DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Augmenting the MetaMask-Wallet with Domain Name Based Authentication of Ethereum Accounts

Jonas Ebel

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Augmenting the MetaMask-Wallet with Domain Name Based Authentication of Ethereum Accounts

Erweiterung der MetaMask-Wallet um eine auf Domänennamen basierende Authentifizierung von Ethereum Accounts

Author:	Jonas Ebel
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	Ulrich Gallersdörfer, M.Sc.
Submission Date:	15. April 2021

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15. April 2021 Jonas Ebel

Acknowledgments

First and foremost, I want to thank my advisor Ulrich Gallersdörfer. His insightful feedback and knowledgeable support helped me to develop this thesis. He provided an environment of great mutual respect, which allowed me to shape my ideas and reiterate my proposals. In our engaging and effective meetings, he improved my approaches and helped me overcome problems during this work.

Furthermore, I thank Professor Florian Matthes that I could pursue this research work at his chair for Software Engineering for Business Information Systems. His insights about the research idea and the thesis title significantly shaped this thesis.

I want to thank all the participants of the usability study who were willing to contribute their time and thoughts to this research project. Their insights tremendously helped me to understand user behavior and to revise my assumptions.

Finally, I want to thank my family for supporting me during my entire studies and making this research possible in the first place. Their constant support helped me enormously throughout this process. I very much appreciate Edda Stephan's valuable insights and her proofreading of this work.

Thank you.

Abstract

This thesis proposes a concept to augment the wallet application MetaMask with a domain name based authentication. Today, users have to resort to manual methods to assert the receiver's legitimacy of a transaction in Ethereum. These cumbersome approaches expose users to the risk of transferring ether to the wrong address due to individual errors or targeted attacks. We expect the authenticating wallet to enhance the user's security. Our authentication approach adopts the concept *TLS endorsed Smart Contracts* (TeSC), which builds upon the existing TLS/SSL infrastructure. This approach decreases bootstrapping issues because it uses an already existing system of trust propagation. We analyze another TLS/SSL adopter to formulate design principles: the browser, which authenticates website owners with TLS/SSL certificates. Based on this analysis, we propose a design concept for MetaMask to communicate the authentication state. Furthermore, we develop an algorithm to facilitate authentication based on TeSC. We demonstrate that such a concept is technically feasible. The results of a usability study show significant improvement in the user's ability to judge the legitimacy of an Ethereum address.

Contents

1. Introduction

Authentication is the process of verifying an entity's claim of having a particular identity. This process establishes crucial trust when entities interact with each other. Whether a person shares information with a friend, grants other people access to an office building, or purchases groceries at the cashier's desk, the person must recognize (i.e., authenticate) its counterparts as legitimate endpoints for this type of interaction.

In computer networking, nodes authenticate each other to establish trusted communication channels. This endpoint authentication is also relevant in terms of the human-computer interaction: the user needs to assert that the controlling entity on the other side of the channel is legitimate for the interaction scenario. The entity's identity must be available in a human-readable and understandable form to provide the user with the means for this assertion.

The internet developed a solution for authenticating endpoint communication in a user-understandable way in 1999. The mapping of IP addresses to domain names translates the machine-readable address to a word, which users can read and have learned to understand. TLS/SSL certificates bind real identities to these domain names. A hierarchical infrastructure propagates endorsements for these certificates from trusted authorities, which scales the usage of such certificates worldwide. With the domain names, the endorsing certificates, and the hierarchical infrastructure of endorsements, the user can authenticate a website's owner. The browser automates this process and shows the user the authentication information in an understandable form.

The general purpose blockchain platform Ethereum lacks such a mechanism for authenticating transaction receivers. A user has to consult off-chain resources to verify an address's legitimacy. Organizations post their Ethereum address on their website from where the users carefully transfer the information into Ethereum. The lack of authentication exposes the users to resource loss due to human errors, such as spelling mistakes, or attacks on their funds. An infamous example is the hacking of CoindDash's website. The attackers on CoinDash altered the website's content and changed the displayed Ethereum address where users could invest in the enterprise. Thus, the users unknowingly sent ether to the wrong address, which led to a loss of $ 7 Million. [1]

We propose to augment user interfaces of Ethereum with transaction receiver authentication to prevent attacks such as the CoinDash hack. We apply a standard for Ethereum proposed by Gallersdörfer et al., who use certificates for authenticating entities in Ethereum [2]. The authors refer to their proposal as *TLS/SSL-certificate endorsed smart*

contracts (TeSC). As the TLS/SSL certificates originate from the internet, we closely align design decisions with the design principles we find in browsers. The alignment allows us to build upon the existing research body about user interface design for authentication warnings over the last 20 years.

1.1. Problem Statement

We want to increase the security of users when they interact with Ethereum. Our proposed solution not only mitigates issues due to mistyping, but it also establishes safeguards to prevent social engineering attacks with fraudulent Ethereum addresses. Our approach is promising because we rely on the established warning concepts of browsers to sketch our proposition's design. However, there are implications of these decisions, which we need to control.

- **Disputed efficiency of browser warnings.** The research community and developers have questioned the effectiveness of warnings for potential security risks since the browsers introduced them. Studies have observed improvements in the design and learning effects of the users over the years. However, users still fall for phishing attacks, showing that the browsers' security concepts are not perfect. We analyze a diverse set of browsers to collect more concepts and identify a larger set of best practices.
- **Missing authentication methods in Ethereum.** One of the value propositions of cryptocurrencies and platforms, such as Ethereum, is its users' anonymity. Hence, authentication is not available without extra effort. We apply TeSC, which allows owners to voluntarily provide their identity information based on certificates. Using TLS/SSL certificates allows the trust, which users place in HTTPS-based server authentication, to be transferred to the interaction with Ethereum.
- **Low adoption of security protocol.** TeSC is still in a proposed state. There exists no Ethereum address on the main network, which applies the protocol. We have to consider that a security protocol, which is fully supported, has more explanatory power than one with merely no adoption. An address, which does not adhere to the protocol, is dangerous in the former scenario but very common in the latter. Thus, we propose a minimal invasive concept to support early adopters of the protocol. We suggest a heuristic to identify fraudulent addresses, such as in the example of the CoinDash hack. However, we refrain from enforcing protocol compliance for every Ethereum address.

1.2. Research Questions

We pose three research questions (RQ) to structure the overall research endeavor, which we strive to answer with this thesis.

RQ1 How can the indication of domain name-based authentication be designed for MetaMask?

The indication of the authentication state is the driving question for a large part of this thesis. It determines whether the user will be able to understand warnings and act on them accordingly. It is detrimental to draw from existing insights about warning design. Thus, a part of this question requires analyzing the related system of TLS/SSL warnings in browsers. Additionally, it is important to understand when authentication is desirable in the application. Based on these two aspects, the thesis proposes a design concept to communicate domain name-based authentication to the user in MetaMask. The following three sub-questions are part of RQ1:

1.1 How does the browser indicate TLS/SSL in its user interface?

1.2 When can domain name-based authentication be applied in MetaMask?

1.3 How can transaction receiver authentication be communicated in MetaMask?

RQ2 What is a feasible architecture concept to authenticate addresses in MetaMask?

This research question requires delivering an architecture model that incorporates domain-name-based authentication in MetaMask. The thesis leverages TeSC to facilitate the authentication. TeSC allows address owners to store identity proofs based on TLS/SSL certificates in the blockchain. MetaMask needs to verify these proofs. Therefore, this thesis proposes a verification algorithm. Following a prototypical implementation, the thesis evaluates the technical feasibility of this algorithm. Finally, it also discusses the implementation of the proposed design concepts from RQ1. Hence, we want to answer the following sub-questions:

2.1 How do we verify domain name-based identities in MetaMask?

2.2 Is this verification technically feasible?

2.3 How can we implement the design concept in MetaMask?

RQ3 Does the application of domain name-based authentication improve the user's security while interacting with Ethereum?

The thesis evaluates the efficiency of the proposed solution in RQ1 and RQ2. For that reason, we conduct a usability study and observe user behavior during a scenario-based experiment. Besides the general correctness of the implementation, the study investigates the users' ability to interact with the prototype and understand security warnings. We pose the following sub-questions:

3.1 Is the application able to identify transaction receivers?

3.2 Does the proposed design concept successfully warn the user during an attack?

3.3 Is the user able to understand authentication problems?

1.3. Research Contribution

The primary contribution of this thesis is a prototype of MetaMask augmented with domain name-based authentication. The evaluation of this prototype investigates the impact of such authentication on the user's security. This thesis's approach is different from existing endeavors because it builds on the existing authentication infrastructure of the internet and does not require a new system for trust establishment. Namely, we contribute:

- a design concept for communicating authentication states based on a browser analysis,
- a verification algorithm for TeSC, which contemplates its current non-existent adoption,
- an implementation of both the algorithm and the design concept in MetaMask to prove their technical feasibility, and
- the results of a usability study about the prototype, which reports the proposed system's effectiveness.

1.4. Outline

We structure this thesis as follows. In chapter 2, we discuss fundamental concepts relevant for this work. We introduce Ethereum, the TLS/SSL infrastructure in today's internet, and the authentication protocol TeSC. Chapter 3 highlights related works, which discuss similar concepts. This chapter consists of two parts. First, we discuss other authentication approaches in Ethereum, and then we review research about the security indication in browsers. Chapter 4 presents the methods which we employ during this research process. It discusses the design science research framework, which shapes our overall work, and a framework of design concepts used to analyze the browser design. Furthermore, it highlights our evaluation method. We introduce the results of our browser analysis in chapter 5 together with a description of the use cases of authentication in MetaMask. Moreover, a discussion on error scenarios, which may occur during authentication, is also presented. Chapter 6 presents two artifacts of the thesis: the authentication algorithm and the warning design concept. It

also discusses the implementation of these authentication methods in MetaMask. We discuss the evaluation of our prototype in chapter 7. Lastly, chapter 8 summarizes our results regarding the research questions. Furthermore, it discusses the limitations of our approach and open topics which might be interesting to investigate further.

2. Fundamentals

This chapter discusses the fundamental technologies of this thesis. Section 2.1 presents the blockchain Ethereum. Section 2.2 introduces the x509 certificate, which is the format of TLS/SSL certificates. Furthermore, it presents the public key infrastructure, which propagates endorsements from central authorities to end certificates. Section 2.3 introduces the protocol "TLS/SSL endorsed Smart Contracts". This protocol leverages the existing TLS/SSL certificates facilitating identity endorsement in the blockchain.

2.1. Ethereum

Ethereum is a public permissionless blockchain, which is open source. Introduced in 2014 by Buterin, it serves as a global Turing-complete computing machine. The Ethereum Virtual Machine (EVM) executes operations, which developers define in so-called smart contracts. Thus, the platform is not only responsible for tracking transactions of a currency from one address to another, as Bitcoin does, but it is a general-purpose blockchain. We follow Antonopoulos' and Wood's explanations about Ethereum in [3] throughout this section.

The platform has an inbuilt currency, which is called ether. The currency's foremost purpose is to monetize the execution of transactions on Ethereum. Whenever a smart contract is executed, the caller must pay a fee of ether depending on the set of basic operations that the contract uses. The fee incentivizes the nodes of the platform to execute a smart contract operation. Ethereum refers to this fee as "Gas".

Though Ethereum is a platform that consists of a variety of components, we only focus on the most important ones for this thesis. Section 2.1.1 presents the Ethereum Account, section 2.1.2 discusses what role transactions have in Ethereum, section 2.1.3 discusses two standards, which Ethereum supports, and the last section 2.1.4 discusses wallet applications.

2.1.1. The Ethereum Account

Accounts track their current ether balance and some meta-information. An address, which is a 40 character string, identifies each account. Thus, we use the terms address

and account synonymously throughout this thesis. Ethereum recognizes two types of accounts: an externally owned account (EOA) and a contract account.

As the name suggests, an external entity (e.g., end-users) controls an EOA. These owning entities manage their accounts with cryptographic key pairs of a public and a private key. Ethereum generates the address of an EOA based on the public key, and any operation altering the current amount of ether requires a signature, which the public key can validate. Thus, a one-to-one binding exists between key and address. Anyone with access to the private key can create a signed transaction and access the funds in this account.

The contract account contains the code of smart contracts. There is no private key for this type of account to access its ether. Only the contract's code can withdraw its funds. The developer must specify the methods which allow withdrawing ether from this account. Otherwise, no one has access to the account's ether. If funds are sent to a contract's address, the EVM executes the contract's code, which describes what happens with the ether. The address of a contract account is generated based on meta-information about the contract creator. For both types of contracts, the address is unique for the entire blockchain.

2.1.2. Transactions

To alter the ether balances, Ethereum disseminates transactions. A transaction is a message which an EOA signs initially. As every contract execution consumes at least the fee for executing its code on a node, any state change of Ethereum requires a transaction. This state change is caused either by sending ether directly to other accounts or calling contract code. Since a contract account has no private key, it may not initiate a transaction. The message consists of several parameters:

1. **A nonce** is a single value associated with the sending account. It defines the number of contracts this account has created, or in terms of contract accounts, the number of transactions it has sent.
2. **The gas price** allows the sender to define the transaction fee s/he is willing to pay.
3. **The gas limit** prevents executing long-running tasks, which would drain all resources from the sender's account.
4. **The recipient** is the address to which the transaction is sent.
5. **The value** is the amount of ether that the transaction carries. It can be set to zero and cannot be greater than the sender's ether balance for the transaction to succeed.
6. **The data field** is a field with which the sender may specify a smart contract's

method and the input parameters. The field can be empty.

7. **The signature components** of the Elliptic Curve Digital Signature Algorithm prove which EOA has initiated this transaction. With these components, the public key of the originating EOA can be calculated. Thus, no explicit mention of the sender is required in the transaction's data.

2.1.3. Standards for Ethereum

Ethereum defines its features with "Ethereum Improvement Proposals" (EIP). These proposals are open-source internet documents guiding the development of the blockchain. A special type of these EIPs is the so-called "Ethereum Request for Comments" (ERC). An ERC defines standards on the application level instead of the inner mechanics of the blockchain itself. The Ethereum ecosystem agrees upon these standards to simplify application development. We describe, in this section, two existing standards which are relevant for the later implementation. [4]

Fungible Token – ERC-20

Ethereum supports the concept of fungible tokens, which can be traded and exchanged against each other. To standardize these token's contracts, Vogelsteller and Buterin introduced the ERC-20 [5]. The document specifies a set of functions that an ERC-20 token contract needs to support. Its adoption is reasonably high. On November 2020 etherscan.io listed 339,849 token contracts [6]. Owners of ERC-20 tokens may transfer their tokens to others or withdraw tokens from a different account if the owner has approved the withdrawer to access this amount of tokens. This interface allows wallets to display and manage ERC-20 tokens. [5].

Interface Detection – ERC-165

Reitwießner et al. describe a standard that allows detecting the interface that a smart contract implements. According to the authors, it is helpful to inquire whether a contract supports a particular interface. Since there is no native support for such a functionality in the EVM, the standard requires the contracts to faithfully declare which interface they support. The ERC-165 defines a mechanism to recognize interfaces based on an interface identifier. The standard suggests calculating the identifier with an "XOR of all function selectors in the interface" [7]. Contracts, which support this detection mechanism, must have a method called *supportsInterface*. This method accepts an interface identifier as an input parameter. It returns true or false depending on the contract's support for this interface. Applications, which use this standard, have to consider that the mechanism does not prevent contracts from misrepresenting themselves.

2.1.4. Wallets

The end-user interfaces are called wallet in Ethereum. A wallet manages the user's keys and abstracts the tedious cryptographic functionality to sign a transaction. It serves as a "gateway to the Ethereum system." [3] Besides signing transactions, it often provides additional utilities, e.g., calculations of the current conversion rate. Antonopoulos and Wood identify three types of wallet applications: a wallet as a mobile app, a web-based application, and a desktop wallet. Additionally, hardware wallets exist, which store the user's keys on an external device. These devices require an additional user interface of the aforementioned types.

This thesis augments the wallet MetaMask. It is a web-based application, which runs as an extension in several browsers (Firefox, Chrome, Opera, and Brave). MetaMask is open source and available at Github. The Consensys Formation drives the wallet's development; however, volunteers also contribute to the source code. The Ethereum instance is configurable in MetaMask, which supports local testing. This adaptability makes it a good candidate for explorative development efforts, such as in this thesis.

Ethereum's approach of providing a world computing machine allows a great variety of application scenarios. Users pay a fee for each execution, which incentivizes participants to join the network and execute transactions. Developers define the execution logic in smart contracts. Among others, the standards ERC-20 and ERC-165 exist, which define fungible tokens and an interface detection mechanic. In this thesis, we use the wallet application MetaMask, which allows users to transact on Ethereum.

The following section discusses a different topic, namely how TLS/SSL certificates facilitate authentication on the internet. We discuss their structure and how the public key infrastructure disseminates the certificates throughout the network.

2.2. Certificates and the Public Key Infrastructure

The implementation of endpoint authentication in today's browser application will serve as a reference point for our design efforts. Thus, this section provides some background on the fundamental building blocks that enable this functionality. The general goal of browsers is similar to Ethereum account authentication: proving and establishing the association between a network address and a real-world entity.

The public key infrastructure enables the issuance of trusted certificates with identity proofs for specific subjects. This infrastructure emerged to be a scalable approach to distribute identity proofs over the entire internet. We will discuss the infrastructure in the next section. The certificates use a standardized format: x509v3. Section 2.2.2 discusses its structure. When a browser establishes an authenticated connection, it evaluates the certificates to determine whether they can be trusted. This evaluation,

according to the algorithm in the certificate's standard, may produce a variety of errors, which section 2.2.3 discusses. The evaluation happens as part of a greater security protocol: TLS over HTTP, also known as HTTPS. Though we will not discuss the entire protocol, section 2.2.4 compiles the different errors that the protocol recognizes during the validation of a certificate. We conclude this section with a comparison of the two sets of errors. The comparison will help us understand how an application, which utilizes x509 certificates, communicates validation errors beyond the simple adoption of the standard's algorithm.

2.2.1. The Internet's Public Key Infrastructure

The general idea of the X509 public key infrastructure (PKI) defined in ISO/IEC 9594-8 [8] is to create a chain of certificates where the parent certificate signs off on the information of its children. Figure 2.1 shows such a chain with three participants: a root certificate authority (CA), an intermediate authority, and an end entity. All entities possess a cryptographic key pair. The root CA starts the chain and creates a certificate.

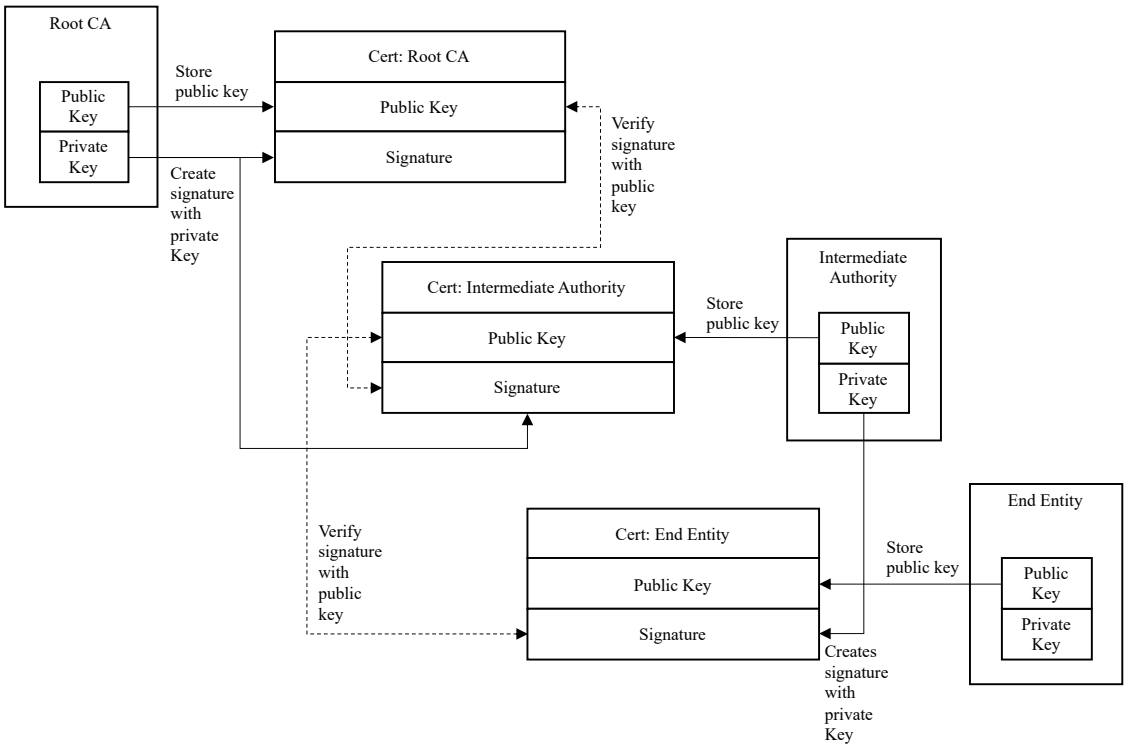

Figure 2.1.: Certificate chaining in X509 PKI with three entities

The CA combines its public key and some identity information (e.g., a common name) to a binary array and creates a signature of this array with its private key. It is the cryptographic assumption that this binary data cannot be changed without the signature

becoming invalid. A signature validation algorithm takes the binary array, the public key, and the signature as input to assert data integrity. The first certificate is called a self-signed certificate because it uses its private key to create the signature. This certificate is now published and establishes a binding of identity information with a cryptographic key pair. Anything else that the root CA signs with its private key can be verified with this certificate because it provides the CA's public key.

If the root CA wants to express that it endorses the identity data in a certificate, it signs the certificate with its private key. Figure 2.1 shows how the root certificate endorses the binding of an intermediate authority's identity with the intermediate authority's public key. Semantically, this binding means that the root CA ensures that the intermediate authority possesses the private key corresponding to the public key and that the identity data is correct. Anything that the intermediate authority signs with its private key can be verified with this certificate plus the root certificate.

This process of signing and verifying repeats for the end entity. The intermediate authority uses its private key to bind the end entity's public key with the entity's identity data in a certificate. To assert the validity of an end certificate, one requires the entire chain of certificates from the end entity up to the root CA.

This chain of certificates evolves into a hierarchical tree in X509 PKI. The root CA establishes several intermediary authorities, which again sign other entity's certificates, and this process repeats as long as the authorities allow this chain of certificates to be.

If a client trusts a root CA's conduct on certificate issuance, this trust propagates through the entire tree down to the end certificates. If one does not trust a root CA, this renders the entire following certificates invalid. While we acknowledge that adopting a PKI requires a more specific concept for trust establishment, it is not the goal of this thesis to evaluate different trust models. We will assume trust propagation in X509 PKI as given. The following section will discuss the internals of the certificates that the public key infrastructure uses.

2.2.2. The X509v3 Certificate

IETF published in 1999 [9] the request for comments (RFC) 3280 that specifies the profile of x509 certificates to use them in the internet public key infrastructure. RFC 5280 [10] is the latest update to this specification published in 2008.

The general structure of an x509 v3 certificate is threefold: the certificate's identity claims, the name of the signature algorithm, and the value of the signature. The signature is created according to the specified algorithm over the entire ASN.1 DER-encoded content of the certificate.

The content itself consists of 10 fields:

- The version,
- a serial number that must be unique for the issuer,
- the signature algorithm identifier,
- the issuer's name,
- the time frame of validity,
- the subject that this certificate belongs to,
- the public key for which the subject possesses a private key as a counterpart,
- the issuer's and subject's optional unique identifiers,
- and the extensions.

The issuer's name, the subject's name, and the certificate extensions have a rather complex structure and are presented in the following discussion.

The name type consists of a sequence of key-value pairs. The RFC 5280 defines a set of standard keys:

- the country,
- the organization,
- the organizational unit,
- the distinguished name qualifier,
- the state,
- the common name,
- and a serial number.

For this thesis, the most important value is the distinguished name qualifier because it can bear the domain that this certificate belongs to. The subject of a signing entity must be the same as the issuer in the signed certificate to create a correct trust chain.

