporary PaaS solutions and RESTful applications.1

![Diagram of architecture](image)

**Figure 2. The prototype architecture.**

### 3.3 Architecture Evaluation

As our evaluation with participation of the above mentioned healthcare experts shows, TCTP fulfills all of the requirements as a security technology, as carried out in detail by Slawik (2013). The employed HTTP proxy and the Rack middleware implement all of the functionality of TCTP, consequently meeting the requirements for client & server security technology. This in turn meets the global requirements R1 (end-to-end confidentiality of transmitted health data), R2 (end-to-end integrity of transmitted health data), and, as both software components are deployed by the cloud customers, they are given control over the negotiation of encryption keys (R3).

With regards to R4 (low implementation efforts and performance overheads), we base the evaluation on our experiences and a practical benchmark.

**Implementation effort:** The implementation efforts for similar architecture instantiations include installing the TCTP proxy on the client, and adding the pre-existing TCTP middleware to the Rack configuration file (in case of Ruby based applications), or installing the TCTP proxy on the server (in case of other programming environments). Based on our experience in setting up the benchmark environment, we assume that performing these actions would require an experienced programmer at most one person hour. As the TCTP proxy is implemented using the Ruby programming language, it is compatible to a wide range of operating systems, e.g., Microsoft Windows XP/Vista/7/8, Linux, Mac OS X, Solaris, and FreeBSD.

**Performance overhead:** A synthetic analysis of the performance overhead introduced by TCTP can be found in the paper by Slawik (2013). This analysis hints at an expected performance impact between

---

5% and 10%. For this publication, we wanted to assess the communication overhead in a realistic setting - the interaction of a user agent with the TRESOR demo, hosted on the OpenShift platform. Our benchmark script simulates a common workflow for medical personnel: logging in, getting a list of patients, creating a number of patient records (five in the benchmark), updating those records with medication and illness information, and logging out.

We measure the time it takes for the workflow to be carried out in three settings: direct communication, communication over the proxy, and communication over the proxy using TCTP. This is done to evaluate how the expected variance in performance of a shared cloud environment relates to the overhead introduced by the proxy and TCTP. We averaged the values over ten consecutive repetitions and repeated the whole benchmark six and twelve hours later to get samples from different times of day. The benchmark client is a desktop workstation equipped with an Intel Core 2 Duo E8400 CPU, running Debian Wheezy 64bit. The network at our test lab in Berlin features a 1 GBit/s link to the Internet. Our results are shown in the following table:

<table>
<thead>
<tr>
<th>Run</th>
<th>Direct</th>
<th>Proxy</th>
<th>Overhead vs. direct</th>
<th>Proxy + TCTP</th>
<th>Overhead vs. direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>9.26 s</td>
<td>10.31 s</td>
<td>+11.3%</td>
<td>10.42 s</td>
<td>+12.5%</td>
</tr>
<tr>
<td>2nd</td>
<td>12.53 s</td>
<td>9.56 s</td>
<td>-23.7%</td>
<td>11.57 s</td>
<td>-7.7%</td>
</tr>
<tr>
<td>3rd</td>
<td>14.62 s</td>
<td>10.97 s</td>
<td>-25.0%</td>
<td>10.45 s</td>
<td>-28.5%</td>
</tr>
</tbody>
</table>

Table 1. Mean communication time of medical workflow in relation to access means.

In this realistic benchmark setting, the anticipated performance variations of a shared cloud environment, such as OpenShift, conceal any communication overhead of the proxy and TCTP. In fact, the communication time fluctuates also within the ten repetitions of the workflow, which is exemplified by the next figure, which shows these time variations for two exemplary workflow actions: listing patients and saving a patient record.

![Fluctuations in communication time of two exemplary actions.](image)

Based on our observations we conclude that our application blueprint can be applied in a realistic setting without impacting the performance in a way that would be distinguishable from the usual performance fluctuations of public cloud offerings.

4 Conclusion, Limitations and Future Work

This paper shows how medical software-as-a-service solutions can communicate sensitive medical records between healthcare providers securely using end-to-end encryption, thus meeting German legal requirements. The proposed architecture blueprint is demonstrated utilizing a prototypical
instantiation based on the novel TCTP protocol. In particular, the proposed blueprint guarantees the confidentiality and integrity of medical records transfer to and from the cloud and requires low implementation efforts while producing a relatively small performance overhead. The presented proposal was evaluated in a tight collaboration with German healthcare experts and has been adopted for the TRESOR healthcare cloud ecosystem.

In our research, we observe several challenges to be addressed in the future. While the blueprint presumes that confidential medical records are transmitted as part of HTTP entity-bodies, some information in the HTTP headers could also contain sensitive data, such as patient identifiers in the URLs of GET requests, which would be accessible by cloud computing intermediaries. This challenge can be mitigated by either encrypting those parts of the URLs, by using pseudonymous identifiers (differing patient IDs from user to user), or forming encrypted POST requests containing those identifiers.

The prototype architecture demonstrates that secure communication with RESTful applications deployed in shared environments can be realized. However, the prototype components, especially the TCTP proxy, are still in their early development phases and present a substantial potential for further improvement. A considerable number of additional benchmarks are possible, which can be used to carry out the assessment/refinement process of the design science approach, strengthening the practical applicability of our work.

As almost all SaaS cloud applications implement a RESTful design, we did not have the goal of providing a blueprint for non-RESTful applications. Still, all communication protocols compatible to HTTP can be secured by TCTP, yet there are already existing end-to-end technologies for some protocols, such as WS-Security for SOAP-based solutions. Nevertheless, TCTP could be used in addition, for example, to enable data-flow protection.
References


