PV204 Project Phase II Report

Team members

- Adam Chovanec (485311)
- Alexandre Deneu (556569)
- Jan Sekanina (493085)
- Marc Monfort Muñoz (545819)

Topic

Message board secured by TPM

About TPM

Trusted Platform Module is a trusted entity with the purpose of improving the security of its user. It is also referred to as an international standard. It usually takes on a form of a dedicated micro-device set on the motherboard, however other forms of TPMs are common, such as Integrated TPMs or Virtual and Software TPMs.

It can be used as a secret storage and it provides a true/hardware random number generator. The TPM can be used for remote attestation, absently checking whether the hardware and software setup complies with the according demands. Two versions of TPMs emerged. Version 1.2 and 2.0, which is not backward compatible with the former.

TPM as part of our project

Our goal is to be able to authenticate one or more clients to a server via a client's TPM. And such as that it is a vital element to our project. The server then checks and authorizes the client to add something to the board or take a peek at the board, or manipulate the board in unpredictable and peculiar ways. As far as it is allowed by the server. The communication should be therefore bound to a concrete device via the TPM, and without using a registered device, access to the board shall not be granted.

On the very first interaction a registration of the client should take place. Afterwards the communication between the server and the client should be bidirectionally authenticated, although only clients use TPM. This is done by TPM holding a set of secret keys.

TPM holds two types of keys, Endorsement Keys and Attestation Keys. Attestation Keys are certified by Endorsement Keys and then used to directly sign or encrypt data. Each TPM has its own unique identity defined by a set of Endorsement Keys that are provisioned by its manufacturer. These keys are not used for data encryption. It should be noted that none of the keys ever leave the device.

Architecture and Design Choices

Overview

Our choice of programming language Golang was heavily influenced by the libraries and projects widely working with TPMs being available to significantly simplify the coding efforts. The most notable is “https://github.com/google/go-attestation”, which states that the project tries to provide high level primitives for both client and server logic.

Registration process

The client initiates a new connection to the server and begins the act of registration. The server sends a challenge to the client. If a client succeeds at the challenge, the registration is successfull and the EK's public key is bound to the username.

Authentication

For any subsequent authentication request, the server takes advantage of the EK's public key saved during the registration and uses it to generate a new challenge for the client. The client proves their identity by decrypting the challenge. Subsequent communication is authenticated via HTTP cookies and secured by TLS.

Command-line Interface

A command-line interface (CLI) is a text-based interface that allows users to interact with a computer's operating system by entering commands in the form of text. In our project, the CLI will be used to send and receive messages on a message board.

To implement the client-side functionality of the message board, we will use a CLI that allows users to perform various actions, such as posting messages. The CLI will prompt the user to enter their message, and then send it to the server for posting. To create the CLI, we will use the bufio library. This library is simple to implement and offers a wide range of functionality. In the future, we may also use the CLI to handle authentication and registration. The CLI could also be used to change the destination of a message.

The CLI will provide a simple and efficient way for users to interact with the message board.

HTTP

In our project, the server will use the HTTP protocol to listen for incoming requests from clients and send responses back to the clients. When a client wants to post a message to the message board, it will send an HTTP request to the server using a specific method, such as GET, POST, or PUT. The request may also include additional information, such as headers and a body, which can be used to provide additional context or data for the request.

Upon receiving the request. The server will then send an HTTP response back to the client, indicating whether the request was successful or not.

Overall, the use of HTTP in our message board project will provide a simple and flexible way for the client and server to communicate with each other. It will allow the client to send requests to the server, and the server to send responses back to the client.

Separation into client and server

In our basic implementation, the communication between client and server is based on a secure data exchange, where the client initiates the process by generating and sending its public key and TPM attestation parameters to the server. The server, after receiving this data, generates a cryptographic challenge that only the legitimate client device can solve, thanks to its TPM.

This strategy ensures that each party fulfills its specific role: the client proves its authenticity and the server validates this authenticity before allowing access or interaction.

In our system, we use an HTTP server to handle communications between the client and the server, with data exchange taking place via POST requests. The data, including the client's public key and attestation parameters, is serialized to JSON for sending, and the server deserializes this JSON to process the information received.

This process allows for a structured and efficient data exchange.

The server then responds to the client also sending a generated cryptographic challenge that the client must solve, thus proving the authenticity of its identity and the integrity of its TPM.

Threat Model

The security of the communication stands on several pillars:

- TMP keys on the client (authentication of the client)
- TLS keys on the server (secure channel, authentication of the server)
- TLS certificate signed by CA (secure channel, authentication of the server)
- Secret used for cookies encryption (session post-TMP authentication)

An attacker may compromise the security of the system, if:

- The server’s TLS key is compromised and the attacker is able to read the messages, including the authentication cookies.
- The certificate authority’s TLS keys are compromised – the implications are the same as above
- The cookies for a single session are compromised – an attacker may act in the name of the logged-in user
- Server’s secret for encryption of cookies is compromised – an attacker is able to act in the name of any logged-in user

We don’t consider these risks to be significant enough to make us reconsider our design choices. We believe that similar risks would be present in any system. TLS is used universally on the web and we consider them to be a solid foundation to build on. Cookies are rotated every login, and TLS certificate on the server has a validity for 90 days.

State of the project

We have successfully implemented registration of the client to the server through remote attestation via HTTP protocol. The server does not use TLS yet and data for the registration are stored in-memory instead of pernament storage, such as a database. We have a proof of concept code for the command line interface.

We believe that the hardest part of the project is behind us and the missing features, such as sending and recieving messages, will be much easier to implement.

Individual Contributions

Jan Sekanina

I took part in engineering architecture, doing research and finding appropriate solutions. I set up a basic html server structure, which in the end (I think) served only as an inspiration. I also participated in managing, merging and cleaning the codebase, dividing the workload among the team members and writing the final TPM report article.

Adam Chovanec

I participated in creating the architecture and set up the repository structure and basic instructions on how to use Golang. I have implemented a proof-of-concept of TMP remote attestation. Later I expanded the code to encode the client-server communication in JSON files as a prerequisite for sending them through the Internet.

I have also extracted the common code into libraries, and finished the implementation of registration of a client using HTTP requests and cookies. I have also put together individual contributions into one document.

Alexandre Deneu

During the second phase of the project, I contributed by dividing the code between the client and the server, and I began to create the CLI interface.

Marc Monfort Muñoz

I have contributed to the development of a secure registration and authentication mechanism within our TPM message board project. Drawing inspiration from a proof of concept initially programmed by, Adam, I dedicated myself to expanding upon these ideas.

My primary role involved architecting and implementing a clear separation of responsibilities between client and server components.

This development entailed establishing a robust and functional communication protocol over HTTP, which facilitated efficient data exchange and also granted the standards of data integrity.

Through careful programming and testing, I ensured that our system manages successfully TPM functionalities using its potential, thereby enhancing the security and reliability of our authentication processes. This work presents a good advancement in our project, setting a solid foundation for future expansions and enhancements to the project.