The RFC 5280 also defines a broad set of extensions. We will focus here on a subset of extensions that are relevant for the later discussion.

The *subject alternative name extension* facilitates the inclusion of additional identity information that belongs to the subject. Several options of identity formats are available. Amongst other formats, a domain name can be stored here.

The *key usage extension* restricts the application of the public key in the certificate. It consists of 9 bits that activate different scenarios.

The *extended key usage extension* describes the purpose that the public key may be used

for. This extension must only appear in the end certificate, and each purpose must match the scenarios the basic key usage extension allows. This thesis wants to authenticate server entities. Thus the most relevant purpose, in this thesis, for end certificates is the value *id-kp-serverAuth*.

The *certificate policy extension* describes the circumstances and principles the authority abides by, which has issued this certificate. If the certificate is not an end certificate, this extension limits the policies that might follow in the certificate chain.

The *policy mapping extension* allows certificate authorities to state that they consider two policies to be equal. If a policy mapping from a to b is included in certificate i and if the certificate i or one of its parents includes policy a in its certificate policy extension, all following certificates must state that they abide by policy b.

The following section discusses the errors that RFC 5280 defines.

2.2.3. Errors during X509 Certificate Verification

RFC 5280 provides a path validation algorithm. It checks whether the running agent can establish a trusted path from a given certificate up to a root certificate. During its execution, several errors might occur and render a certificate invalid or untrusted. An application that utilizes this algorithm must communicate these errors to the user. In this section, we discuss all these errors. The table in appendix C shows at which page of the RFC we find each error.

The algorithm builds a trusted path from end certificate to root certificate by verifying for each certificate that the issuer of certificate i is the same as the subject in the preceding certificate i-1. If this is not the case, the algorithm rejects the certificate because it cannot establish a trusted certificate chain.

The basic constraint extension specifies an error maximum path length. The number of hops from this certificate to the end certificate must not exceed this limit. Otherwise, a chain depth error occurs. Self-signed certificates where the issuer and the subject have the same value do not count as a hop in this evaluation.

If the root certificate is not trusted, an untrusted anchor error occurs.

The algorithm takes the expected subject value for the end certificate as input. If the certificate's subject does not match this input value, the algorithm rejects the certificate.

With the name constraint extension, a certificate authority specifies matching rules for all subject names in the following certificates. For example, the owner of *example.com* might only allow subdomains for subjects in certificates that s/he signed: e.g., *foo.example.com*. This constraint applies to the entire subtree of certificates. If one certificate has an unallowed subject name, it renders the entire trust path as invalid.

The algorithm must evaluate whether each certificate along the path matches its signature. If this is not the case, it rejects the certificate.

The validity period of a certificate limits the time it can be used. Thus, the algorithm has to evaluate whether this period has exceeded or not yet begun. A problem with this error is that it assumes synchronized clocks. If the system time of the evaluating machine is not correct, the algorithm falsely rejects the certificate. Thus, an error message must communicate this possibility (see also [11]).

There are several means to revoke a certificate for a variety of reasons. We will not discuss the different approaches here, but we note a general mechanic that can revoke a certificate. A revoked certificate cannot be trusted, and the algorithm rejects it.

A client has to define the set of policies, which it is willing to accept. If a certificate does not match this requirement, the algorithm throws a policy mismatch error occurs.

A certificate authority can establish policy constraints. This extension imposes two different errors. First, it can forbid particular policy mappings. Such a restricted policy mapping applies to all following certificates. Second, the number of certificates in the chain that do not define any policy can be restricted. If a certificate does not meet these constraints, it becomes invalid.

The key usage extension might not match the purposes for which the client wants to use the certificate. A client can accept a certificate only if the key usage extension allows the intended type of application.

If a certificate marks one of its extensions as critical, the client must process it. An unknown extension error occurs if the client does not know the critical extension and cannot process it.

A certificate, which has its version set to three, must include a basic extension. This extension defines whether the owner serves as a certificate authority and signs other certificates. Additionally, it specifies the maximum path length for the following chain of certificates. If the certificate misses the extension, it is not compliant with the specification.

The standard forbids the use of the value *anyPolicy* in the policy mappings. This particular value represents the entire set of all policies. A certificate with such a mapping renders the entire chain invalid.

Though RFC 5280 does not explicitly mention it, we add the general error that a certificate is not compliant. The standard defines the structure and rules for several fields that a certificate must abide by. If this is not the case, the client cannot verify it and has to reject it.

TLS v1.3 applies the validation algorithm of RFC 5280 during its handshake. The following section discusses the errors that the standard of TLS v1.3 describes to occur

during its certificate validations. The discussion provides an understanding of the communication of the low-level validation errors to higher-layered applications.

2.2.4. Errors during TLS Handshake

RFC 8446 defines the protocol of TLS v1.3 that provides security guarantees such as endpoint authentication, channel encryption, and message integrity. For this reason, it leverages the identities provided in X509 v3 certificates that RFC 5280 defines. Part of this protocol is to validate the provided certificate. This section presents all errors that the request for comments specifies in its validation step.

The *bad_certificate* error is thrown for three different reasons:

1. The certificate uses MD5 as a signature algorithm,
2. it uses SHA-1 as a signature algorithm, or
3. it is corrupt in its entirety, e.g., due to a bad signature.

The *certificate_expired* error is thrown if the validity period has expired or has not yet begun.

If the certificate has been revoked, a *certificate_revoked* error occurs.

If the root anchor of the certificate is unknown, an *unknown_ca* error is thrown.

The *unsupported_certificate* error occurs if the trust chain is not acceptable, if the certificate does not fulfill a policy, or if the certificate has not set version three.

The *certificate_unknown* error is a general error. It happens if an unspecified issue causes the certificate to be unacceptable for the TLS handshake.

We see that the standard defines fewer validation errors than RFC 5280. We assume this to be the case because the requirements of error communication are different. The next section compares X509 validation errors with TLS certificate errors and discusses their overlap.

2.2.5. Comparison of Errors during X509 Verification and TLS Handshake

This section assigns the X509 path validation algorithm errors found in RFC 5280 to the TLS v1.3 certificate verification error categories defined in RFC 8446. Based on the description for each error in RFC 8446, we deduce the respective RFC 5280 errors, which cause this error category. Table 2.1 shows this assignment.

This listing provides an understanding of how an application might communicate certificate validation errors. It also points us to an issue, which RFC 5280 does not cover:

RFC 5208	**RFC 8446**
AnyPolicy uncompliance	bad_certificate
Chain broken	bad_certificate
Chain depth error	bad_certificate
Invalid signature	bad_certificate
Key usage mismatch	unsupported_certificate
Name constraint error	bad_certificate
Non-compliance	bad_certificate or certificate_unknown
Policy constraint	bad_certificate
Policy mismatch	bad_certificate
Revoked	certificate_revoked
Unknown extension	unsupported_certificate
Untrusted anchor	unknown_ca
Validity expired	certificate_expired
Version error	unsupported_certificate
Wrong subject	not specified
not specified	bad_certificate (deprecated hashing algorithm)

Table 2.1.: Certificate errors defined in RFC 5280 assigned to certificate validation errors in RFC 8446

deprecated cryptographic methods. RFC 8446 requires rejecting all certificates signed with an MD5 hashing algorithm and with a SHA-1 hashing algorithm. This decision has to be made on an application level, as it highly depends on its security requirements.

This section presented the internet's private key infrastructure, the certificate format X509 v3 used in this infrastructure, and the errors that the certificate path validation algorithm defines. The comparison with a standardized application provides ideas for adoption scenarios. The following section introduces the protocol of TeSC - a security standard that heavily relies on the X509 based public key infrastructure.

2.3. TeSC - TLS/SSL-certificate Endorsed Smart Contracts

To prove the binding between a website and an associated smart contract in Ethereum, Gallersdörfer et al. propose TLS/SSL-certificate endorsed Smart Contracts (TeSC).[2] TeSC leverages the identity information in the TLS/SSL certificates to cryptographically tie an Ethereum address to a domain name. An application of this protocol facilitates the authentication of the owner of an Ethereum address. According to the authors, one of its main advantages is the adoption of the existing TLS/SSL certificates. Thus, the protocol overcomes bootstrapping issues, which other approaches have. Gallersdörfer et al. define four components, which we will discuss in the following section.

The Endorsement

The endorsement is the basic datatype in the protocol. It consists of four claims and a signature. The four claims are:

1. the address of the smart contract which is endorsed,
2. the domain which endorses the smart contract,
3. an expiration date,
4. and flags which alter aspects of the endorsement, e.g., data encryption of the domain claim.

Additional to the claims, the endorsement contains a signature. The endorsers create a hash of a concatenated string of the claims. Then, they use the private key of the endorsing domain's TLS/SSL certificate to sign that hash. This signature proves the binding between the domain name and the Ethereum address because only the owner of the domain name's certificate has access to the private key.

The TLS/SSL-certificate Endorsed Smart Contract

To show that the contract is associated with its owner, the owner stores an endorsement in the smart contract. To be able to store an endorsement and to retrieve it for verification, the smart contract must implement the TeSC interface, which Gallersdörfer defines in his proposed EIP [12]. The interface specifies getter methods for the claims and the signature. It also defines methods that update the claims and the signature.

Throughout this thesis, we define an Ethereum address to be TeSC compliant if it complies with the following two requirements:

1. it must be a smart contract address and
2. it must abide by the TeSC interface.

A TeSC compliant address claims that an endorsement identifies its owner. However, it does not mean that the endorsement's verification results in a valid authentication state.

Off-chain Verifier

Gallersdörfer et al. specify the off-chain verifier as a component, which fetches the domain' certificate and verifies the endorsement's legitimacy. This thesis focuses primarily on that component. We implement a verifier unit for MetaMask to facilitate authenticating a given Ethereum address. Gallersdörfer et al. describe necessary steps, which a verifier has to execute. Based on this description, we propose an algorithm to verify a TeSC endorsement in section 6.2.1.

The TeSC Registry

The TeSC registry is an additional smart contract, which stores TeSC endorsed addresses and their endorsing domain. This registry serves as a centralized storage of all publicly known TeSC-compliant contracts and their domains. The registry allows retrieving all Ethereum addresses for a given domain. A transaction receiver address, which is not TeSC compliant, can be questionable if the TeSC registry contains a binding for the current website's domain to a different contract. Besides detecting possible fraudulent transaction receivers, the verifier requires this information for certain types of endorsements, which allow no other smart contracts with the same endorsing domain to exist. Hermann et al. propose a reference implementation of the TeSC registry, which we leverage in this thesis [13].

This chapter presented the fundamental technologies, which this thesis uses. We introduce the general-purpose blockchain Ethereum and its components. Additionally, we discuss the X509 public key infrastructure and the format of TLS/SSL certificates.

The infrastructure provides a trust propagation mechanism to scale the distribution of the TLS/SSL certificates throughout the internet. Finally, we present the approach of TLS/SSL-certificate endorsed smart contracts. It facilitates authentication based on domain names in Ethereum using the X509 PKI. The next chapter highlights other works, which discuss related topics to this thesis.

3. Related Work

To found this thesis on the current research results, we discuss other works related to this topic in this chapter. Section 3.1 presents other authentication solutions in Ethereum. We discuss existing methods, which support the user to verify an Ethereum address's correctness in section 3.2. Section 3.3 presents research results about the security indication in browsers. We focus on those indications, which the verification of TLS/SSL certificates produces.

3.1. Authentication Solutions in Ethereum

This section discusses different approaches in the realms of authentication in Ethereum. First, we present an overview of on-chain identity management systems. The following section introduces the Ethereum Name Service (ENS). It is an effort to establish a DNS-like mapping of machine-readable identifiers to human-readable names on the ledger. We outline what issues each approach resolves, where their deficiencies are, and how TeSC differentiates from them.

3.1.1. On-chain Identities

We discuss the general concept of decentralized identities and on-chain identities in this section. Therefore, we present a selected set of examples of current projects. The goal is to determine how TeSC and this thesis fit into the evolving research body of decentralized identity. It is relevant to discuss identity management systems because one of the driving questions of this thesis focuses on how Ethereum accounts can be authenticated. Though we do not strive to identify single persons, identity aspects are required to authenticate an address and its owning entity. Thus, to understand the existing knowledge base, an overview of on-chain identification is required.

The research community discusses the concept of decentralized identification (DID) since the advent of Bitcoin. In the light of existing user-centric identity protocols like OpenID Connect or other federation identity concepts, the goal is to overcome the provider lock-in and enhance cross-organizational portability of identity concepts. [14]

The W3C discusses a working draft that shall define a data model for decentralized

identities. It does not define a particular infrastructure but rather "a) the generic syntax for all DIDs, and b) the generic requirements for performing the four basic CRUD operations on the metadata [...] associated with a DID (called the DID document)" [15]. Though this draft does not specify the underlying technology, it mentions the distributed ledger technology as one possible basis. The architecture presumes a "Verifiable Data Registry" where the DIDs are stored. In comparison, TeSC defines the architecture of such a registry, but it leverages the existing hierarchical identification of the public key infrastructure instead of the concept of DID.

Bernabe et al. review privacy-preserving mechanics in blockchain technology and, therefore, also some identity management systems. They conclude that a widespread issue of those solutions is that they do not conform with the General Data Protection Regulation (GDPR), especially in terms of the 'right to be forgotten'. There exist scalability issues due to the high costs of cryptographic on-chain operations. Furthermore, they conclude that users must develop complex mental models, which inhibit the general usability. [16]

This thesis focuses primarily on evaluating the applicability of an identification protocol. Issues such as GDPR-conformance need to be resolved on the protocol level and are therefore not discussed further. As we design an off-chain verification, we follow the arguments in [2] that only a one-time fee needs to be paid by the address controller, who wants to become identifiable. Once the linkage between a real-world entity and a smart contract is established in Ethereum's state, a verifier can retrieve the endorsement information without paying any transaction fee. Thus, the argument of the high costs of cryptographic operations does not apply to TeSC's approach. We attempt to simplify the authentication process so that users do not need to employ a complex mental model when validating a transaction receiver in MetaMask. We evaluate the ability of the users to process authentication information with a usability test. Chapter 7 discusses this experiment.

One example of a decentralized identity implementation, which builds on Ethereum, is uPort. It aims at being a self-administered identity provider that is simple to use. The user manages his/her identity with a mobile app. The blockchain stores the identity proof as a decentralized identifier. The concept is designed to be compliant with GDPR and overcomes the privacy issues mentioned in the paper by Bernabe et al. by encrypting attributes with the user's private key [16]. uPort's goal is to identify persons. It is complicated to identify a consortium or an enterprise with this solution. Thus, the business case of uPort is different from TeSC and this thesis. We aim at identifying Ethereum address owning entities on the same level, on which the TLS/SSL certificates identify webpage owners. This granularity is already trusted by users on today's internet when they send private information like pictures or bank account data to enterprise websites. TeSC is a protocol that authenticates on-chain entities and does not provide means to prove identity in off-chain environments as uPort does. Thus, we argue that we do not compete with solutions like uPort. We acknowledge the approach of uPort,

which allows a granular per person identification, as a veritable goal, which the TeSC protocol cannot provide.

This section discussed the concept of on-chain identification of persons, and it explains the difference between the existing identity concepts and the goals of TeSC. The following section describes an effort at Ethereum to provide a mapping service from readable strings to Ethereum addresses similar to a domain name service.

3.1.2. Ethereum Name Service

The Ethereum name service exists since 2017. It is a decentralized naming system deployed on Ethereum that stores mappings from human-readable names to identifiers like Ethereum addresses or other machine-readable names. The naming scheme is similar to domain names. It is hierarchical, dot-separated, and has two top-level domains: the main domain *.eth* and for testing purposes *.test*.

Any owner of a domain can create and manage subdomains. A registrar contract defines the distribution mechanics of subdomains. The domain owner chooses an appropriate registrar contract and, therefore, defines the allocation of subdomain names. The domain *.eth* implements a first-come, first-served principle. It costs 5$ per year to keep a domain. The owner can transfer the ownership of the ENS names as a non-fungible token to other Ethereum addresses. This transferal establishes a market for domain names. [17]

Additionally, the owner may define a resolver contract. This contract is responsible for mapping the domain name to addresses. ENS is not bound to map only to Ethereum addresses. It also accepts other EIP standardized record types, like cryptocurrency addresses or hash addresses on the Interplanetary File System (IPFS). This broad specification enables a diverse set of use cases. For this thesis, the mapping of domain names to Ethereum addresses is the most relevant one.

ENS' adoption is relatively high in Ethereum. In September 2020, it counted 190,000 registered names with 52,300 owners. 158 different Dapps and wallets support the naming system. For example, MetaMask accepts ENS names as transaction receivers. [18]

We acknowledge the potential of Ethereum Name Service to improve the usability of Ethereum. However, we argue that ENS does not provide trustworthy authentication of Ethereum accounts as the allocation of names does not require any identification. For example, it is not possible in ENS to prove that the owner of *google.eth* is in any way associated with Google. Anyone could have registered for this domain name. We argue that TeSC could complement ENS at this point, as TeSC-compliant contracts can confirm the linkage between the Ethereum Address and an established identity on the internet. The approach of coupling could be part of a follow-up work of this thesis. We provide additional input in section 8.3.

This section has presented an overview of other existing address authentication methods in Ethereum. It discusses their advantages and disadvantages and how TeSC relates to those mechanics. The following section presents some address verification methods, which exist in Ethereum.

3.2. Address Verification in Ethereum

This section discusses different approaches that Ethereum supports to verify the correctness of addresses. These approaches support the user in identifying spelling mistakes or comparing different addresses.

3.2.1. EIP 55 - Checksum for Ethereum Addresses

In 2016 Buterin and Van de Sande introduce the concept of checksums encoded into the Ethereum addresses with EIP-55. The algorithm hashes the address with keccak256 and compares character-wise the original hexadecimal address with the hexadecimal hash. If the original address at index *i* is a letter (i.e., out of [A - E]), then it is capitalized if the value of the hash at *i* is greater than 7. On average, a randomly generated address has 15 letters. The proposal concludes that the probability a mistyped address "will accidentally pass a check, is 0.0247%". [19]

This mechanic allows improving user security. If wallet applications control the checksum, it may prevent errors like accidentally leaving out or adding one character when copy-pasting or manually typing the address. The checksum enhances the probability that an Ethereum address is valid and the transaction is not lost.

One disadvantage is that the proposal's security properties depend on the number of letters in each Ethereum address. As they vary, this concept does not guarantee reliable error detection. Furthermore, the proposed algorithm does not provide the means to verify whether the transaction's recipient is the intended entity. We conclude that the concept provides some security features, which further proposals like the one in this thesis should complement.

The next section discusses a concept to generate more readable addresses.

3.2.2. Vanity Addresses

The idea behind vanity addresses is to predefine a specific subpart of the public address and then search with brute force a public/private key pair, for which the Ethereum address contains the search string. This approach provides a partially readable string.

A fictitious example address would be *0xfeed0A6e52ac5d995c9fAc8A6935FF11D52dEa16*, where the prefix *feed* has been the search string.

The concept is already known from Bitcoin. Garbe et al. discuss vanity addresses as a countermeasure against man-in-the-middle attacks when copying addresses from websites. They show that the software vanitygen can generate a six-character prefixed address in about 200 seconds. [20]

It has to be considered that the longer the search string becomes, the less likely it is to find a matching key pair. The basic cryptographic assumption of blockchains like Ethereum or Bitcoin is that the generation of a key pair based on a given public key is technically not possible. Thus, it is not feasible to generate a key pair for a completely predefined address.

Since anyone can generate an address with a short, readable prefix while longer prefixed addresses are not feasible to calculate, this concept cannot provide reliability.

The following section discusses a library that provides graphical representations of the Ethereum address.

3.2.3. Identicons

One approach of Ethereum to improve the readability of addresses is the introduction of Identicons. The idea is to generate symmetrical three-color pictures based on the address. The symmetry leads the user to recognize faces or objects in those images. The owners of the Ethereum project Blockies hope to improve the comparability between different addresses with that illusion. [21]

Figure 3.1 shows an example of such an Identicon. A comparison with figure 3.2, an Identicon for an Ethereum address that differs only in the last character, shows their difference.

Figure 3.1.: Identicon of 0xdc51Bac25e1c22E2F04bAAc20396D99fe56f7359

Figure 3.2.: Identicon of 0xdc51Bac25e1c22E2F04bAAc20396D99fe56f7350

Though this might improve the manual comparison between different Ethereum accounts, it cannot contribute reliably to user's security. Identicons require the user to

memorize the picture for each address. Consequently, the security would be entirely dependent on the user's ability and motivation to remember the correct picture. Additionally, the concept excludes any person with color vision deficiencies. According to Salih, vision impairment is observed for 15% of some male populations [22]. This proportion results in a considerable part of the user base being excluded from this security concept. This issue of accessibility reduces the effectiveness of such a concept. Finally, Identicons do not help the user to determine whether a trusted entity owns the address. Hence, Identicons have the same deficiency that we already criticized for other approaches. Identicons do not provide address authentication.

This section discussed three approaches, which Ethereum supports to help the user with verifying an address. We maintain that none of these approaches prevent all addressing errors reliably. All of them cover only certain pitfalls. We expect that TeSC can complement these approaches and will enhance user's security further. The next section discusses the research, which evaluates internet browsers' design choices during the adoption of TLS based hostname verification over the last twenty years. This discussion yields design rationales on which we will build the indication and warning design to communicate the verification state of TeSC to the user.

3.3. Security Indicator and Warning Design in Browsers

Since the introduction of TLS 1.0 in 1999 [23] and HTTP over TLS in 2000 [24], browsers try to inform their users about the current security state of the visited webpage. Researchers have invested effort in developing an understandable and practical design of active and passive security indicators. Passive indicators are only displayed in the browser's chrome at the top or the bottom. Active warnings block the page and display their warnings in the content area. The following gives an overview of different seminal papers that contribute insights on design principles and uncover ineffective concepts.

One of the earliest research efforts is a controlled group experiment with 22 participants by Dhamija et al. in 2006 [25]. They ask participants to identify fraudulent pages and question the reasoning behind their decision. The results show that more than 20% are not considering the indicators. In 40% of the cases, the testers make a wrong classification. Dhamija et al. interpret the results as a best-case behavior because they prime the users to identify fraudulent pages.

Several other studies in the late 2000s employ different survey designs or focus on different aspects of the security indicators. However, all results show the browser design's inefficiency in communicating the security state: the users do not notice the passive indicators (e.g., [26]). A high false-positive rate of active warning messages leads to the users' habituation and eventually ignorance of actual errors [27]. Additionally, the users cannot comprehend the situation and cannot make an informed decision [28], [29].

It is important to understand all findings in light of HTTPS adoption rates. One early large-scale HTTPS study is from 2010 by Holz et al. [30]. They monitor web traffic in nodes distributed over the world. Additionally, they request certificates from Alexa's top pages and analyze them in terms of their validity. Alexa is an Amazon service, which provides data about web traffic.[1] Holz et al. do not paint a complete picture of SSL/TLS adoption on the internet at that time. Nevertheless, it certainly shows that most of their analyzed hosts fail to provide valid certificates. According to Holz et al., just 18% of all TLS certificates would have been accepted without a warning if the certificate verifier uses the Mozilla Root Store. This performance explains the early findings of habituation. A high warning rate leaves the user in a constant emergency, which eventually becomes the status quo.

With Akhawe et al., TLS error design research started to leverage field studies instead of laboratory experiments in 2013. Though the results are still not satisfying, the users perform better than in most previous laboratory experiments. The authors argue that, besides the browser's improved design, one of the reasons might be a laboratory bias in the previous studies. [31]

At the same time, Google and Mozilla deploy telemetry data analytics in their browsers. Their analysis is publicly available, e.g., at the Google Transparency Report [32]. This data provides additional insights for researchers. Felt et al. discuss the HTTPS adoption based on the telemetry reports. They analyze HTTPS adoption from three different angles: monitoring servers, network communication, and page loads in browsers. Each angle shows different figures, but overall they observe an increase in HTTPS support. E.g., the share of bytes sent with HTTPS over the network increased from 20% in 2014 to nearly 40% in 2017. [33] With the rising adoption, the researchers observe improvements in indicator design as the user's adherence to warnings improves.

In 2018 Reeder et al. argue that the browser's indicator is now in a state where the problems of security indication and warning design cannot be attributed anymore to a small set of issues, which hinder a large part of users from browsing safely. They base their argument on a large-scale in-situ field study of Google Chrome users. With a survey posted after encountering a security warning, they ask the participants for their reasoning to bypass a warning or not. The results show a variety of arguments for both cases. The authors suggest that the different contexts the users are browsing in explain the various reasons. Thus, a more context-centered error design should be investigated. [34]

Acer et al. classify the cause of HTTPS errors that a sample of Google Chrome users encountered. They identify the general categories of server-side, network-based, and client-side causes and further distinguish them within each category. [11]

Thompson et al. strengthen the importance of active negative warnings. They examine

[1] https://www.alexa.com/

Google Chrome's indication of EV-certificates and show that users cannot recognize spoofed certificates by manually reading the certificate's information next to the URL and evaluating its legitimacy. They argue that this task requires too much computational load of the user, so most of the time, information is disregarded. Instead, the browser should display an active negative warning in case of EV-Certificate errors. [35]

The designs have changed a lot over the last 15 years, and the browser tried different approaches. In 2020 Jelovčan et al. review literature on security indicators and show that researchers still agree that passive indicators are not effective means to stop users from interacting with fraudulent pages. However, there are findings that these indicators can be an efficient part of a greater security design concept. [36]

Some researchers focus primarily on the warning's comprehensibility. Bravo-Lillo et al. observe that advanced users try to evaluate a priori the risk of clicking through a warning, while novice users cannot evaluate the risk, as they do not understand the warning [29]. Felt et al. distinguish between a user who adheres to a warning and a user who comprehends a warning. They hypothesize that a higher rate of comprehension improves the user's security. Though they can only improve adherence rates with their proposed texts being "simple, non-technical, brief and specific", the result still highlights the importance of an understandable error text. [37] Stojmenović et al. evaluate Google Chrome using a within-subject study, where participants identify phishing websites. The users improve their performance significantly after an educative session on certificate errors, eventually reaching a 100% success rate. An additional Cognitive Walkthrough shows that Chrome's error messages are not understandable for most users because the terms are too technical. [38] Yi et al. support this claim with their interview-based evaluation of warnings in Google Chrome. Participants are not able to understand them because the texts were too technical or did not explain the risk well enough.[39]

We conclude from these findings to focus on active negative warnings with understandable explanations, interrupting the user if a valid error scenario occurs. We acknowledge the inefficiency of passive warnings. However, we suggest indicating with a passive symbol if a contract does not adhere to the protocol. Due to the low adoption rate of TeSC, a passive indication avoids habituation to high-frequent warnings (see for a more detailed discussion section 5.3).

Today's browser's TLS warning management results from the research efforts over the last 15 years. Though it is not yet in a perfect state, it has undoubtedly improved its efficiency. Thus, we will base the design of warnings and security indicators on existing browsers. For this reason, section 5.1 contributes a comparison of different browsers and the main attributes of their error display.

This chapter discussed several authentication and address verification solutions, which already exist in Ethereum. We argue that none of the existing approaches support a completely secure address legitimization of Ethereum accounts. Thus, we hypothesize that TeSC complements existing security features and will enhance the user's security.

This chapter also highlights the current research outcomes about browser UIs. These results contribute to the warning concepts this thesis proposes. The next chapter presents methods, which we employ in this thesis.

4. Method

This chapter presents important methods, which we employ during this thesis. Section 4.1 presents the Design Science Research framework shaping our entire research process. We discuss a framework for conceptualizing designs of interactive systems in section 4.2. Section 4.3 introduces the methods, which we employ to evaluate our design proposals.

4.1. Design Science Research

Researchers in information systems strive to develop scientific constructs about utilizing information technology that serves humans and their organizations in a useful manner. Hevner et al. develop a research framework to ensure a rigorous scientific process to achieve this goal: the Design Science Research (DSR) framework. The core of it is an iterative process of designing artifacts and evaluating them. The process resides on two pillars: the contextual environment and the knowledge base. The contextual environment provides requirements and field testing opportunities to ensure the relevance of the artifact. The knowledge base contributes the necessary grounding of the process and establishes rigorousness. Hevner et al. conclude with seven guidelines, which support the execution of the framework. [40], [41]

DSR has drawn much attention in the research community as it helps to integrate the creative and unstructured process of designing artifacts with the systematic approaches, which scientific research requires.[42]

Throughout the entire thesis, we apply the research framework of DSR. We discuss here the implementation of Hevner's guidelines.

1. **Design as an artifact -** We develop three viable IT artifacts: the design model for TeSC's state indication, a validation algorithm for endorsements, and the instantiation of both artifacts in MetaMask.
2. **Problem Relevance -** Users of MetaMask have no methods to verify the authenticity of Ethereum accounts. Section 3.1 discusses several existing methods, which try to solve parts of this issue, but there exists no holistic approach for authenticating Ethereum addresses. The TeSC protocol provides the technical means for a solution, but it has not been applied in a user-facing product yet.
3. **Design Evaluation -** To evaluate the artifacts, we instantiate the design model

and the verification algorithm with the extension of MetaMask. This extension is thoroughly tested in regards to its performance to warn users of attacks during an experiment.

4. **Research Contributions -** Our contribution to the knowledge base is the design artifact itself and the evaluation of its performance in attack prevention. It responds to the heretofore unanswered question, whether the application of the TeSC protocol in a user interface is feasible and desirable.

5. **Research Rigor -** Guideline 5 requires a rigorous research approach. During the construction phase, we follow the requirements elicitation techniques of Wiegers and Beatty [43] and we leverage the well-established design patterns of Erich et al. [44] for the architecture design. The evaluation of the artifact follows the discussion on the significance of usability studies by Sauro et al. [45] (see also section 4.3).

6. **Design as a Search Process -** In terms of solution space, we acknowledge the indefinite size of it. Following Hevner et al., the existing constraints and laws in the environment restrict this project's search process. These are the already existing design decisions in MetaMask, the boundaries, which browsers enforce on web extensions, or the rules the internet's PKI imposes. The process searches for a satisfactory solution, which "works well for the specified class of problems". [40]

7. **Communication of Research -** We communicate our research results most and foremost in the context of this thesis and during the chair's seminars. We present our progress and results to members of the knowledge base, hereby optimizing and reiterating our approaches.

The subsequent parts of this chapter discuss methods we employ for particular tasks during this project.

4.2. Design Conceptualization

To compare and apply the existing design concepts of certificate validation state communication in browsers, we need to identify each state's conceptual model. This section discusses the framework of conceptual design for interactive systems by Parush [46]. The framework provides a reproducible methodology for the analysis in the later chapter 5.1.

In general, a conceptual model is an information model with a focus on the business objectives. It often uses a domain-specific language and can be located during the early analysis tasks of general-purpose development processes, as it helps to abstract the fundamental concepts of the domain. [47]

The framework of conceptual design for interactive systems by Parush formalizes an iterative design process starting from highly abstracted conceptual models to the detailed user interface design of an application [46]. Metaphorically, he describes the conceptual model as an architectural plan, where each function has its "place" and the user has to navigate from one place to another via defined "routes". The framework develops these places and routes in an iterative process with five layers:

- The first level has a **functional** focus on the general objectives. It describes groups of tasks or objects that help the user to achieve a certain goal.
- The **configuration** defines the places for the functional groups of the first level and their relation to each other. It does not define the physical placement of a task. It describes the logical order of places, where to perform tasks, and from where each place is available.
- The next level defines the **navigation and policy** required to move between the physical elements, which contain one or more conceptual places specified in the previous level.
- The following two levels **form and details** provide a detailed user interface model. The form is the last step before the complete user interface is defined.

Parush uses this framework in a reversed order to identify concepts encoded into an existing user interface. We follow this approach to conceptualize the error communication of browsers in section 5.1. The following section discusses methods, which we employ to validate the instantiated IT artifact.

4.3. Artifact Evaluation

Part of this research project is a study, which evaluates the proposed design solutions and their implementation. While chapter 7 describes the experiment's conduct, this section presents the general principles of the employed study design and the statistical methods to evaluate its outcome.

4.3.1. Study Design

We follow the well-known principles for user research, which Nielsen establishes in 2010 [48]. He highlights several aspects to be considered when building a relevant sample of test users.

Next to a balanced sample in terms of age distribution or educational background, it is especially relevant in usability research to regard the participants' prior experience. Whether the participant is an expert or novice user highly influences the results of the

study. We want to control this influence by inviting an equal proportion of experts and novices. We identify a novice user as someone who has not been engaged with blockchain or cryptocurrency topics previously.

When comparing several design suggestions, Nielsen discusses the options to measure between subjects and within-subjects. He remarks on the advantage of controlling the individual variability of a subject with an intra-subject measurement. When choosing this design, each participant is exposed to all of the different applications. If we want to compare the security performance of a TeSC augmented MetaMask with the original MetaMask, all participants must interact with both applications. We compare the performance by measuring the different behavior per application for each individual. Nielsen highlights that this approach results in the user not being a novice anymore for the second treatment. We follow his suggestion to split the entire sample into two groups. The groups differ in the order of the treatments. This approach shall control the influence of the order on the study's results.

While the experiment must consist of clearly defined tasks from the experimenters' point of view, the user should have an impression of realism for the overall experiment, according to Nielsen. Realism facilitates a better understanding of the tasks and enhances the external validity of the results. We follow Nielsen's suggestion of a scenario-based experiment. We present the scenario of the experiment for this thesis in section 7.2.

To collect data during the study, we closely monitor the participants during the execution of their tasks. We also employ Nielsen's method of *thinking aloud*: we ask the participants to verbalize their reasoning and thinking to help us understand their perception of the application.

4.3.2. Statistical Methods

We discuss the methods to analyze the study results with statistical significance testing in this section. We present the McNemar test to compare the security performance of MetaMask with the augmented MetaMask. Moreover, we apply the adjusted Wald test to estimate the conversion rate of the prototype and the System Usability Scale (SUS) to evaluate the overall usability performance of MetaMask.

The McNemar test evaluates the significance of the difference between two bivariate binary variables. It takes the number of discordant pairs of measurements into account. Discordant pairs are those responses where a participant answers with "yes" for one variable and "no" for the other. Table 4.1 shows the schema of a 2 x 2 contingency table for a bivariate binary measurement. Discordant pairs are the measurements in cells b and c.

The McNemar test asks whether the proportion between the discordant pairs "is greater than what we'd expect from chance alone." [45, p. 83] This results in nonparametric

binomial testing, which is also known as a sign test. Equation 4.1 shows the probability formula. We set $p = 0.5$ and calculate the p-value as the cumulative distribution from 0 to the smaller number of b or c.

$$p(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x} \tag{4.1}$$

To estimate the confidence interval for the probability that a user can detect an attack by using the augmented MetaMask, we follow Sauro and Lewis [45] and use an adjusted Wald method. Agresti and Coull show that the known Wald method performs poorly if the true probability is close to 0 or 1 and if the number of participants is rather low. They suggest an adjusted measurement, which performs as well as Wilson's score confidence interval. Both overcome the original Wald method's limitations. The p-value is calculated with equation in 4.2, and 4.3.

$$\hat{p}_{adj} \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{\hat{p}_{adj}(1-\hat{p}_{adj})}{n_{adj}}} \tag{4.2}$$

$$\text{with } \hat{p}_{adj} = \frac{x + \frac{z^2}{2}}{n + z^2}, \text{ and } n_{adj} = n + z^2 \tag{4.3}$$

To evaluate the overall system's usability, we use the System Usability Scale (SUS) by Brooke [49]. It is a questionnaire with ten items that the participants have to rate on a five-point Likert scale from *strongly disagree* to *strongly agree*. The questionnaire asks about the perceived usability. The respondents should answer it immediately after exposure to the system. The items alternate between a positive and a negative wording, where the odd-numbered items are positive. Brooke calculates the overall score based on the answers. The calculation requires coding the answers according to the wording of the question. Thus, the odd-numbered items (i.e., positively worded) contribute their scale rating minus one to the score. The evenly numbered items contribute five minus their scale rating. These numbers are added and multiplied by 2.5 so that the overall

	Variable 1: yes	**Variable 1: no**
Variable 2: yes	a	b
Variable 2: no	c	d

Table 4.1.: Contingency table for McNemar testing

scale ranges between 0 and 100. Equation 4.4 formalizes the calculation.

$$SUS = \sum_{1}^{10} x_i * 2.5$$
$$\text{with } x_i = \begin{cases} r_i - 1, & \text{if i is odd} \\ 5 - r_i, & \text{otherwise} \end{cases} \tag{4.4}$$
$$\text{with } r_i \text{ being the rating for item i.}$$

We compare our study results with a general reference value because we have not found any previous usability examination of MetaMask with the SUS. Sauro [50] builds a benchmark based on 446 studies using the SUS. He classifies the reported scales based on the measured type of interface. The relevant category for MetaMask is the B2C category. It comprises "public-facing mass-market consumer software such as office applications, graphics applications, and personal finance software." [45, p. 205] The calculated mean value of this category is 74.0. We test the significant difference between our mean and the reference value with the t-distribution and the following null hypothesis: $H_0 : mean(x) - \mu \leq 0$ as Sauro and Lewis suggest [45].

The evaluation of the proposed design concepts applies the statistical methods based on a usability experiment, which aligns with Nielsen's recommendations. Chapter 7 discusses this evaluation.

We employ all methods, which this chapter has presented, throughout our research process. The next chapter applies Parush's method of conceptual design on different browsers to identify design concepts. Furthermore, it provides an analysis of the use cases of authentication in MetaMask.

5. Analysis

This chapter forms the foundations for the design concept and implementation of our proposed solution. For that reason, we analyze the TLS indications in browsers to conceptualize their design model in section 5.1. Our prototype will adopt several aspects of their concepts. Following the browser analysis, section 5.2 examines the different use cases of TeSC in MetaMask. It discusses the flow of events, which guides the users through the wallet application and its implications on a TeSC verification. Section 5.3 evaluates which errors can occur during the authentication algorithm of TeSC.

5.1. Browser Warnings and Security Indicator Designs

The following section discusses how the user interface of different browsers communicates the certificate validation state to the user. We evaluate Mozilla Firefox, Google Chrome, and Microsoft Edge. Figure 5.1 shows that these are three of the four most used browsers in Germany, with Chrome having the most extensive adoption by far. Due to technical limitations, we were not able to include Apple's Safari in our evaluation.

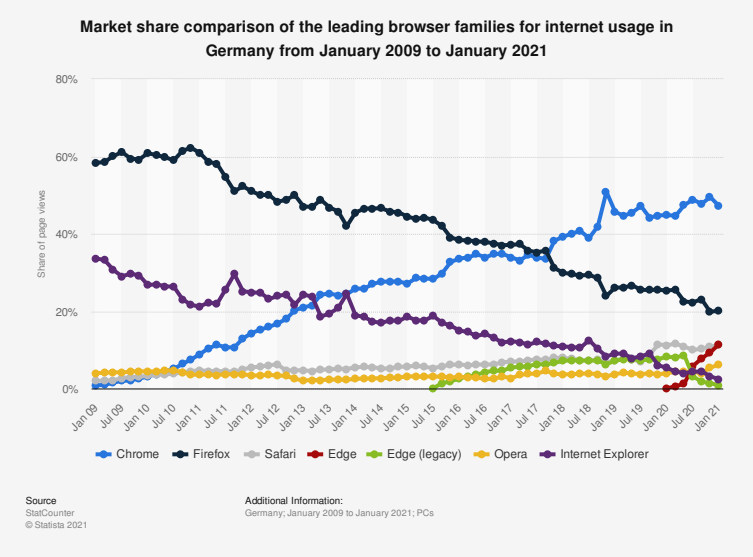

Figure 5.1.: Market share of browsers in Germany from 2009 until 2021 derived from [51]

At first, we present the environment, which we employ to analyze the browser in different validation states. Then, we examine a valid HTTPS-secured web page. The examination provides an understanding of the browser's behavior when no error occurs. The following two sections explore the behavior in an error case for Firefox and the two Chromium-based browsers, Edge and Chrome. This evaluation relies on the elaborated list of certificate validation errors that RFC 5280 defines (see also 2.2.3). However, we have to exclude the policy mismatch error from our analysis. Due to the undefined policy requirements of a general-purpose browser, we could not induce such an error. The last section evaluates the passive indication if the website does not support HTTPS.

5.1.1. Browser Analysis Environment

In order to analyze the browser during different states, we need to induce different certificate validation states. We investigate all errors we have found in the compiled list in table 2.1 and positive indications. We induce these states with two methods. The Chromium project provides a testing page at *badssl.com* that triggers several browser errors [52]. This web page has already been used in other research undertakings (e.g. [53]–[55]). The page covers the error states *Revoked*, *Untrusted anchor*, *Validity expired*, and *Wrong subject*. For the remaining states, we generate certificates manually with OpenSSL [56]. We add a self-signed certificate as a certificate authority to the browsers under test. We use this certificate for signing the generated end certificates. Nginx serves pages via HTTPS, and entries in the file */etc/hosts* reroute requests to the full qualified domain of the certificates to localhost.

5.1.2. Positive Indication in the Browser

According to RFC 8446 [57], the browser can trust the server's certificate after a successful HTTPS message exchange. Otherwise, an error would have occurred. The browser indicates this positive, secure state to the user. We evaluate this indication with two different websites. One is serving a valid certificate with no additional identity information; the other shows an extended validation (EV) certificate. Badssl hosts both of these websites[1].

When the connection is secure, all three browsers show a lock icon next to the web address. The design does not show many differences between Firefox, Edge, and Chrome. The ancillary shield icon in Firefox provides a privacy utility that is not related to HTTPS and X509 certificates. Additionally, the user interface highlights certain parts of the address. Firefox prints the full address, including the HTTPS protocol identifier. However, it highlights only the root part of the domain name. The rest is written in a grey color. Chrome omits the protocol identifier and shows only the domain. The

[1] https://sha256.badssl.com/ and https://extended-validation.badssl.com/

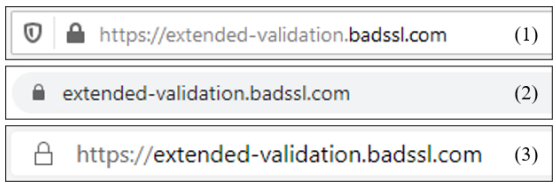

Figure 5.2.: Address bar of Firefox (1), Chrome (2), and Edge (3))

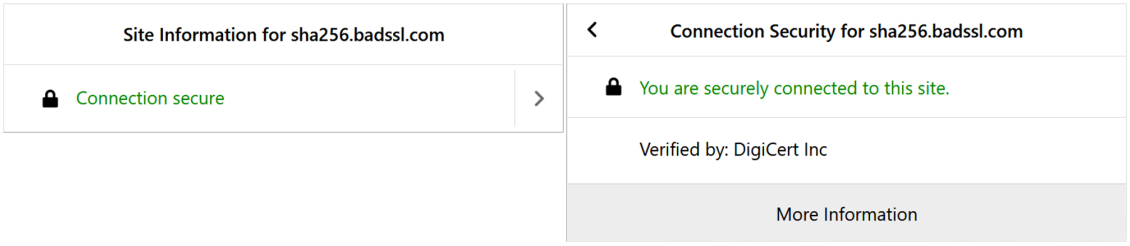

Figure 5.3.: The two pages of the Firefox popup

domain has no special highlighting. Edge displays the complete address, including the protocol. Nevertheless, it writes the entire domain name in black, whereas the protocol part is less visible in grey. Figure 5.2 shows the address bar of all browsers.

The user can click on the lock symbol, and a popup shows up for all three browsers. The popups state with a green text that the connection is secure. In Chrome and Edge, the user can inspect the certificate in a system-specific tool when s/he clicks on the popup's certificate entry. Firefox shows a second page when the users press the arrow symbol. It mentions the certificate issuer's name, and the user can open the browser's tool to inspect the certificate. Figure 5.3 and figure 5.4 show the screens of Firefox, and Chrome and Edge, respectively.

A special type of certificate is one that guarantees extended validation of the certificate's subject information. The interface of the browser does not change if the website uses such a certificate. However, all browsers display the name of the issuer in the popup. Additionally, Firefox shows on the second page of its popup more information about the issuer. Appendix D provides figures of the popups with an EV certificate.

The following two sections evaluate the different screens, which the browsers show on erroneous cases of certificate validation.

5.1.3. Negative Indication in Firefox

This section evaluates how Firefox communicates an untrusted certificate to the user. For this reason, we induce each of the errors defined in section 2.2.3. For each error, we discuss its indication with one exception. We have to exclude *policy constraint* errors as Firefox's certificate validation algorithm does not evaluate the certificate extension (see

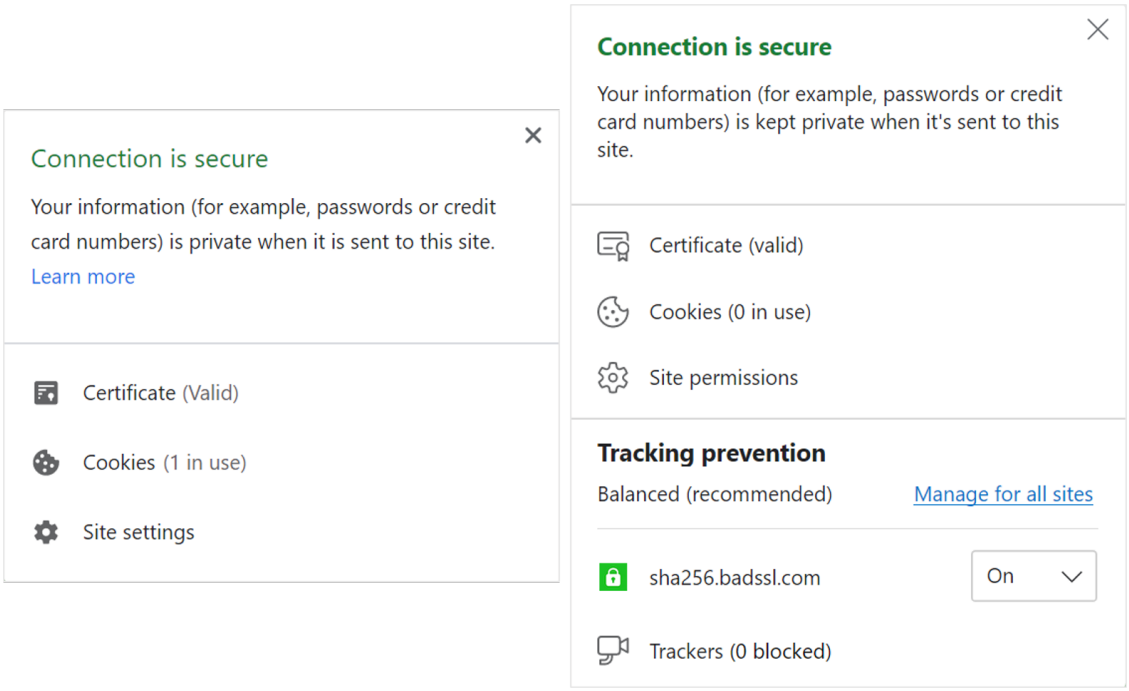

Figure 5.4.: The popup of Chrome and Edge

also the bug report in Bugzilla [58]). We categorize the screens into different groups, depending on visual attributes and the semantics of the error text.

We identify the following attributes, which an error page in Firefox might have:

- A yellow frame around the entire screen,
- a two-stages model: pressing the *advanced* button allows to access the second stage with additional information and enables a button to continue to the page ignoring the error,
- a headline stating that a security risk is ahead or that a secure connection failed,
- a statement that the server might cause this issue (e.g. "website is misconfigured"),
- a statement how the client might cause an error,
- scenarios of what could be happening if the user continues to the page,
- an explanation of what the user can do to resolve the issue, and
- an explanation about what causes the issue.

The table in appendix E defines for each page whether it shows these attributes. Based on this table, we identify two different types of error pages in Firefox.

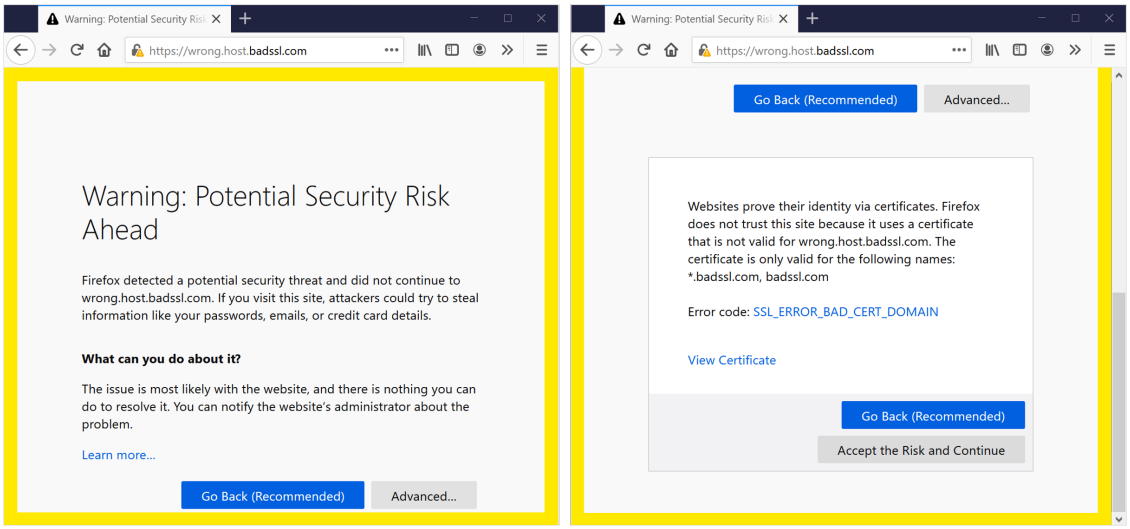

Figure 5.5.: An overridable exception in Firefox

We define the first type as an overridable error page. An example can be seen in figure 5.5. This type has two stages. The first stage informs about a potential security risk. If the user accesses the second stage with the *advanced* button, an explanation of the root cause of the error becomes available. Depending on the error, the user can resolve some of the errors that are causing this page to show up. If that is the case, the second stage displays an explanatory sentence. The user can always ignore the error and continue to the page. A yellow frame highlights the warning visually on both stages.

To understand the user interface concept on an abstract level, we develop a conceptual model following the framework for interactive systems by Parush [46]. We display in this thesis the model of Parush's iterative process halted at the configuration level. Figure 5.6 shows our result for Firefox. It shows that this error type has a two-stages model. These stages distinguish this error type from the critical error type, which we discuss later. The first stage interrupts the user from executing an insecure interaction and warns about the potential security risk. The stage provides an early statement on what the user may do about the situation. S/he may also access more information on an explanatory website about Firefox's security warnings in general. The second stage explains the technical issue in more detail. If the user accesses this stage, s/he can ignore the error and continue to the erroneous page. It is remarkable that on each stage, the user can return to a previous safe page. In contrast, the browser places the potentially dangerous action to ignore the error less conveniently on the second stage. We interpret this design as an endeavor to nudge the user towards a safer behavior. The HTTPS indicator next to the address bar provides the functional task of *accessing site information*.

Firefox displays the second error type on critical failures. Figure 5.7 shows Firefox with a website serving a certificate that has been revoked. This type of screen has no yellow border and only one stage. The user cannot resolve or ignore this issue. Thus,

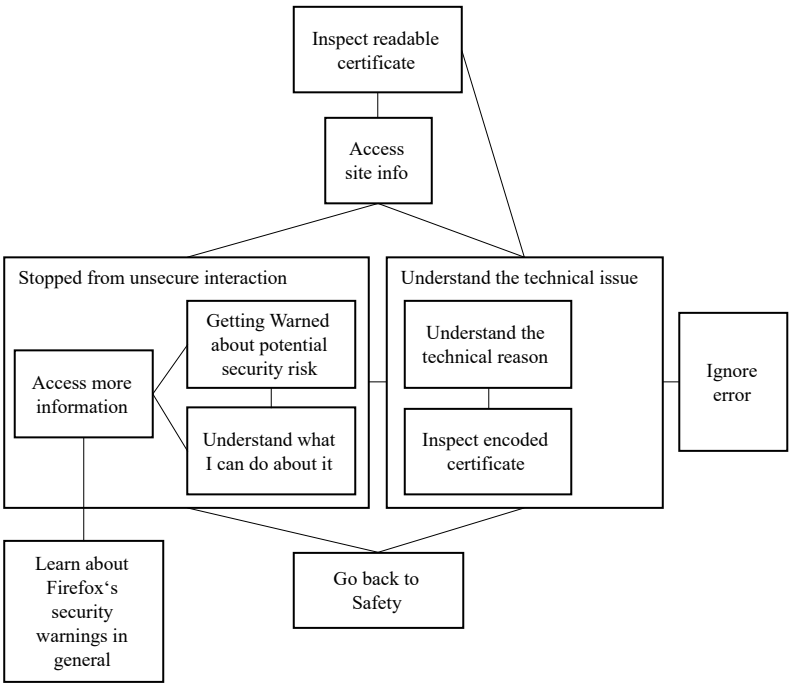

Figure 5.6.: The conceptual model of the overridable error page in Firefox

the explanation of a root cause is reduced to one sentence with a remark that s/he may contact the website owner for resolution. The most apparent difference to the overridable error is the missing yellow frame. Other differences become apparent when we compare the conceptual models.

Figure 5.8 shows the conceptual model for critical HTTPS errors in Firefox. The page interrupts the user with a single-stage design. The user cannot ignore this error. S/he can only try to load the page again, but there is no option to override the error. The interrupting page misses a second stage, and therefore the explanation of the technical reason for the exception is on the first stage.

Both page types contain a link, which references an external educational page that the Mozilla Foundation maintains[2]. This page provides the user with additional background information on certificates, and it explains which different types of errors might have caused the interrupting page to show up.

Additionally, the indication on the address bar changes. Here exist also two types: one for overridable errors and one for critical failures. If the user may circumvent the issue, a yellow warning partially sign hides the lock symbol. The highlighting of the domain name in the address bar is not different from the positive indication. When the popup opens, a red text states, "Connection not secure." Suppose there is a critical failure, the

[2]support.mozilla.org/en-US/kb/secure-connection-failed-firefox-did-not-connect

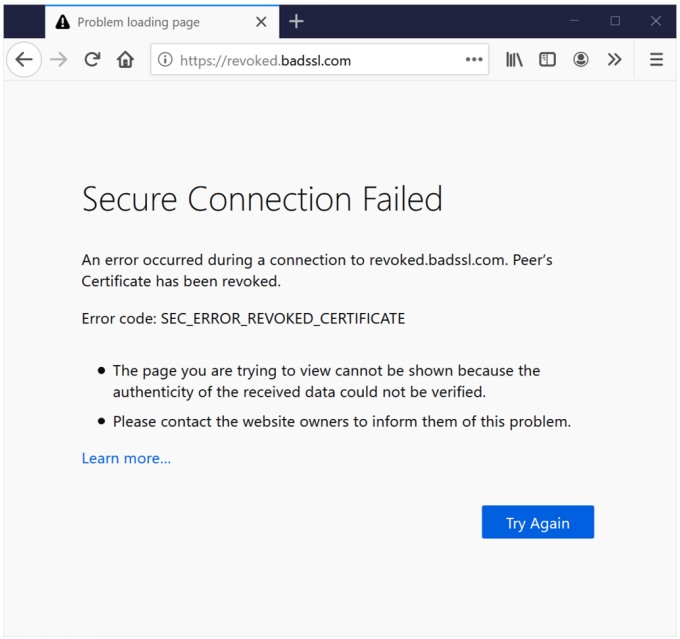

Figure 5.7.: A critical error in Firefox

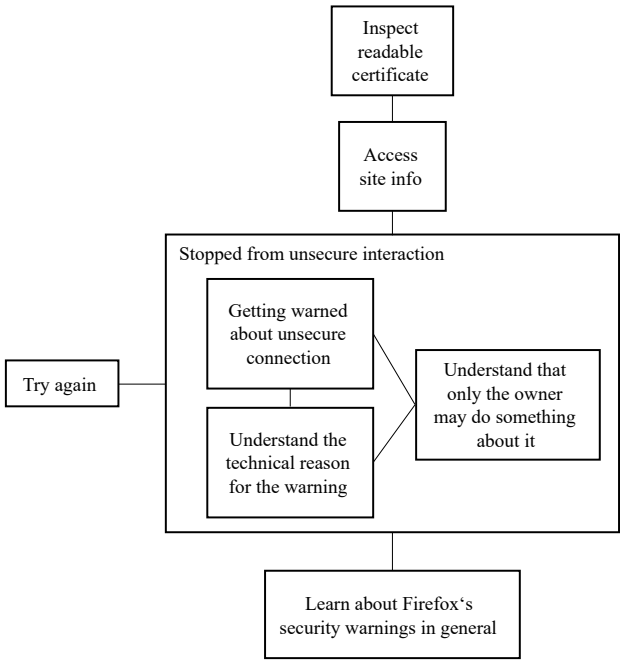

Figure 5.8.: The conceptual model of a critical error in Firefox

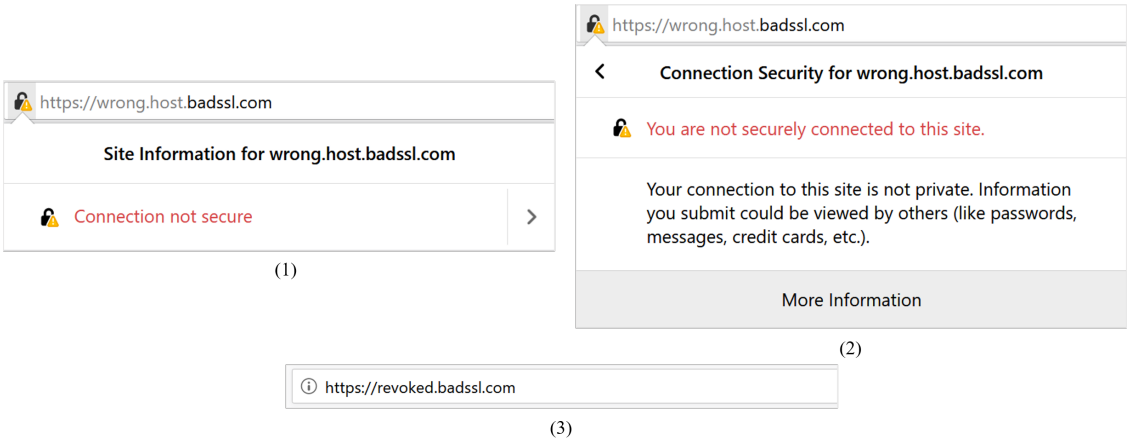

Figure 5.9.: Firefox's address bar with an overridabel error in (1) and (2), and with a critical error in (3)

symbol next to the address changes to the encircle letter i. The content of the popup is not different and, therefore, not shown here. Figure 5.9 outlines the Firefox's address bar for both types of errors.

In the next section, we discuss how Chrome and Edge communicate errors to the user.

5.1.4. Negative Indication in Chrome & Edge

The browser Google Chrome and Microsoft Edge are both based on the open-source project Chromium. Thus, the general behavior is very much alike. This section will first compare both browsers to outline their general differences and their common attributes. Afterward, we will discuss the different types of error pages using Chrome as an example.

Figure 5.10 compares both browsers on their first error page. There are only a few differences between the browsers, such as the font, the additional link "Learn more" in Chrome, and the crossed-out protocol identifier in Edge's address bar. The second page shows no difference besides the font, and we omit it here. In the following section, all referrals to Chrome's user interface also apply to Edge unless stated otherwise.

A comparison of the different screens shows that Chrome has one general design concept for all certificate-caused errors. The page has two stages. The first one warns the user that the connection is not private and what the attackers can do. It displays the error code, has a button to show the second stage, and a button to get back to the previous secure page. The content of the second stage is highly dependent on the current error code. In general, it explains the technical reason for the error in more detail.

Figure 5.11 shows the conceptual model of chrome's error page. It is comparable to

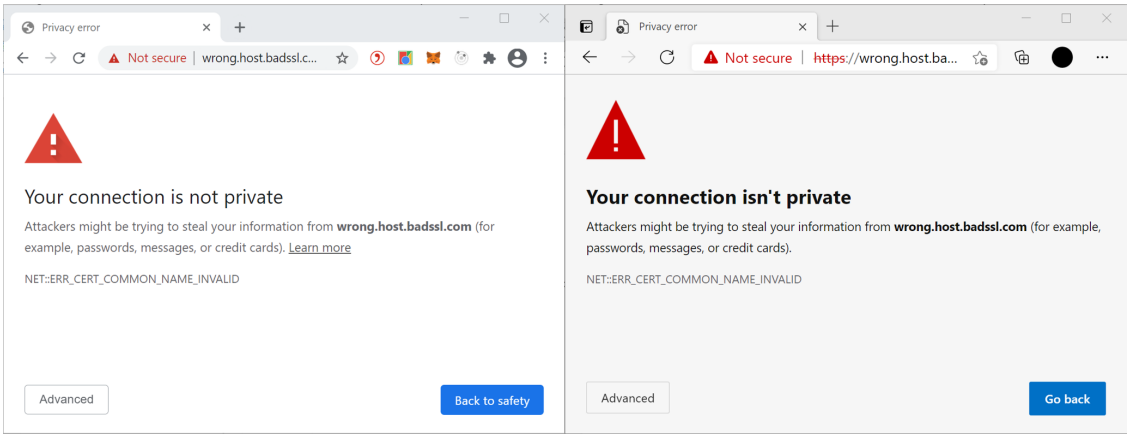

Figure 5.10.: Comparison of the first stage in Chrome's and Edge's error pages

Firefox's overridable error page. The two-stages mode is present, and the error can only be ignored on the second stage, while the user can always go back to a secure page on both stages. Additionally, both browsers have a conceptual separation between a general-purpose warning on the first stage and a more detailed technical explanation on the second stage. Chrome provides access on the first stage to an explanatory help page[3].

The key differences between the specific chrome pages for each error scenario are the error code, the describing text on the second stage, and whether the user can still proceed to the next page. We will discuss the second stage's content for the six existing different error codes in the following sections. Appendix F provides an assignment of the defined certificate errors to Chrome's error codes with the second stage's error message. We will omit the first stage for each error screen in the following, as the only difference is the error code.

Date_Invalid Error In case the *Date_Invalid* error shows up, it indicates that the certificate has expired. The text states the period since the expiration, and it points out what might have caused this error. Besides the possibility of being a server-induced error, the client can also have affected this. Thus, Chrome describes how the user can control whether the system clock is set correctly and how it can be fixed. Chrome also allows the user to proceed to the following page. The link that facilitates this navigation is not as visible as other buttons on the screen. It has the same font and color as a describing text, and it is different from the other navigational buttons. The reason for this styling might be to increase security through obscurity. Figure 5.12 shows the second stage.

Common_Name_Invalid Error If the certificate's common name does not match the URL, Chrome shows a *Common_Name_Invalid* error page. The user cannot resolve this error because the server causes it. Thus, the explanatory text on the second stage is

[3]https://support.google.com/chrome/answer/6098869

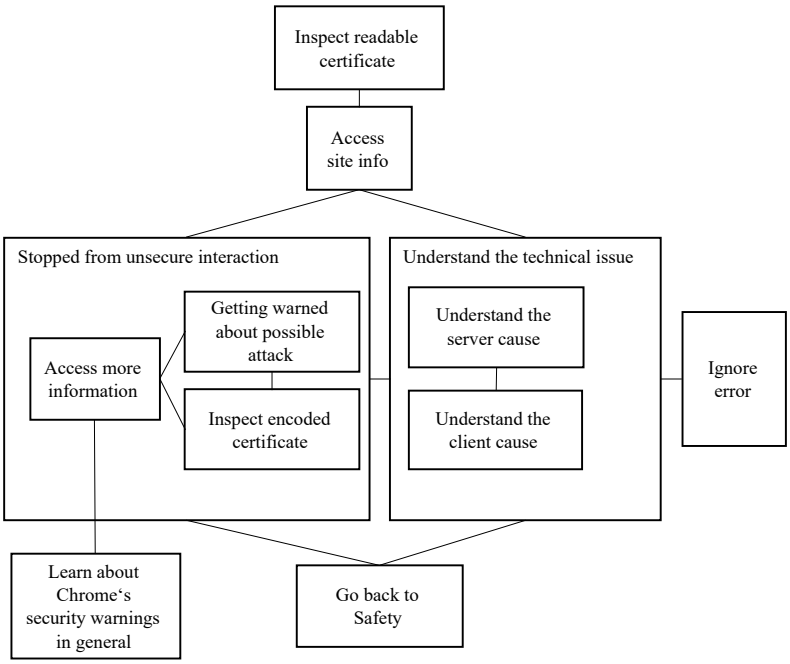

Figure 5.11.: Conceptual model of the error page in Chrome

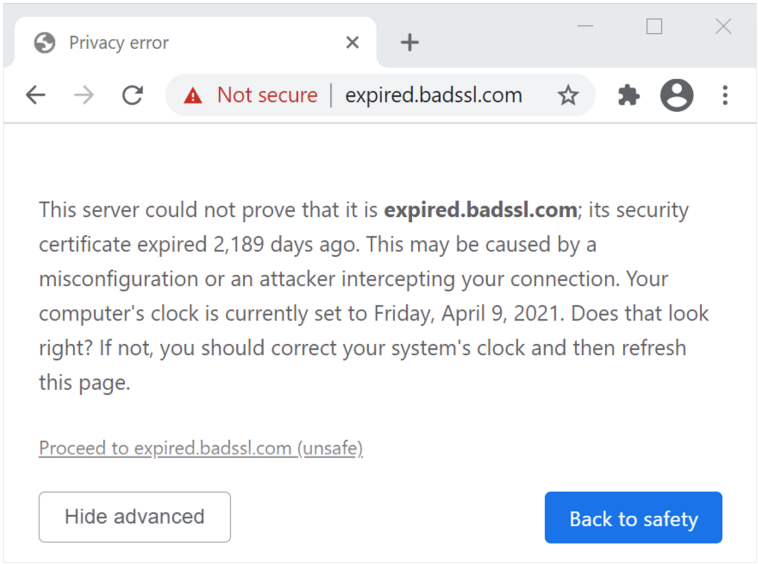

Figure 5.12.: Second stage of an expired certificate error screen in Chrome

short. The text compares both URLs, states that they do not match and that the server or an attacker might be causing this issue. The user can still proceed to the page.

Authority_Invalid Error Chrome renders the authority invalid for a variety of reasons. The root certificate may not be trusted, the server presents a self-signed certificate, an intermediate certificate of the chain might be missing, or the signature is not valid. In all of these cases, the user can not proceed to the target page. The error states that the computer's operating system does not trust the security certificate. There is no further explanation about the cause of the issue.

Weak_Signature_Algorithm Error If a certificate uses a deprecated hashing algorithm, Chrome shows a *weak_signature_algorithm* error page. The second stage explains that the signature algorithm is weak so that the certificate could have been forged. Thus, the user might be communicating with an attacker. S/he can accept the risk and proceed to the target page.

Revoked_Certificate Error The revocation of a certificate causes an error with more text on the second stage. In general, it explains what a certificate error is, and that Chrome did not receive the required credentials. The provided text illustrates that an attack might be happening or that the network configuration is causing the error. The message also highlights that the user's data is still safe because the connection was interrupted. Finally, the error states that the certificate has been revoked, and the user should try again later. S/he cannot proceed to the following page.

Certificate_Invalid Error A variety of misconfigurations render a certificate as being invalid. Refer to the table in appendix F to get an overview on the different types. The second stage is similar to the previous error screen of a revoked certificate. Aside from the general information about certificate errors, Chrome informs the user that it is impossible to process the scrambled credentials the website has sent.

The browser also indicates the negative verification of a certificate with a red warning symbol and a text "Not Secure" next to the address bar. The indication is the same for all errors mentioned above. Edge additionally crosses out the protocol identifier and highlights it in red. Clicking on the warning symbol opens a popup again. The content here is also different from the positive indication. A red text states that the connection is not secure, and further down, a text highlights that the user should not enter any sensitive data. The certificate can still be inspected. Figure 5.13 depicts the indicator and the popup for both browsers.

The next section discusses the different screens of the browsers if the website does not support the entire security protocol.

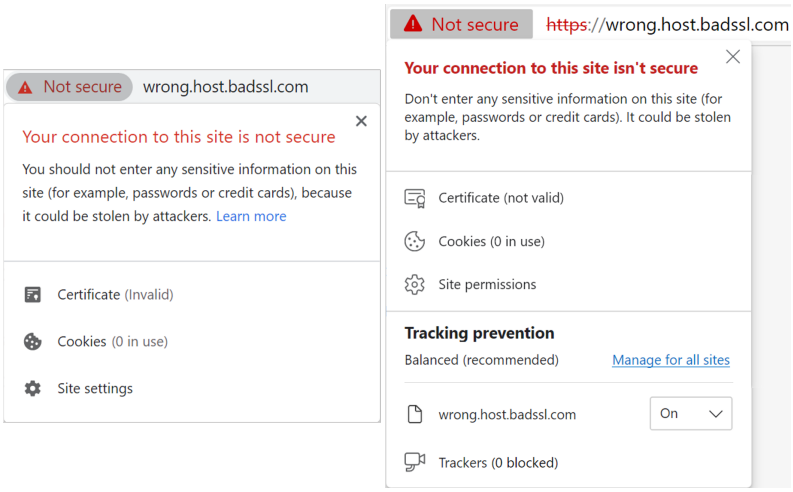

Figure 5.13.: Negative indication in Chrome and Edge

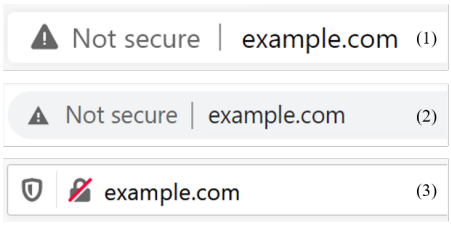

Figure 5.14.: The address bar of Firefox (1), Chrome (2), and Edge (3) serving an HTTP website

5.1.5. Indication of a Protocol Downgrade

A protocol downgrade happens in the browser if the browser is unable to establish HTTPS. In that case, a plain HTTP connection transports the information in an unsafe and unauthenticated manner. However, because the adoption of HTTPS is not high enough to block the older HTTP protocol entirely, the browsers show only a passive negative indication.

Figure 5.14 shows the address bar of Firefox, Chrome, and Edge with an HTTP page. In Firefox, the lock symbol, which would indicate a positive security state, is crossed out with a red line. Both Chrome and Edge display the same warning symbol as in an error case but without any colors. Section 3.3 presents the discussion about the ineffectiveness of passive warnings. We emphasize that all three browsers choose a modest, non-interruptive approach to warn the user that they cannot establish a security protocol. Chromium's browsers do not even use the signal color red to highlight the indicator. Besides the inefficiency of such a passive warning, the browsers choose not to interrupt the users because the insecure HTTP protocol is still in use.

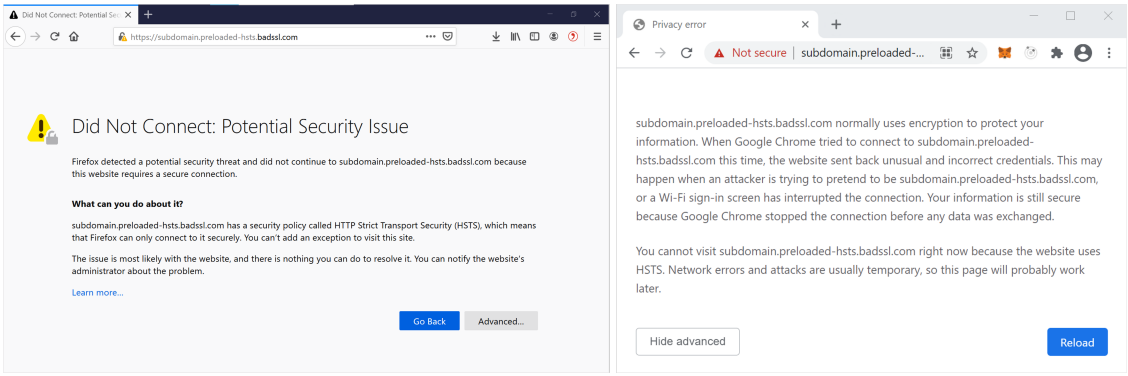

Figure 5.15.: The altered error texts in Firefox's first stage and Chrome's second if the page is known to use HSTS

An exception to this passive indication happens if the server indicates that the connection must use HTTPS. RFC 6797 defines the HTTP strict transport security (HSTS) protocol [59]. With this specification, servers require that any caller strictly communicate over HTTPS and not over HTTP. If this requirement is known for a website, the client must not downgrade the connection to HTTP. If the provided certificate is not valid, the browser must inform the user that s/he can not override this error because the website specifically forbids downgrading the security. An example page with HSTS enabled serves a certificate with a wrong hostname, which triggers a Common_Name_Invalid error in Chrome and Edge, and a BAD_CERT_DOMAIN error in Firefox. Firefox uses the two-stages design of an overridable error without the yellow frame and without the button to proceed to the insecure connection. Instead, it states on its first stage that HSTS hinders the user from adding an exception for this error. Both chromium-based browsers use the already known error design and add a paragraph on the second stage explaining that the user cannot visit the page because of an HSTS requirement. Figure 5.15 shows the second stages of Firefox and Chrome.

The difference between passive indication and interrupting warning pages during a protocol downgrade will be relevant in the later sections where we discuss the downgrade prevention of TeSC.

We conclude this analysis with the notion that, in general, the approaches of all three browsers are similar. Especially the passive indication in Chrome, Edge, and Firefox is very much alike. Firefox stands out with a varying error page design, whereas the Chromium-based browsers have one design for all different certificate errors. We cannot determine by the user interface or the literature why Firefox does not provide a more concise error design. All browsers require the user to activate the second stage of the error page to access the button, which forwards to the erroneous target website. We see that Chrome and Edge try to hide this button from the user by designing the link differently than other navigation elements on the page. Both, Firefox and Chrome provide links to external resources, which explain the protocol and error.

5.2. Use Cases of TeSC

This section describes the use cases of TeSC in MetaMask. An adoption of TeSC must support these use cases to augment the security of all user interactions with Ethereum. We compile these use cases following a top-down approach described by Wiegers and Beatty [43]. First, we analyze the business objective of this project. Based on the objective, we conduct an interface analysis of Ethereum and MetaMask to detect relevant user interactions with Ethereum. From this analysis, we develop three use cases.

5.2.1. Analysis of Primary Business Objective

The goal of the TeSC protocol is to authenticate Ethereum accounts with the help of domain names. The primary motivation of this authentication is to enhance the user's security. This thesis focuses on adopting TeSC with the security objective to prevent the misspending of Ethereum resources like Ether, ERC-20 tokens, or Gas. Refer to 8.3 for a discussion of alternative levels of security.

Following the primary security objective, the protocol must be applied during resource spending interactions with the blockchain. Only a new transaction changes the ownership of resources in Ethereum. Section 2.1.2 introduces the concept and parameters of a transaction in Ethereum. We define the misspending of resources as having values for the transaction's parameters that the user does not expect. Assuming that the transaction receiver (i.e., parameter *To*) is not the user's intended account, but a different address and this transaction is committed to the blockchain, the associated resources are misspent. The TeSC protocol can prevent such a loss if it is established in the life cycle of a transaction before MetaMask commits the transaction to Ethereum. Thereby, the user can react to an error. In summary, the protocol needs to verify the parameters at the point before the transaction becomes unchangeable, i.e., before the committer signs the transaction.

We analyse the RPC API of Ethereum [60] and conclude that the following two requests prepare a signed transaction on the basis:

- *eth_sendTransaction*, and
- *eth_signTransaction*.

The reasoning for *eth_sendTransaction* is apparent: calling this request prepares the transaction and commits it for mining with the provided gas limit. Depending on the request, it might additionally transfer ether or ERC-20 tokens.

Eth_signTransaction covers a subset of the functionality of *eth_sendTransaction*: the parameters are encoded and signed, so that this request can be committed to Ethereum with *eth_sendRawTransaction* later. The decision on spending resources is already made with

this request.

For the completeness of our analysis, we also investigate the MetaMask RPC API [61]. Web applications, which submit transactions on behalf of the authorized user in MetaMask, leverage this API to send their request to Ethereum. We conclude that no transaction on Ethereum can be triggered or prepared with this API, except by wrapping the two Ethereum RPC requests from above in the JavaScript function call *ethereum.request*.

Based on this analysis, we identify three use cases for TeSC, where it can enhance the user security in MetaMask. We derive all of these use cases from the single, relevant use case of transacting on Ethereum. The figure 5.16 depicts that relation.

The first use case covers all transactions, which a web application triggers outside of MetaMask. The second use case builds upon transactions that the user triggers when interacting with MetaMask. The last use case discusses the exchange of fungible tokens according to ERC-20 [5].

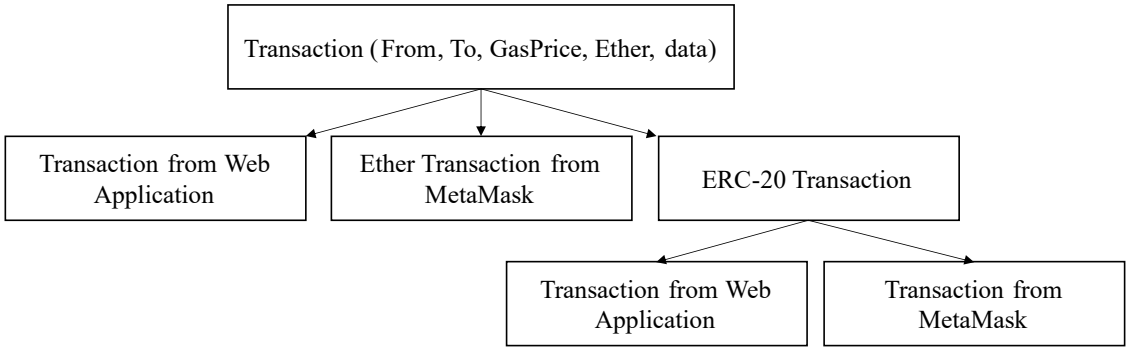

Figure 5.16.: Tree of all use cases derived from transacting to Ethereum

We do ignore all possible errors in this section. A detailed discussion of error scenarios follows in section 5.3.

5.2.2. Use Case: Transaction from a Web Application

In this use case, a web application calls the JavaScript method *ethereum.request* with the respective Ethereum RPC API requests. When the application calls this method, MetaMask pops up with a confirmation screen, which the user must approve. We assume that a reasonable action happened in the web application so that the user is aware of the popup's context. Otherwise, the user is not able to evaluate the legitimacy of the request and should decline. Figure 5.17 shows this confirmation screen. Based on the user input, the wallet forwards the transaction to Ethereum or cancels it.

Before the confirmation screen is prompted, MetaMask shall authenticate the transaction

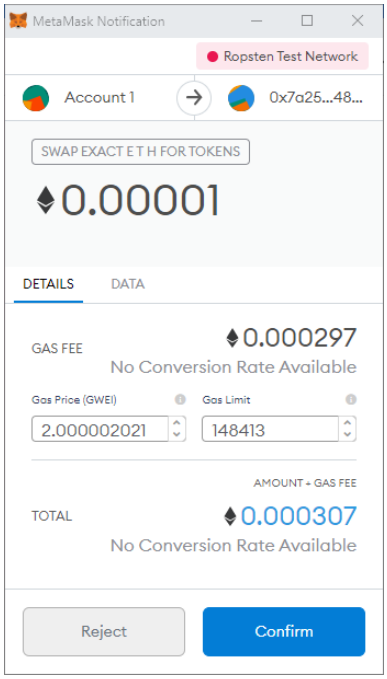

Figure 5.17.: Confirm Screen of Metamask with web application triggering transaction

receiver. If a trusted certificate can prove the authenticity of the receiver's endorsement, two different scenarios are possible:

1. Either the web application transacts to a smart contract signed by the application's domain.
2. Or an entity outside of the application's domain endorses the receiver of the transaction.

The first case is a best-case scenario from a security perspective. The certificate is trusted, and the endorsing domain matches the expected value from the browsing context. MetaMask should show a passive security indicator confirming that it can authenticate the account. The user retrieves additional information by hovering over this indicator, similar to the passive security indicators in the browser (see section 5.1). An example for this scenario is the interaction with the web application Uniswap [4]. Uniswap is an on-chain market for depositing and exchanging tokens. When the user invests in a so-called pool, a transaction executes Uniswap's smart contract and submits information about the user's investment. If the smart contract of Uniswap is TeSC-compliant, MetaMask can verify that Uniswap's domain endorses the smart contract.

In the second case, the protocol cannot verify whether the correct domain endorses the smart contract. The user needs to verify the domain manually. MetaMask shall show an

[4] app.uniswap.com

action message with the endorsing domain. The message explains that a different entity owns the transaction receiver than the current browser page. This scenario depends on the judgment of the user whether a resource misspending can be identified.

Though we expect the scenario of a payment to a different domain to be a rare case, we present the following real-world example to prove that it is occurring: The web application MyEtherWallet[5] helps to interact with Ethereum. The wallet must connect with a client-controlled wallet application to store the user's credentials. After configuring MetaMask as the backing wallet, the user may send transactions to any Ethereum address in the interface of MyEtherWallet. Internally this transaction is sent to MetaMask with *ethereum.request*. In MetaMask, a transaction arrives with a recipient address that the domain of the current browser page does not endorse. In this case, TeSC provides no capability to validate whether the recipient's address belongs to the intended identity. The user must assert the receiver's identity manually.

5.2.3. Use Case: Transaction Configuration in MetaMask

MetaMask covers this use case entirely. A precondition is that the user knows the Ethereum account's address s/he wants to transfer ether to. We do not specify the means to acquire such an address because of the vast possibilities to do this, but the later discussion will present an example.

The user opens the MetaMask wallet in the browser and accesses the user interface for sending ether. First, s/he needs to input the receiver's Ethereum address. MetaMask already supports several input formats: a QR-Code, the public hexadecimal address, or the account's ENS (Ethereum Name Service) identifier.

After entering resolvable, valid address data, the user selects the amount of ether and the gas for this transaction. The final screen is a confirmation dialog, which shows the details of the transaction. The UI displays the recipient's hexadecimal address only if it cannot resolve the address into a human-readable form, such as an ENS identifier or a wallet-specific contact name. Otherwise, the UI shows the address in a readable format.

We identify two phases in this flow where the TeSC protocol can contribute additional security features: the input phase, where the recipient's data is entered, and the confirmation phase, where the user inspects the transaction details before committing.

We augment the possible input formats and add fully qualified domain names (FQDN). TeSC can resolve an FQDN to an Ethereum address using the TeSC registry. If the domain is resolvable, MetaMask shall validate in the confirmation phase that the Ethereum account can identify itself for the domain the user has entered. Therefore, the wallet must verify the binding between the entered domain and the resolved Ethereum address. If the domain resolves ambiguously to multiple addresses, the user selects the intended

[5] myetherwallet.com

address from a dedicated dialog. If the domain is not associated with an Ethereum address, MetaMask shall show the same error, which already exists for unresolvable ENS names.

We also augment the confirmation screen to communicate the authentication state as in the previous use case. A passive indicator signals a positive result. On hovering or clicking on this indicator, the user receives more information on the identity and may also read about the protocol.

We expect that this use case could have prevented the CoinDash hack as we describe it in chapter 1. When CoinDash offers the option to invest in its company with ether, it announces its Ethereum address on its web page. The user copies this address into MetaMask. When TeSC successfully verifies the address, the user can be confident that the address belongs to CoinDash. The user can also type the domain address of CoinDash into the interface, which MetaMask resolves to the associated Ethereum address. The risk of resource misspending due to spelling errors or phishing attacks is reduced if the user is vigilant and considers the security indicators in MetaMask and the browser.

5.2.4. Use Case: ERC-20 Transactions

The last use case focuses on a special type of smart contracts: token contracts. Section 2.1.3 introduces the standard of ERC-20 tokens. Since three stakeholders participate in the transaction, this is an additional case to be considered. The stakeholders are the sender, the token receiver, and the smart contract address of the token.

In this case, a user wants to send a token to another account. The transaction can be triggered from a web application or directly in MetaMask. Thus, we will explain the use case separately for both subcases.

Use Case: ERC-20 Transactions from MetaMask

MetaMask maintains a list of predefined ERC-20 contracts with the project *contract-metadata* [62]. Furthermore, the user can manually add tokens to its wallet. MetaMask shall authenticate the contract's address with TeSC when the user introduces a new token. The user cannot trigger a token transaction in MetaMask's UI if the contract is not configured. Thus, we assume that the token contract's address is already validated with TeSC when a user starts a token transaction in MetaMask.

The user opens the interface for sending ether as in the use case in 5.2.3. In the list of assets, s/he selects the token to transfer. Then, s/he enters the token's recipient address and confirms the transaction. Figure 5.18 shows the confirmation screen of a token transaction.

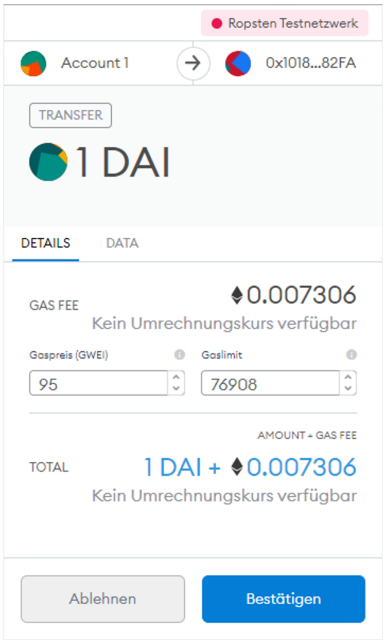

Figure 5.18.: Confirm screen on token transaction triggered in MetaMask

Since the token contract is already authenticated, MetaMask must only verify the recipient during the confirmation process. Thus, a token transaction can be handled the same way as any other transaction triggered from inside of MetaMask. Depending on the input format of the recipient's address, TeSC authenticates the recipient, and MetaMask shows the corresponding security indication in the confirmation screen. Section 5.2.3 covers this behaviour already completely.

Use Case: ERC-20 Transaction from Web Application

When a web application is triggering a transaction, MetaMask has to evaluate whether this transaction is an ERC-20 token transferal. The ERC-20 token standard specifies two functions for a token contract, which facilitate the token exchange from the holder's perspective: *transfer* and *approve*. If one of those methods is part of the transaction data, the wallet must evaluate whether the contract is indeed an ERC-20 token contract. If this is the case, TeSC shall authenticate the smart contract's address and the token's recipient address. Accordingly, passive indicators shall be visible in the UI. The difference to the use case in 5.2.2 is that the verification algorithm runs twice: once for the contract's address and once for the token recipient's address.

Appendix A provides tables that summarize all defined use cases regarding their triggers, pre- and post-conditions, exceptions, and flow.

With these three primary use cases in mind, the following section discusses all error

scenarios during a TeSC verification.

5.3. Authentication Error Scenarios

As a basis for the discussion about authentication error indication, we identify the following scenarios that need to be communicated to the user. We identify these scenarios in two ways: we analyze the use cases from section 5.2 and their control flow (see also the control flow diagrams in B). This analysis provides the protocol errors in the next section. These errors are newly introduced solely because of the design of TeSC. One important building block of the TeSC protocol is the verification of the certificates. RFC 5280 already standardizes these errors. From those sources, we condense a set of root causes that MetaMask shall communicate to the user. The second section discusses these scenarios.

5.3.1. Protocol Errors

We derive this set of errors from the description in [2] and the flow diagrams of the use cases in appendix B.

The first error happens if the endorsement stored in the blockchain has expired. A reason for this error could be a wrongly configured client's system clock. The user could be able to fix this issue. The error message should therefore mention this possibility.

With each endorsement in TeSC, certain flags can be set. One of those is the flag of exclusiveness. It indicates that the domain must exclusively endorse this Ethereum address. An exclusiveness error occurs if the TeSC registry lists another smart contract associated with this domain. Such an error renders all endorsements with this domain invalid, and MetaMask must show a corresponding message to the user.

The x509 certificate needs to be retrieved from the endorsing domain to validate the authenticity of a signature. Thus, the verification process sends an HTTPS request to the domain itself and inspects the certificate. If the certificate is not available, the protocol cannot continue, and the user shall see a corresponding error message. We expect this error to occur either due to the unavailability of the domain's server or a misconfigured endorsement where the domain name is not correctly set.

In the final step, the protocol asserts the cryptographic proof of the endorsement. The public key in the certificate and the signature in the endorsement allow verifying that the endorsement's claims are not legit and that the certificate's owner has signed and thus created the endorsement. If this verification fails, TeSC cannot establish the receiver's authenticity, and a corresponding error needs to be shown to the user.

The above discussion about protocol errors assumes that the receiver's address is TeSC-

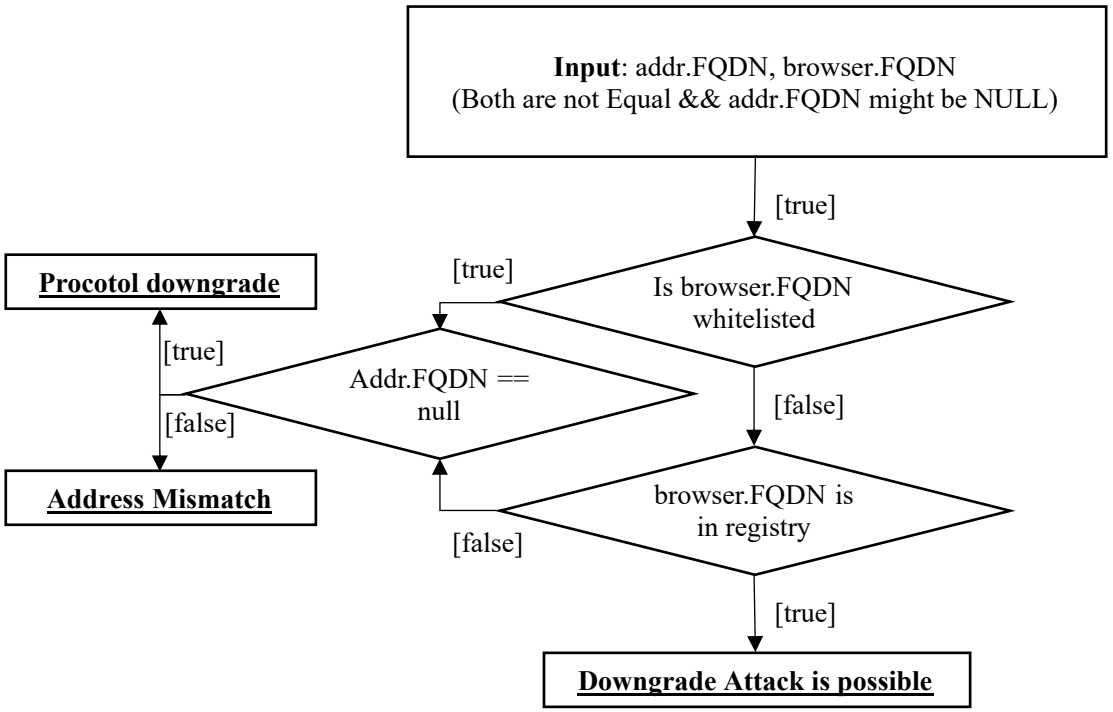

Figure 5.19.: Protocol Downgrade Algorithm

compliant. If this is not the case, MetaMask has to inform the user that it cannot guarantee the TeSC security features for this transaction. It is dangerous to downgrade the security to a non-authenticated state without interrupting the user since s/he might not realize the diminished protection. However, an enforced TeSC-compliance results in a high frequency of interrupting error messages due to the low adoption of the TeSC protocol. Thus, assessing the current threat level shall help decide how the user must be informed about the security protocol downgrade. For this reason, we specify the protocol downgrade algorithm. Figure 5.19 depicts its flow.

The algorithm decides whether MetaMask should interrupt the user actively or only show a passive negative indication. It assumes that in most cases, the user manually retrieves the receiver's address from a website, or a decentralized web application triggers the transaction programmatically. Hence, most transactions in MetaMask originate from the context of a current webpage, where the owner of the website controls the information about the receiving address and a mismatch between website domain and endorsement indicates a higher risk.

Thus, a policy is desirable that enforces the receiving Ethereum address to be bound to the same domain as the website. However, the algorithm also considers the known TeSC registration of the website's domain. This aspect allows assessing the threat level more exhaustively. The algorithm only perceives an address mismatch as dangerous if the

website's domain is already bound to an Ethereum address. I.e., the owner of the website has already registered an Ethereum address for this domain name in the TeSC registry. The receiver seems to be unfitting if the website is known to endorse a different Ethereum address than the transaction receiver. However, if the domain's endorsement behavior is unknown, MetaMask cannot determine whether a different Ethereum address would be more plausible in the current context. In that case, MetaMask should only indicate the downgraded security to the user with a passive indication.

The algorithm results in three different error states:

1. **A Downgrade Attack** happens if the website's domain is bound to a different Ethereum address. It is not relevant whether another domain endorses the receiving address or it is not TeSC compliant.
2. **An Address Mismatch** happens if the receiver's address is TeSC compliant, but the domain of the current website, which is not listed in the TeSC registry, does not endorse it.
3. Finally, if the website is not listed in the TeSC registry and the receiver's address is not TeSC compliant, the algorithm resolves to a **Downgrade Warning**.

Table 5.1 shows the error states depending on the receiver's TeSC compliance and the website's domain's listing in the TeSC registry.

	Receiver's address is TeSC-Compliant	**Receiver not TeSC-Compliant**
Current website is in registry	Downgrade Attack	Downgrade Attack
Current website not in registry	Address Mismatch	Downgrade Warning

Table 5.1.: Errors resulting from the downgrade algorithm

The verification process also evaluates the certificate of the domain on its internal validity. Trusted authorities must sign it, and the endorsed domain must match the subject of the certificate. During this process, many different errors may occur. These are discussed in the following section.

5.3.2. Certificate Errors

The validation of certificates is a well-known task. RFC 5280 [10] defines an algorithm to evaluate x509 v3 certificates. This algorithm provides several possible error states. To complement this set of errors, we also consider error states defined in RFC 8446 [57], which describes TLS v1.3. Section 2.2.5 already compares these standards in terms of their error scenarios. This section defines certificate validation errors that the TeSC

verification shall communicate to the user.

The root cause of an error determines the user's capabilities to react to it. If the cause is on the client-side, the user might fix the issue and circumvent the exception. If the owner of the certificate or the server caused this error, the user has no option to override such errors. The resolution of client-caused issues must be explained to the user. Thus, we mention this communication requirement of the error messages where it applies.

If a certificate does not adhere to the RFC 5280 standard, it is invalid. Thus, the verifier throws a *non-compliance error*. The error means that MetaMask cannot establish the security protocol. Only the owner of the certificate can mitigate this issue. If this error occurs, the user will not be able to access the associated domain either. Thus, a blocking error message is suitable to communicate this error.

If the certificate's validity period has expired or has not yet begun or the owner has revoked it, a *validity state error* happens. The certificate must not be trusted, and MetaMask should interrupt the user in his/her flow. If the period expired, the error message should state that a wrongly configured system clock at the client can cause this issue. A validity state error also happens if the certificate has been revoked. In that case, it is not a client error, but the server must provide a new valid certificate.

A *chaining error* happens if the certificate's chain cannot be trusted or established. Several reasons might cause this error: the client does not trust the root certificate, the certificate chain is incomplete, or the chain is longer than a certificate authority allows. The client might circumvent this error by adding the root certificate to its set of trusted authorities. We expect this is not the most common root cause but instead that this error is a sign of a malicious certificate.

A *tampered certificate* is recognized if the certificate's signature does not match its content. This error renders the certificate untrusted, and the process must be interrupted. MetaMask shall explain why the user cannot circumvent this error.

A *subject error* happens when the certificate does not identify the expected subject (i.e., domain) or if the subject's name does not match a constraint instantiated by a signing authority. The owner of the certificate is responsible for this configuration. Hence, an error message should explain why it blocks the user from this interaction.

RFC 8446 marks the hashing algorithms MD5 and SHA-1 as unacceptable because the security community considers them broken. We follow the argumentation of RFC 8446 and adopt this deprecation. Thus, the protocol must not use a certificate with such a signature algorithm. The owner of the certificate needs to provide a different certificate. Until then, MetaMask shall stop the user from interacting with this domain. A corresponding error message shall be displayed.

A certificate authority defines its principles of issuance in the policy extension of its certificate. If the client does not accept these principles, it throws a *usage error*. The errors

of an unacceptable key usage statement or a forbidden policy mapping also count to this category. The error message should explain that an unacceptable certificate causes the error. The user will not be able to override the error. RFC5280 defines several values for the extended key usage extension. As there is no dedicated usage defined for TeSC, we suggest that the most fitting is the TLS server authentication usage. The endorsement binds the Ethereum address to a domain name. Thus, we expect that the endorsed smart contract contributes to the server application's functionality and is a substantial part of the server. The close coupling of website and Ethereum address is the same assumption that is also relevant for the previously discusses downgrade algorithm.

Additionally, we expect that this approach enhances the protocol's usability from the domain owner's perspective. Since an additional certificate introduces more costs for the owner, the server should use the same certificate for the HTTPS handshake as the endorsement. By requiring a TLS server authentication usage, the owner can use the TLS/SSL certificate to sign the endorsement without maintaining an additional certificate with a different key usage extension. Thus, if the endorsing certificate is dedicated for different usage, a *usage error* is thrown.

We specify a *general technical error* that renders the certificate corrupt or unacceptable. For this error case, it is not possible to determine the root cause. It might be a transportation error, bit-flips, an unknown extension, or anything else. If an error happens and none of the before-mentioned scenarios match, we call it a general error.

Table 5.2 shows how the errors in RFC 5280 are mapped to above categorization.

This chapter established the basis for the following discussion about adopting TeSC in MetaMask. The browsers' TLS design concepts inspire our design proposal of indicators, which communicate the TeSC verification state. This chapter defines use cases that a security module such as a TeSC verifier must cover. Each verification error this chapter discusses has to be explained to the user so that s/he comprehends its security implications. Both the following design concept and its implementation discuss how each error is covered.

RFC 5208	**TeSC Certificate Validation Errors**
AnyPolicy uncompliance	Non-Compliance error
Chain broken	Chaining error
Chain depth error	Chaining error
Invalid signature	Tampered error
Key usage mismatch	Usage error
Name constraint error	Subject error
Non-compliance	Non-Compliance error
Policy constraint	Usage error
Policy mismatch	Usage error
Revoked	Validity state error
Unknown extension	General error
Untrusted anchor	Chaining error
Validity expired	Validity state error
Version error	Non-Compliance error
Wrong subject	Subject error

Table 5.2.: Certificate errors defined in RFC 5280 mapped to the error categories that the artefact recognizes

6. Design and Implementation

The goal of this work is to augment MetaMask with a domain name based authentication mechanism. This chapter discusses the design concept we propose to communicate the authentication state to the user in section 6.1. Building upon this concept, we implement a prototype to evaluate the design's efficacy and feasibility. Section 6.2 presents an architecture concept to evaluate the TeSC endorsement of a given Ethereum address, and it discusses the new UI components we introduce to implement the design concept.

6.1. Design Concept for TeSC in MetaMask

For any of the three use cases in section 5.2, MetaMask shows a confirmation screen to the user before it submits the transaction to Ethereum (see for example figure 5.17). The user cannot edit the transaction on this screen but may reject it or return to the previous pages. After MetaMask has evaluated the transaction parameters and has added the transaction to its internal persistence, it shows this confirmation page. We augment this screen with an indicator to communicate the TeSC validation state. We replace the confirmation screen with a warning page if the TeSC protocol throws any errors. We choose this place for state indication because the user reviews the transaction parameters on this screen to decide whether s/he wants to follow it through.

This section presents the concepts for communicating the TeSC validation state to the MetaMask user. We define a positive state indication, a passive warning indicator in case of a protocol downgrade, and a conceptual model for interrupting the user in a potentially dangerous situation. The section also describes the instantiation of this model on the example of one of the protocol errors that section 5.3 defines.

The best-case scenario from a user's perspective is a successful authentication of the receiver of a transaction. The TeSC protocol guarantees the correct binding between the current browser page and the receiver's Ethereum address. Thus, this binding shall be made visible by placing the domain name and the address next to each other. Such a close placement supports the user in considering whether s/he expects this binding to exist for the current transaction. We follow the passive positive indication known from the browsers when we place a green tick next to the addresses. The browsers use a lock symbol to indicate a connection, which HTTPS protects in terms of confidentiality, message integrity, and endpoint authentication. However, the TeSC protocol does not

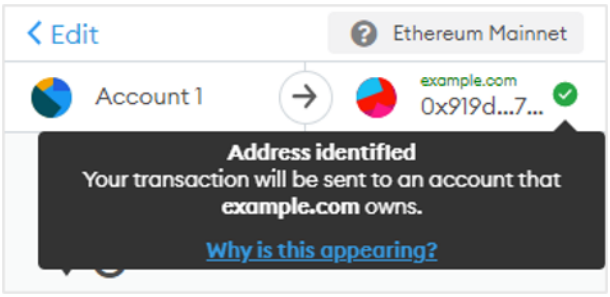

Figure 6.1.: Positive state indication in MetaMask's confirm screen

provide the features of a protected communication channel. Thus, we choose a tick to indicate an Ethereum address has proved to be associated with the given domain. If the user hovers over the symbol, a popup confirms the identification of the receiver. The current confirmation screen of MetaMask already displays the receiver's address on its top. Figure 6.1 shows the upper half of the confirmation screen with the positive indication and the popup being open.

If the receiver's address does not abide by the TeSC interface, MetaMask cannot perform the authentication. In that case, the transaction is in an insecure but not necessarily dangerous state. The user must manually verify whether the receiving address can be trusted. To indicate the negative state, we show a yellow warning indicator next to the receiver's address instead of the tick symbol. MetaMask also uses the symbol in other warning messages. Therefore, we expect the user to be familiar with it. If s/he hovers over the symbol, a message explains that the user has to verify whether this is the correct receiving address. We also use this warning indication if an active TeSC error interrupts the user, as the following discussion shows.

Any other error is communicated with an active interruption of the user. If such an error occurs, the transaction is in a potentially dangerous state. We design a general page layout to communicate this state to the user. The layout basis on a conceptual model from the previous analysis about browsers communicating HTTPS validation errors. Figure 6.2 shows the UI model at the configurational level following Parush's conceptualization [46]. The model has a two-stages approach. First, it stops the user during a dangerous transaction. A general warning message is displayed, and the user may access an additional informational page. On the second stage, the user reads more about the technical cause of the error and gets informed what s/he can do about it. On both stages, general information about the transaction is accessible. The sending account, the receiving account, and the amount of ether and Gas are always visible to provide the current interaction context. The user may always "go back to safety", i.e., cancel the transaction on both stages. However, to ignore the transaction, s/he has to be on the second stage. Comparing this with the analysis of the browsers in section 5.1, one can see the close relatedness of the conceptual model with both Firefox's and Chrome's HTTPS error pages.

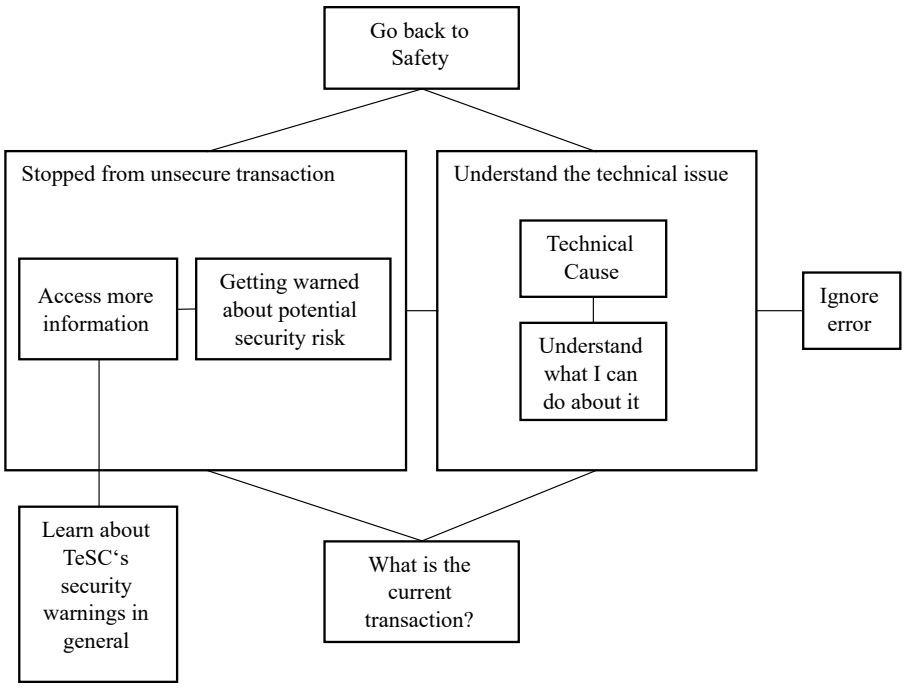

Figure 6.2.: Proposed conceptual model of an authentication failure warning

We explain the instantiation of the conceptual model using the example of an *expired endorsement error* displayed in Figure 6.3. The general information about the potentially dangerous transaction is displayed on top of both stages. The sending and receiving addresses augmented with the warning indicator of TeSC are already known from the previous discussion about passive indication. The area below states the total transaction costs: the amount of ether, which the transaction carries, and the associated Gas price on top of it. We reuse the address information and the total transaction amount from the already existing confirmation screen. They are displayed next to each other because the concepts are closely related. At the bottom of the error screen are always two buttons available: "show/hide advanced" opens and closes the second stage; "cancel" aborts the transaction. Following the design concept in Chromium-based browsers, the user interface highlights the cancel button as the main action. This action is the safest option the user can choose. Thus, highlighting it shall help the user to find the option immediately.

Between the buttons and the information about the transaction, an area is reserved for the two stages. The first stage summarizes the current situation. It warns the user that the transaction is not secure and underlines that the user’s assets are in danger. The stage shows the error code, and a link forwards to a general information page. This page informs about TeSC in general and each error code in detail. The visual elements of the warning’s design are inspired by Chrome’s error pages; this browser has a comprehensive design independent of the error types, contrary to the error pages

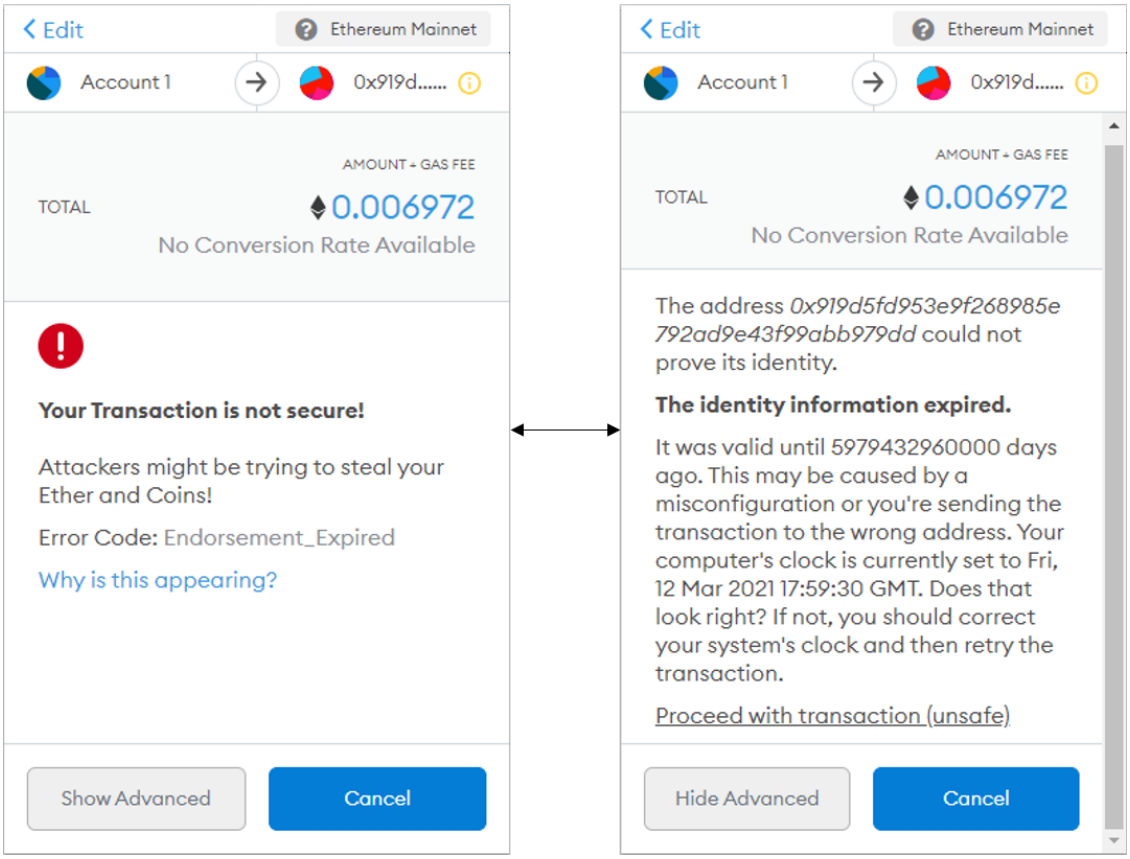

Figure 6.3.: Error page in MetaMask if a TeSC endorsement has expired

in Firefox. The second stage explains the error. First, it states that the identity for the given address cannot be established. Then, a bold text highlights the main reason for the error. Following this statement, the text reasons how the user might resolve the issue. If the error is overridable, the text ends with a link that allows ignoring the error and continuing to the confirmation screen. If the user proceeds, MetaMask shows only a passive warning indicator on the subsequent confirmation screen.

The content of the second stage depends on the error type of the TeSC validation. Appendix G adds an overview for all texts of the second stage based on their error type. It also defines whether a user may override an error or not. With this concept defined, the following section discusses the components we add to the existing MetaMask source code to communicate the correct TeSC validation state to the user.

6.2. Integration of TeSC in MetaMask

To evaluate our previous design concept and scrutinize the technical feasibility of authenticating Ethereum accounts with TeSC, we extend the existing MetaMask application with a TeSC verification logic. At the time of this project, MetaMask's current version was 9.0.3 [63]. This section discusses the TeSC verification flow and the components, which implement it. Additionally, we present the internal verification state object and the adaptions of MetaMask's UI components, which are necessary to communicate the state to the user.

6.2.1. The TeSC Verification Flow

The metadata of an unapproved transaction must be added to MetaMask's state management before it shows the confirmation screen, where the user may accept or cancel the transaction. One central controller class is responsible for managing the transaction in the state: the *TransactionController*. It's method *addUnapprovedTransaction* prepares and validates the information. Every transaction, which MetaMask commits on Ethereum, originates from this method. As it is already responsible for verifying the transaction data, it shall also authenticate the transaction receiver. This section demonstrates the TeSC verification flow that shall be triggered whenever the *TransactionController* adds a new unapproved transaction.

Based on the analysis of use cases in section 5.2 and the discussion of Gallersdörfer et al. we develop the verification logic that figure 6.4 shows. The algorithm expects as input the receiving address and the current web page serving as the point of reference with which MetaMask compares the endorsement. If the user enters the domain manually in MetaMask, the user input serves as the reference instead of the current web page. The algorithm determines whether a smart contract exists at the given Ethereum address that abides by the TeSC interface. If that is not the case, the protocol downgrade algorithm defined in section 5.3.1 determines a suitable warning message for the user. If the address implements the interface, MetaMask retrieves the claims of the endorsement. The method evaluates the expiry date and the exclusiveness flag. The algorithm requests a valid TLS certificate from the domain that the receiver's address claims to be endorsed by. Given a certificate is available and trusted, the signature of the endorsement is evaluated. If all steps are successful, the endorsement is generally valid. Finally, we assert that the endorsing domain is equal to the current browser web page. Suppose this is not the case, we forward to the protocol downgrade algorithm. Otherwise, the receiver has been authenticated successfully.

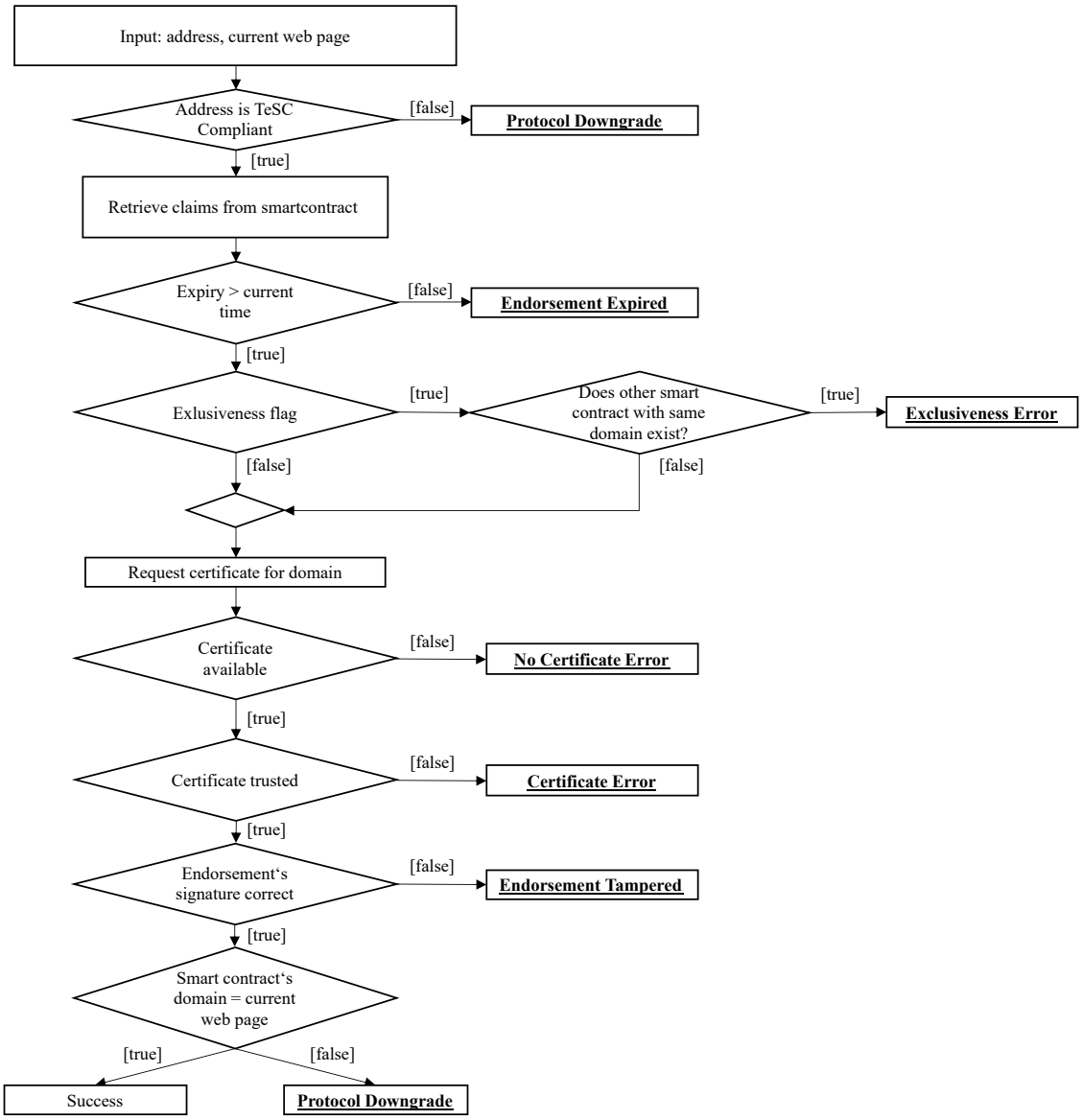

Figure 6.4.: Flow diagram of the TeSC verification algorithm

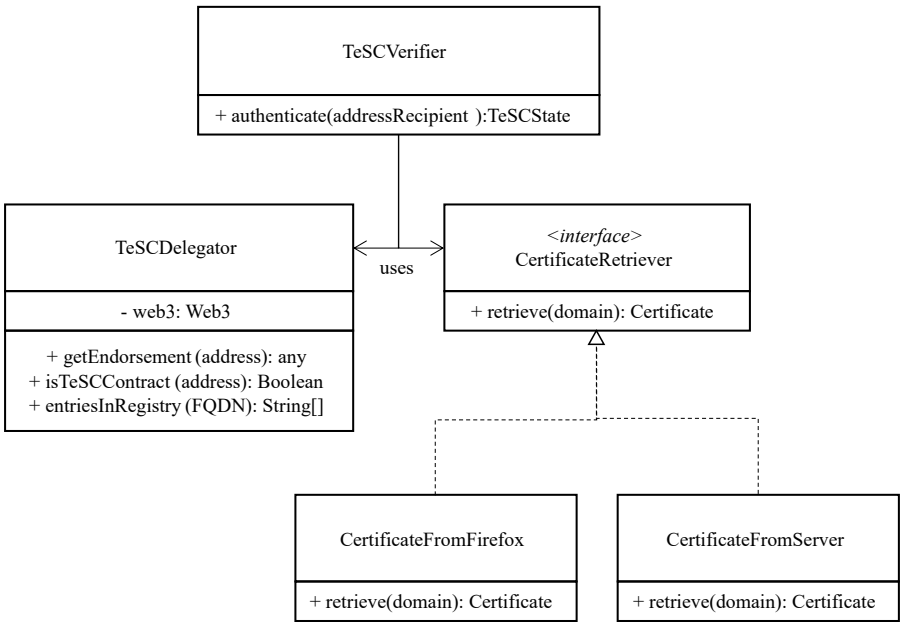

Figure 6.5.: Class diagram of TeSC verification unit in MetaMask

6.2.2. Components for TeSC Verification in MetaMask

The previously discussed TeSC verification algorithm and the protocol downgrade algorithm facilitate a transaction receiver's authentication. To implement these flows, we introduce a new component in MetaMask consisting of three classes:

- the *TescVerifier* which is responsible for managing the entire authentication algorithm,
- the *CertificateRetriever* that queries the certificate for a given domain and evaluates it according to RFC 5280 [57],
- and the *TeSCDelegator* which encapsulates utility functions that retrieve data from Ethereum.

Figure 6.5 shows the dependencies between these classes. Especially the CertificateRetriever highly depends on the API of the browser. We leverage the strategy pattern by Gamma et al. [44] to cover the different behaviors. This pattern's advantage is its adaptability to different strategies of retrieving a certificate, which might evolve with browser updates. The class model depicts two strategies. *CertificateFromFirefox* is specific for Firefox. We also propose the *CertificateFromServer* strategy as a fallback method if the browser does not support certificate retrieval natively. We present each class in the following sections and discuss how they are implemented and which problems they mitigate.

Certificate Retrieval for TeSC

To retrieve a certificate from a web server, which is reachable via HTTPS, one has to trigger the server to respond with an HTTPS handshake message carrying the certificate. This utility is often not natively accessible in high-layer programming languages. The JavaScript engines in the browsers provide several methods that result in sending requests across networks. Programmers can trigger an HTTPS handshake by setting the proper protocol when they call these JavaScript methods. However, the JavaScript API that the browsers support limits the possibilities of accessing the HTTPS handshake. If it is required to execute the entire TeSC authentication algorithm in the browser's runtime environment, the browser's API constitutes technical feasibility limits. MetaMask currently supports Firefox, Chrome, Edge, and Brave [64]. A thorough review of their API documentations shows that all of these browsers allow intercepting network requests with the *WebRequest* API for WebExtensions [65], [66]. However, only Firefox provides the possibility to access the certificate of an HTTPS request with the interception of the WebRequest API. We have not found any other option to access the requested certificates of a webserver with the browser's JavaScript API. Thus, a browser-based implementation of the certificate retrieval is only feasible with Firefox. Listing 6.1 shows the Firefox implementation of certificate retrieval.

The WebRequest API of Firefox allows accessing the request's security information with an identifier the browser uses internally to manage network requests. The identifier becomes available if the event *onHeadersReceived* triggers its listeners. Using this listener, we can access the certificate for a given domain name. The certificate retrieval in Firefox requires four steps:

1. MetaMask registers a listener for the event *onHeadersReceived* for any request to the given domain name.
2. The program sends an HTTPS request to the domain and awaits its resolution.
3. When the listener is triggered, the algorithm requests the certificate with *webRequest.getSecurityInfo* and stores it in a variable that is accessible outside of the listener function.
4. When the request succeeds, the certificate is available, and the listener can be removed again.

Some aspects are essential to note here. We send the request with the *fetch*-API. This method does succeed even if an HTTP error occurs. It only fails on networking errors. This error model is advantageous, as we do not know whether the domain supports an empty GET request to the root path. However, it is not relevant whether the server responds to the request with application data as long as a valid HTTPS connection can be established. An invalid HTTPS connection causes such a networking error. Thus, if the request succeeds, we know that the certificate is valid. Otherwise, an error would

have occurred. If the HTTPS request has been successful, we also know that the listener has been executed, and the certificate must be available. A limitation of this approach is that Firefox's network error does not provide detailed information about why a network error occurred. Thus, we cannot apply the granular error communication for certificate errors defined in section 5.3.2. MetaMask only shows a general technical exception in the UI if a network error occurs. An HTTPS error at a given domain means that the user cannot access the website in the browser via HTTPS either. Thus, we expect the user to see a detailed notification in the browser that s/he is at an insecure webpage.

```
1  const certificateFromFirefox = async function (domain) {
2      let cert;
3
4      const webRequestListener = async function(details) {
5          const securityInfo = await browser.webRequest.getSecurityInfo(
               details.requestId, {
6              certificateChain: false,
7              rawDER: true
8          })
9          if (securityInfo?.certificates) {
10             cert = extractCertificate(securityInfo?.certificates)
11         }
12     }
13
14     await extensions.webRequest.onHeadersReceived.addListener(
           webRequestListener, {urls:['https://${domain}/']}, ["blocking"
           ])
15
16     await fetch('https://${domain}/')
17
18     if (!cert) {
19         throw new TescError(TESC_ERROR_CODES.NO_CERTIFICATE_ERROR)
20     }
21     return cert
22 }
```

Listing 6.1: Certificate retrieval in Firefox

The problem with any other browser is that they do not support the method *getSecurityInfo*. To our knowledge, it is not possible in the Chromium-based browsers to access a certificate at any webserver solely with Google's JavaScript engine V8. Thus, we propose a server-based approach where MetaMasks forwards the request to the suitable server that handles certificate retrieval for the given domain. A server-based approach has several indications and requirements, which we cannot cover in this thesis. We point to the works of Hermann et al., who developed a server, that can retrieve certificates from

any domain [13]. Following their findings, we suggest two approaches in section 8.3, which could be interesting to research in follow-up work.

We continue with the *TeSCDelegator*, that encapsulates the chain interaction in regards to TeSC.

Chain Interaction in TeSC

The TeSC verification algorithm requires several queries to Ethereum. To structure the code, we move these requests into a dedicated delegator class along with some data transformation logic. The delegator is instantiated with an already existing web3.js instance. Web3.js facilitates communication with Ethereum. For interacting with a contract, web3.js requires that the contract's interface is available in a JSON format. The *TeSCDelegator* calls three different smart contracts: the TeSC endorsed contract by Gallersdoerfer [12], an ERC-165 contract by Reitwiener et al. [7], and the TeSCRegistry contract by Hermann et al. [13].

The method *getEndorsement* collects the endorsement information at the given Ethereum address. All six claims form the endorsement object that the delegator returns. Before it returns the object, the method decodes the hexadecimal number of the flags to an array with constants, and it asserts that all attributes are set. Otherwise, we would throw an error because the endorsement is not complete.

To evaluate whether the given Ethereum address is a TeSC compliant contract, the method *isTeSCContract* assesses two things: it determines whether the address is a contract address, and it uses the ERC-165 standard to verify whether it claims to support the TeSC interface. The code of the TeSC interface is *0xd7de9043*. If the address satisfies both requirements, the verifier assumes that it is a TeSC compliant contract.

The third contract is the TeSC registry. The method *entriesInRegistry* performs a lookup for a given domain name in the registry. It returns an array with all Ethereum addresses currently associated with the domain name in the registry.

The TeSCVerifier uses these three methods to retrieve all information from the blockchain to authenticate an Ethereum address. The following section discusses the verifier.

TeSC Verification

The class *TeSCVerifier* executes the entire TeSC verification algorithm in the method *authenticate* according to the flow diagram that figure 6.4 depicts. Its implementation is straightforward, and we will not discuss it in detail here. This section presents decisions regarding the signature evaluation, the matching logic of the hostname in the current browser page with the domain name in the endorsement, and the implications

of whitelisting a domain for Downgrade Attacks.

Listing 6.2 shows how a signature is verified in MetaMask. We implement this method based on the existing TeSC API server by Hermann et al. [13]. The certificate retriever returns the certificate as an object of the npm package *fidm/x509* [67]. This package provides the utilities to evaluate the signature. The claims of the endorsement compose the text that TeSC expects the signature to sign. The claims are the contract address, the domain name, the expiry date, and the hexadecimal encoded flags. A dot separates the values from one another. The verification algorithm requires this text, the signature encoded as a Buffer, and the signature algorithm's name. As the code listing shows, we set the signature algorithm to RSA-SHA256. Thus, Metamask can only verify endorsements whose signatures use this algorithm. The restriction to this specific cryptographic method is sufficient for this prototype. We acknowledge that a viable verifier of TeSC must be able to consume all known types of signature algorithms to foster the protocol's acceptance. Thus, an enhancement of this prototype should also dynamically determine the correct signature algorithm. The verification method returns a Boolean value whether it can validate the signature or not. Depending on the result of this method, the TeSCVerifier either continues or throws an appropriate error.

```
1  export default class TeSCVerifier {
2      /* Omitted constructor, authentication, and some helper methods */
3      async verifySignature(endorsement, address) {
4          const certificate = await this.certificateRetriever(
               endorsement.domain);
5          const claim = this.createClaim(address, endorsement)
6          const pubkey = certificate.publicKey
7          const buf = Buffer.from(endorsement.signature, "base64")
8          return pubkey.verify(claim, buf, "RSA-SHA256")
9      }
10
11     createClaim (contractAddress, endorsement) {
12         const {domain, expiry, flagsEnc} = endorsement
13         const address = this.web3.toChecksumAddress(contractAddress)
14         return `${address}.${domain}.${expiry}.${flagsEnc}`
15     }
16 }
```

Listing 6.2: Signature verification

After successfully evaluating the endorsement values, retrieving a trusted certificate, and verifying the endorsement's signature, the last step of the algorithm is to ensure that the domain in the endorsement matches the current webpage. If this is true, MetaMask can authenticate the receiver, and it indicates the positive state to the user. Otherwise, the security of the TeSC protocol cannot be ensured, and the downgrade algorithm evaluates

whether the user must be warned of a potentially dangerous attack. The matching of the current browser domain with the endorsement is not necessarily a check of equality. Gallersdörfer et al. define the flag *ALLOW_SUBDOMAIN*. It allows smart contracts to be used in a subdomain context, even though only the regular domain endorses the contract. The authors do not specify the number of valid subdomains. Our implementation in the prototype tolerates only one step of difference in the domain hierarchy between the current webpage and the endorsement's domain. That implies that an endorsement of the domain *example.com* never matches with a webpage at *foo.bar.example.com*, but it can be valid for the webpage *bar.example.com* if the respective flag is set. When the endorsement sets this flag to true, MetaMask requires the reference domain to be a subdomain of the endorsing domain. This strict enforcement is undoubtedly arguable and may be discussed further in research works following this thesis.

If the matching of the domains fails, the downgrade algorithm evaluates the immediate security threat heuristically, as section 5.3.1 discusses. Part of this evaluation is a whitelist that allows the user to customize the error behavior. If a domain is whitelisted, MetaMask does not interrupt the user's flow on a Downgrade Attack but shows only a passive negative indication. The whitelist must be persistently stored even if the browser is closed. The *PreferencesController* is responsible for maintaining the user's customization. We add the whitelist as an array of strings in the observable, persistent store of this controller. That requires adding a method that updates the current whitelist with a changed set of domains. Additionally, we implement a Boolean function that evaluates whether the current whitelist includes a given domain name. The TeSCVerifier uses the second function to decide about the appropriate error state.

After executing the authentication algorithm, the *TeSCVerifier* returns a state object to the *TransactionController* holding the result or state of the evaluation. The controller adds the state to the meta-information of the unapproved transaction object. Finally, the metadata is stored in the global Redux store of the application and becomes available for any other component to read or modify. Thus, any component that accesses the transaction information may also access the current TeSC evaluation state. This design choice helps to decouple the evaluation logic from the presentation layer. The next section presents the data model of the state object.

6.2.3. The Global TeSC Evaluation State

MetaMask uses Redux to manage its current state centrally in one place. As a predictable state container, Redux unifies the application's data operations. Its reproducible logic ensures consistent behavior across environments. [68] Internally, the data store is a JSON object. MetaMask shows the confirmation screen if the Redux store contains unconfirmed transaction data. We add the TeSC state at the path `confirmTransaction.txData.tesc`. Listing 6.3 shows an abbreviated JSON file of the redux store with a positive TeSC state.

```
{
  /*...*/
  confirmTransaction: {
    txData: {
      /*...*/
      tesc: {
        validationState: 'TESC_AUTHENTICATED_RECIPIENT',
        domain: 'example.com',
        referencePage: 'example.com'
      }
    },
    /*...*/
  }
}
```

Listing 6.3: Positive TeSC state in the Redux store

The TeSC object has three attributes, which are always expected to be present: the *validation state*, the endorsing *domain*, and the *referencePage* holding the current webpage. The validation state tells MetaMask if it should show an interrupting error page or what kind of passive indication is appropriate. This attribute has five different values, as listing 6.4 shows.

```
const TESC_VERIFICATION_STATE = {
    AUTHENTICATED:"TESC_AUTHENTICATED_RECIPIENT",
    ERROR:"INTERRUPT_DUE_TESC_ERROR",
    PROCEED_THROUGH_ERROR: "PROCEED_THROUGH_ERROR",
    DOWNGRADE_WARNING:"DOWNGRADE_OF_TESC_PROTOCOL",
    ADDRESS_MISMATCH: "ADDRESS_MISMATCH"
}
```

Listing 6.4: TeSC verification state

Most of the values are self-explanatory. The downgrade warning and the address mismatch occur if the downgrade algorithm does not detect a serious security risk, but MetaMask must highlight that the domains do not match. Section 5.3.1 discusses when which of both states is appropriate. If the verification algorithm succeeds, TeSC is in an authenticated state. If TeSC is in an erroneous state, Redux also stores the error code in an additional attribute. When the user decides to ignore the error and proceeds with the transaction, TeSC is in the *PROCEED_THROUGH_ERROR* state. Depending on the error code, TeSC might add more attributes. For example, if the endorsement expired, the state also carries the expiration date so that the user interface may inform about the reasons for the error message.

The TeSC state object contains all the information so that the presentation layer shows the correct indicator and interrupts the user if the TeSC verification has failed. The last section of this chapter discusses the user interface components that communicate the current state to the user.

6.2.4. User Interface Components for TeSC

Based on the validation state, which the previously discussed algorithm determines, the user interface shows the corresponding and appropriate indication. It either interrupts the user with a warning page, or it shows a passive indicator. The previous section 6.1 presents the model of the user interface. The following discussion introduces the UI components that realize the UI concept from a technical perspective.

Interrupting Error Page

If the user must be interrupted due to an error during the authentication of the receiving address, the verifier sets the TeSC validation state in the Redux store to *ERROR*. Based on this state, the application routes the user to a new page that warns about the error and highlights the options to move forward with the transaction.

MetaMask already has a routing class, which redirects the user to the appropriate confirmation page depending on the type of transaction. Its name is *ConfirmTransactionSwitch*. We extend it to redirect to the TeSC error route if the validation is in an erroneous state. The UI component that serves the content after the redirection is the newly introduced class *ConfirmTeSCError*.

The page informing the user about the TeSC error reuses several concepts of the other transaction confirmation pages. For example, all other sites display the header with an edit button and an element that shows the sender and receiver addresses. The *ConfirmDetailRow* component is also used in the summary screens to display the total transaction amount. Furthermore, the *PageContainerFooter*, which materializes the two action buttons at the bottom of the UI, exists already, too. The logic, which the *ConfirmTeSCError* class implements, derives from the already available implementations of other confirmation screens. Nevertheless, the order of elements is different from the existing components. The total transaction amount is usually placed at the bottom of the page. Also, the screen does not show the parameters of the transaction fee. Finally, the TeSC error confirmation page uses the new component *DetailsTeSCError*, which is responsible for rendering the different stages depending on the error code.

The first stage always has the same content for each error type except for the displayed error code. Thus, the detailed TeSC error component only shows different pages on the second stage. We initialize the component with a mapping of error codes to the

respective page generator methods. Each generator method defines whether the user may ignore this error. Appendix G defines the content of the second stage for each error code.

A passive state indicator complements this active interruption of the user's flow, which we discuss in the next section.

Passive Indicator Component

MetaMask uses different confirmation pages for different types of transactions. The indicator must be visible on each of these screens. To reduce redundancy, we extend a component that is visible on all of them. The *ReceiverToRecipient* class is responsible for displaying the sender's and receiver's addresses. It is used for all types of transactions, and due to the semantical affiliation of the component's information with the TeSC protocol, it is an ideal candidate to enhance with the TeSC indication. Thus, this component uses the new class *TeSCIndicator* to add a symbol at the end of the recipient's address.

Additionally, ReceiverToRecipient is responsible for displaying the endorsed domain if available in the current TeSC state. If TeSC could successfully authenticate the receiving address, it displays the text in green, otherwise red.

The *TeSCIndicator* consists of two components: the symbol and the popup. The image of the symbol depends again on whether the authentication was successful or not. Figure 6.6 shows the green tick and the yellow warning sign for the erroneous case.

Figure 6.6.: Symbols for TeSC state indication

The popup's content also depends on the validation state. If the receiver is authenticated, the message states that the transaction will be sent to an account owned by the respective domain. Suppose the domain addresses of the endorsement and the current website mismatch, the popup's content depends on whether an endorsement is available at all. With an existing endorsement, the popup states that both addresses do not match: the domain name in the endorsement and the current website's address. For manual comparison, MetaMask displays both addresses next to each other. If the endorsement is unavailable, the comparison of both addresses is not possible. Instead, the popup highlights that the current website is associated with different Ethereum addresses. A mismatch error without an endorsement happens if the verifier would classify the transaction as a downgrade attack, though the user has whitelisted the current website's

domain. With any other validation state, the popup states that MetaMask cannot identify the recipient and that the user has to manually verify the receiver's authenticity.

This chapter presented our proposed design concept for MetaMask and discussed how we integrate the TeSC verification logic and the design in the wallet application. While the user interface design concept proves to be technically viable, the implementation of the verification algorithm reveals a significant limitation. A client-only approach is only feasible in Firefox. Any other browser does not support the necessary functionality in its JavaScript API to retrieve a certificate. We acknowledge that a widespread acceptance of the protocol requires an implementation for other browsers such as Chrome. In section 8.3 we discuss approaches to overcome these limitations, which should be investigated further. Nevertheless, we show with the prototype for Firefox that integrating the TeSC protocol is possible. The following section evaluates the efficiency of our proposed design concept and its prototypical implementation in Firefox to communicate an attack to the users and enhance their security.

7. Evaluation

With the last of our three research questions, we ask whether our proposition improves the user's security within Ethereum. To answer this question, we evaluate the previously described design proposal and its implementation with a user study of 40 participants. Each participant is required to interact with MetaMask version 9.0.3 and our prototype to fulfill a given task. The task corresponds to the second use case, in which the user manually enters the receiver address (cf. section 5.2.3). While the user is repeating the transaction several times, we induce a Downgrade Attack. We choose this type of error because it could be detectable without using TeSC: the changing Ethereum address should alert the participant. Additionally, we suspect that this error would have shown during the Coindash attack, which we describe in the introduction.

The following section 7.1 describes our test environment's general technical setup before we discuss the results in detail. We discuss the conduct of the test in section 7.2. We observe the behavior of our participants and count the conversion rate whether they cancel the dangerous transaction. Section 7.3 discusses the results of our observations and complements the analysis with the results of a questionnaire that the participants answered.

7.1. Test Environment

We prepare a test environment with which the participants connect from their own devices. As we have to conduct each session remotely, this is a convenient approach to confront each participant with the same surroundings, enhancing the internal validity.

Figure 7.1 displays the entire environment with all of its components. The user connects via the remote desktop protocol (RDP) with our Windows server instance. This server provides the testbed for the participants. On the desktop, we store three PDF documents containing instructions. Two of them contain a link to an imaginary enterprise called *greatcoin.io*, which simulates an initial coin offering. Since this enterprise does not exist, the host file creates a loopback to the local machine for any request to this domain. An Nginx serves via HTTPS a manually created HTML page at *https://localhost* that provides the enterprise website. A privately controlled authority signs GreatCoin's certificate. We add the self-signed CA certificate to Firefox so that the browsers can initiate a trusted HTTPS connection. To create both certificates, we use the tool OpenSSL [56]. This part

of the setup provides the scenario into which the participants shall project themselves.

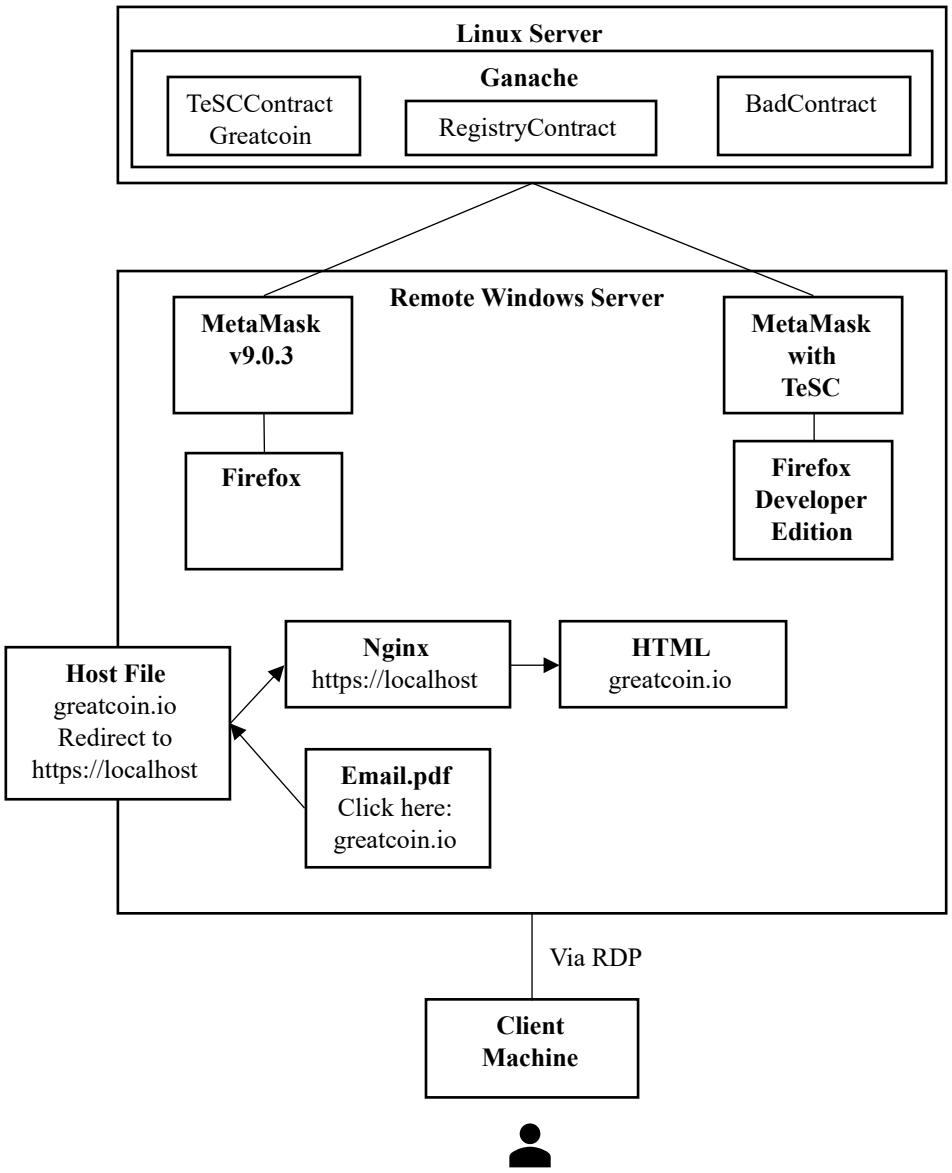

Figure 7.1.: Usability test environment

The Windows server has installed two different Firefox instances: the regular one and the Firefox Developer Edition. While we configure the regular one with the publicly available MetaMask version 9.0.3, the developer instance has our prototype built-in. We need to use the Firefox Developer Edition because the browser prohibits installing unsigned web extensions. Only the developer version allows overriding this rule. Both MetaMask instances connect to a private test chain that we run on an additional server instance. The separately run test chain allows us to modify the blockchain during the

experiment without accessing the Windows server. We configure each MetaMask with the same Ethereum account, which owns two ether for each test run. To define which MetaMask instance the participant shall use, we set the respective Firefox instance as the default PDF tool. Therefore, the participants open the scenario instructions in the correct browser. Any link they will be clicking on will also open in this browser. Only in one case, we needed to instruct a test person to stay in this browser when s/he wanted to use a different one.

The test chain contains three contracts. The *RegistryContract* is an implementation of the TeSC registry interface. It is provided by Hermann et al. [13]. Also, the *TeSCContract*, which the certificate of GreatCoin endorses, is built upon a reference implementation by Hermann et al. For the TeSCContract, we added the functionality of an ERC-20 token contract. When a user sends ether to this address, the contract credits the same amount of token to the user. Finally, the *BadContract* accepts any funds that a transaction sends to the contract's address. No domain endorses this contract. It simulates the malicious attack that steals funds. Table 7.1 displays the Ethereum addresses used in the experiment. We highlight the difference between the legit GreatCoin address and the bad contract address. We assume that this difference improves the user's ability to detect the change of addresses.

TeSCRegistry	`0x0678D9838740c79170139e6d48b86b71460795c2`
GreatCoin Contract	`0x919d5FD953e9F268985e792aD9E43F99AbB979dd`
Bad Contract	`0x1566E143b59ba6590d52D6fB3bf2fc4f6e7d5ebF`
Participant Account	`0x5C553867B3B01D4F2e68B0070c1E84e1e12E4A0C`

Table 7.1.: Ethereum Addresses for Usability Test

With this environment, we can conduct the tests as the next section describes them.

7.2. Test Procedure

The overall scenario derives from the use case of manually triggering a transaction in MetaMask. To determine the efficiency of our prototype, we use an A/B-testing design. Thus, we compare the participant's ability to detect an attack if s/he uses the original MetaMask or our augmented prototype. There is no data available about the security performance of users in the regular MetaMask. Thus, each participant receives both treatments during the experiment: the original MetaMask and the prototype. Therefore, the measurement is within-subjects. To control biasing the user's response based on the treatments' order, we randomly assign the order to each participant. The procedure of the experiment is independent of the order of treatments.

We ensure that all participants have at least an elementary understanding of the technology. Thus, the experiment starts with an informational document that explains

the basics of Ethereum and MetaMask. The text states that Ethereum is a blockchain, and it shows an Ethereum address as an example. We also highlight that it executes transactions without requiring external supervision by any authority or organization. The text describes MetaMask as a user interface that facilitates the interaction with Ethereum to transfer ether. It states that the exchange rate is approximately $ 1720 for one ether. After the participant has read this information, we show where to find MetaMask in the browser and how to access it.

As a next step, the experimenters ask the participant to open the first scenario text. This text describes a situation that should lead the participant to execute a transaction. However, we have to make sure that s/he is still aware of the possibility of losing funds if something goes wrong. Therefore, the text establishes a scenario in which a trusted friend named Alice exists, an expert in trading and investment. She recommends an investment in a novel cryptocurrency that is called "GreatCoin". The enterprise is currently offering its coins to the public for the first time. The document highlights that the participant is willing to invest his/her ether in high-risk ventures. S/he does not require the money for anything else. The text ends with a picture of a mail that Alice sends. The e-mail highlights the business opportunity, and it contains a link to the enterprise's homepage and the Ethereum address of GreatCoin's smart contract. Alice also states that the participant is bearing the risk all by him/herself. Figure 7.2 displays the first mail that Alice sends. If the user wants to assert whether this is a legitimate mail from Alice, the supervisors can confirm that.

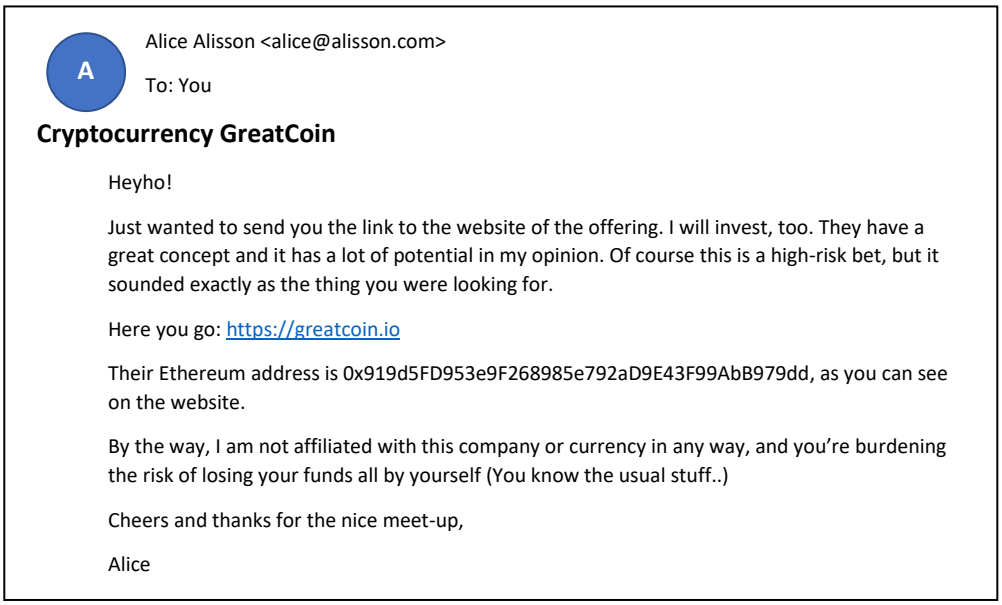

Figure 7.2.: First mail from Alice to invest in GreatCoin

With this background information, the experimenters ask the participants to project themselves in this situation and start interacting with the system. Eventually, they open the link in the mail to investigate the proposed investment. Figure 7.3 depicts the web page that opens. While the HTML page provides some information about the opportunity, we cannot anticipate every required information, which the participants need to trust that this is a legitimate opportunity. Secondary sources of information, such as Etherscan or testimonials in newspaper articles or blog posts, are not available. If the participant asks for such information, the experiment's conductors tell them that they should assume to find compelling information about the enterprise in general. Still, the information is scarce because this is an early investment that has not attracted much attention. Some participants might want to read the smart contract code by accessing the meta-application Etherscan. In that case, we highlight that the address shows that there is a contract deployed. However, the code itself is not available. Due to the high level of trust that they place in Alice, the missing code does not seem to cause that any participant does not execute the transaction. We do not answer any questions about our assessment of the situation: e.g., "Is this secure?" or "Should I send the transaction?" should be answered by asking the participant to decide this as s/he would in a real-life situation.

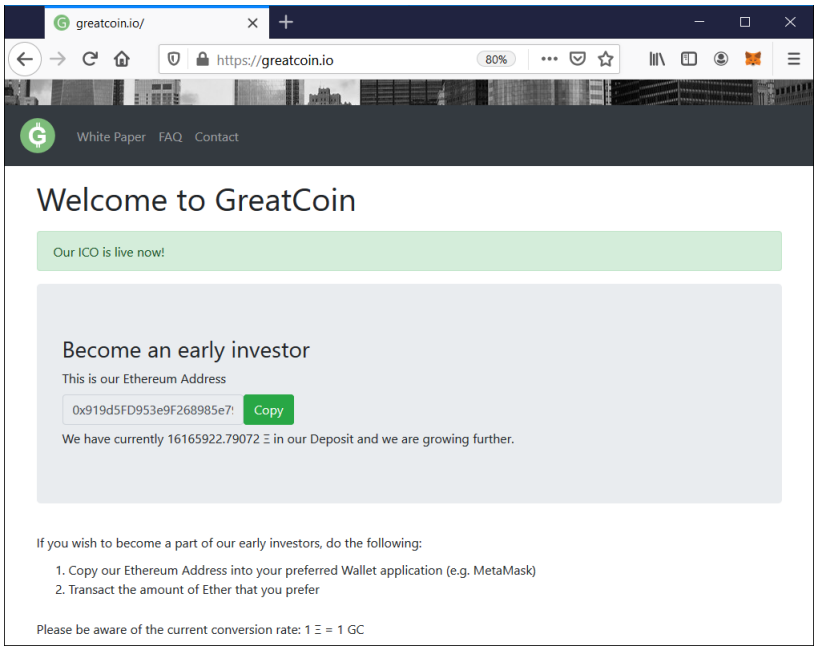

Figure 7.3.: GreatCoin's homepage

After investigating the web page thoroughly, the participants start to transact ether to the Ethereum address in the webpage. Fifteen participants compare the address on the webpage with the one included in the mail, and seven want to use Etherscan. In total, 18 participants mention secondary sources that they would use to verify the

address's legitimacy. The time the participants spend pondering about the transaction amount in MetaMask, lets us believe that they suspect the survey to be studying the chosen transaction amount. This misunderstanding prevents that the performance bias influences the actual variables of interest. This bias can cause the participants to change their behavior based on the suspected interest of the study.

After the successful transaction, we ask the participants to read the second scenario text. It states that the first investment proves to be profitable. It also contains a new mail, which Alice sends a couple of days later. Figure 7.4 depicts it. Alice congratulates the participant on its investment and suggests an additional offer that the enterprise extends to its early investors. The mail contains a new link for this offer. We also include the previous mail, which contains the Ethereum address of the previous investment.

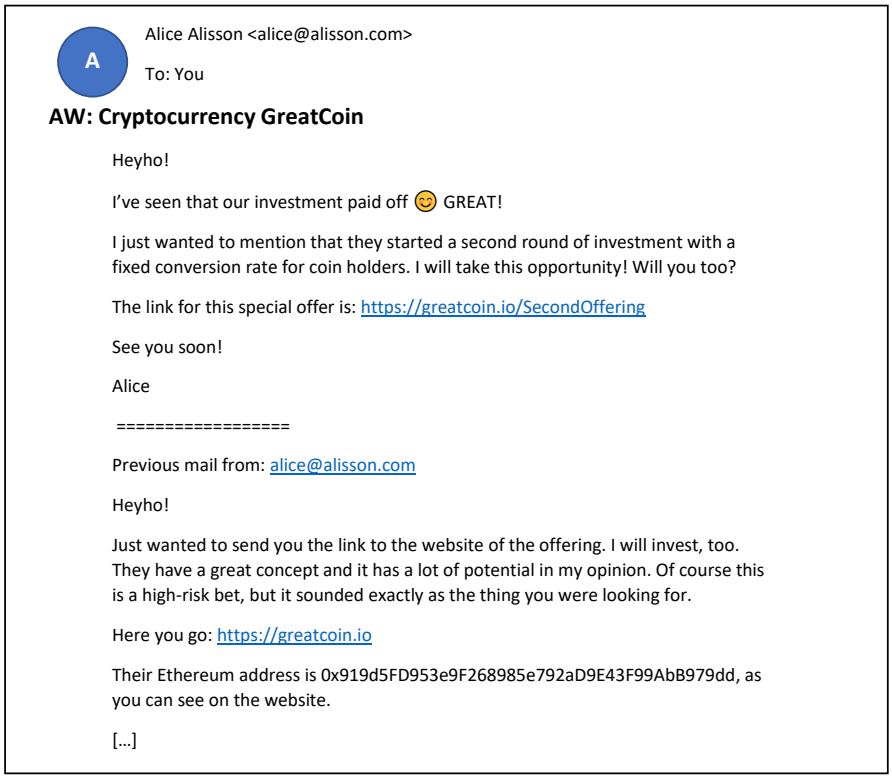

Figure 7.4.: Second mail from Alice to invest again in GreatCoin

When the participants open the provided link, the website is similar to the one in figure 7.3 with three differences: the stated conversion rate improves, the text highlights that this is a special offer for GreatCoin's early investors, and the displayed Ethereum address is different. This Ethereum address points to the malicious contract that is not associated with GreatCoin in any way. If the participants transact their ether to

this address, they lose their funds to an attacker. Again the user might describe how they would investigate this new offer. The source code cannot be accessed, but we tell participants, who ask for it, that Etherscan shows a different bytecode. Depending on the MetaMask instance, a transaction to this address is interrupted with a TeSC error. We measure the conversion rate, whether the user will confirm the transaction or is reluctant to do so.

After this scenario, we request the participants to answer the first half of a survey. We ask about the participant's experience with blockchain and his/her perception of any errors that might have happened. In the meantime, the experiment's conductor changes the default PDF tool to the other Firefox instance. Now the participant interacts with a different MetaMask when we repeat both scenarios of the experiment. From a user's perspective, there is no difference between this iteration to the previous one, except for the TeSC error when transacting to the attacking contract. It shows up for the first time or does not appear this time. After the second treatment, we ask the participants to finish the survey. Afterward, we wrap up the experiment and answer any questions the participants might have.

The following section discusses the results that the described approach obtains.

7.3. Result Analysis

As a result of the experiment, we collect two different data sets. The experimenters observe whether the participant recognizes the attack with the TeSC MetaMask and with the original MetaMask. Additionally, they note open text observations about significant events. A questionnaire serves as a complementary source of data. It asks the participants about demographic factors, their previous knowledge of blockchain, and the attendants' understanding of the TeSC errors. For that reason, there are three questions: whether the user noticed an error, what s/he believes was the cause of the error, and why s/he canceled the transaction or ignored the error. This questioning line shall help us understand some design flaws in either the experiment or the user interface. Finally, we report on the general usability of MetaMask in regards to the use case of sending a transaction. For that reason, we use the System Usability Scale (SUS) as section 4.3 describes it. We discuss the results of the questionnaire and the observations in the following section.

7.3.1. Sample Demographics

Overall we test 40 participants. We recruit them from various places. We ask students of a course about blockchain at the Technical University of Munich to participate. Other respondents are associated with the Blockchain Bayern e.V. Additionally, the authors

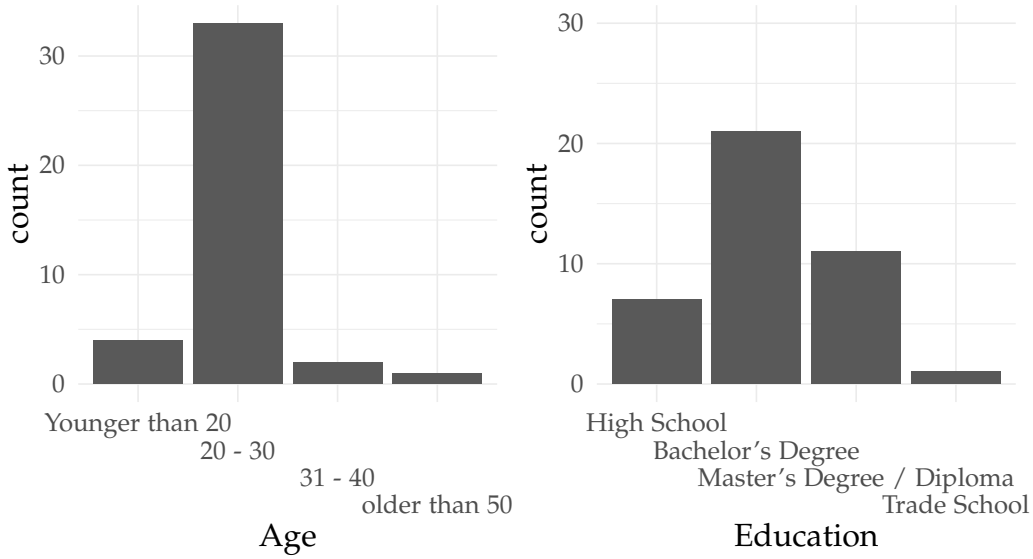

Figure 7.5.: Demographics of study sample

approach personal networks to include participants without a technical background in the sample. This inclusion facilitates a broad knowledge pool in the sample. We assume that prior experience with blockchain correlates with the ability to recognize a fraudulent Ethereum address.

Figure 7.5 shows the age distribution in our sample and the participant's education. As one can see, the sample overrepresents the age group of people between 20 and 30. The age distribution in the sample might limit the applicability of our results for older generations. Also, in terms of educational background, more than 75% of the participants have either a Bachelor's or a Master's degree, and everyone has attended high school or an advanced degree. It would be interesting to see whether a broader representation of the overall population changes the study results.

Figure 7.6 shows the participant's prior experience with blockchain. The stacked bar diagram depicts all participants' responses, whether they have ever worked with a blockchain in any way. We highlight that the group with previous experience and the group that never worked with the technology are almost the same size. We asked the 21 participants, who stated to have already some experience, how they have interacted with blockchain technologies. The chart on the right site displays their answers; three participants state that they were developing a blockchain themselves. Five explain they developed applications, which use a blockchain platform. However, the majority states they have just interacted with one.

This observation shows that the sample covers different levels of expertise. The following section discusses the efficacy of the proposed solution according to the study results.

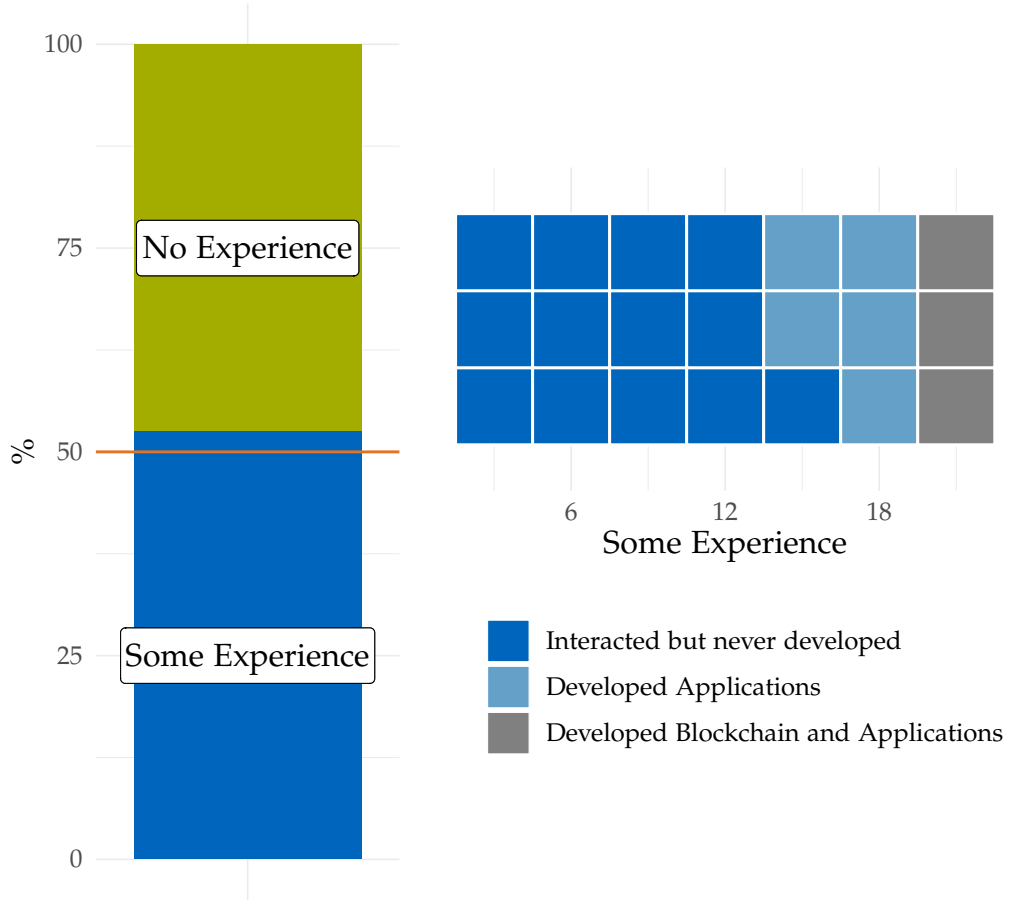

Figure 7.6.: Participant's experience with blockchain prior to the study

7.3.2. Security Performance of the TeSC Warning

We compare the security of the original MetaMask with our prototype. We are interested in the ability of the participants to notice a dangerous transaction. Thus, we measure the task conversion rate for both MetaMask instances. The task is to detect that the transaction is dangerous and cancel it. Table 7.2 shows the contingency table of the user behavior during the fraudulent transaction on both applications. Twenty-nine users disregard the dangerous transaction with TeSC but execute the transaction with the original MetaMask. Simultaneously, no one recognizes the error without TeSC but transacts to the fraudulent address when MetaMask shows a TeSC error. We see four participants who do not transact to this address with either one of the MetaMask instances. Finally, seven participants send the transaction in the original MetaMask and if the wallet shows a TeSC error. Thus, these participants would have lost their funds using the augmented wallet.

	Cancel in MetaMask with TeSC	**Confirm in MetaMask with TeSC**
Cancel in MetaMask v9.0.3	4	0
Confirm in MetaMask v9.0.3	29	7

Table 7.2.: Paired behaviour when participants encounter a fraudulent transaction. The table counts the number of participants who canceled the dangerous transaction in the augmented MetaMask and at the same time stopped the transaction when using the original MetaMask

To determine whether the different behavior between both applications is significant, we use a McNemar test statistic to determine the p-value. For a discussion about this method see section 4.3. As there are no records of a participant recognizing the error without TeSC but with TeSC s/he transacts to the address, the calculation for the p-value of the exact McNemar test is as follows.

$$p(0) = \frac{29!}{0!(29-0)!} 0.5^0 (1-0.5)^{(29-0)} = 1.862645e^{-9} \tag{7.1}$$

Thus we can reject the null hypothesis that the proportion between the two discordant pairs is 0.5 with a significance level beyond $\alpha = 0.001$. Thus, the communication of the TeSC state helps significantly to increase the security of users that attackers try tricking into transacting to a fraudulent Ethereum address.

To further analyze the performance, we estimate the probability that users will abide by a downgrade attack warning that MetaMask displays. We present two confidence intervals because there are two groups of participants: one starts the experiment by

	Lower bound	Upper bound
Starting with TeSC	57.8 %	92.5 %
Starting without TeSC	63.1 %	95.6 %

Table 7.3.: Confidence intervals of probability that users adhere to the TeSC warning

working with the original MetaMask instance, the other begins with the prototype of this thesis. Both have exposure to the augmented MetaMask instance, but the different order of treatments requires this separation.

We estimate the confidence intervals with the help of the adjusted Wald methods by Agresti and Coull [69]. The significance level is at alpha = 0.05. Table 7.3 shows the values of the confidence intervals. As one can see, both intervals spread over an area of 32.5% respectively 34.7%. A study with more participants could decrease the interval's width. In general, the results show that most participants are not willing to send a transaction on a TeSC error as the lower bound is above 50%. The upper bounds are 92.5% respectively 95.6%. The data does not suggest that we should dismiss the concept, but it is ambiguous whether the proposed solution enhances the user's security sufficiently or whether additional measures need to supplement it. It is certainly interesting to investigate this question further.

We note during the experiment that three participants return from the GreatCoin website to the PDF document with the instructions in the browser when they send the transaction. TeSC's downgrade algorithm does not show an active interrupting page if the current browser web page is not associated with a different Ethereum address. Thus, MetaMask does not show an error on transacting to the second Ethereum address for these three attendants. The participants do not recognize the passive indication and send the transaction. This limitation originates from the decision not to enforce the TeSC protocol for every transaction. However, we attribute the behavior of the participants to the experiment's design. We suspect that the conversion rate would be higher if there is a dedicated mail client where the user opens the mail. To send a transaction with MetaMask, the user must still open the browser, where the website of GreatCoin would be open. We do not exclude them from the data set, but we argue that the experiment's design might have impaired the performance measure resulting in a worse conversion rate for TeSC.

The experiment uncovers a particular user behavior. Figure 7.7 shows an observation with which UI elements the users interact to gain additional information about the TeSC warning. The figure identifies two particular elements which we deemed to be not an obvious choice, besides the two-stages warning page.

If the user hovers over the TeSC indicator, a popup provides some additional information on the current evaluation state. We consider this a *hidden feature* because the indicator's design does not reveal the popup's existence. Thus, the user must discover the popup

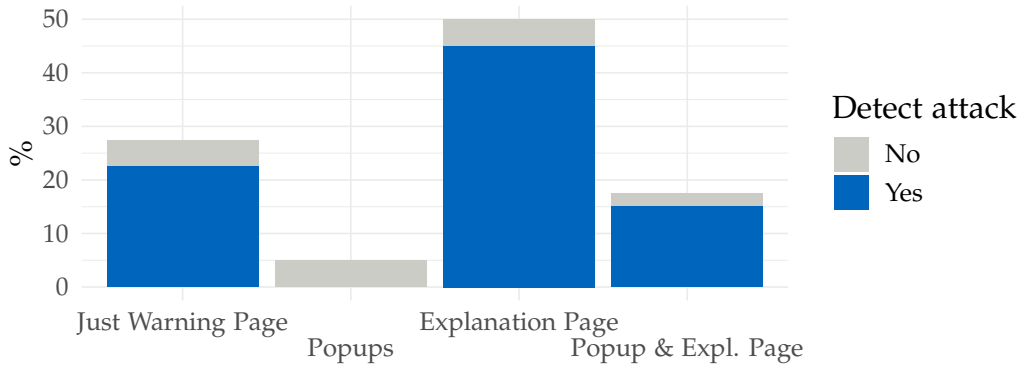

Figure 7.7.: UI elements participants access in a TeSC warning page. Participants detect an attack if they cancel the transaction.

merely by accident. The data in the figure confirms its invisibility. In total, only nine participants look at the popup. We expected such low attention for the UI element. Thus, the popup does not provide information that is not already available on the warning page.

While the popup affirms our assumption to be a feature that is seldomly noted, the explanation page attracts more interest from the participants than we assumed. Overall, 27 users open the external informational site. We mention such a page in the design concept in section 6.1, but we have not conceptualized this document as detailed as the authentication indication in MetaMask. The page provides a general explanation of TeSC and discusses technical details about the cause of each error. Its design is not yet optimized for usability, which also some participants remark. A production-grade implementation of TeSC authentication should reiterate this page.

We also observe that only eleven participants do neither use the explanation page nor the popup. These participants gain enough information from the two-stages warning sites to decide whether they want to continue with the transaction. We cannot identify one approach to be significantly better to hinder the user from transacting to the fraudulent address. Two participants confirm the transaction considering only MetaMask's warning. Additionally, two participants, who access the popup, and two participants opening the explanation page ignore the warning. One participant transacts to the fraudulent address after looking at the popup and the explanation page.

This thesis reports the system usability scale (SUS) by Brooke [49] to measure the overall usability of MetaMask. Our participants rate MetaMask with this standardized scale at the end of the experiment. Thus, the rating is not only applicable to our prototype. Instead, it describes the usability of MetaMask in the case of manually entering a transaction receiver in the UI. Figure 7.8 shows a boxplot of the participant's SUS ratings. Furthermore, the diagram depicts the individual ratings to identify any special

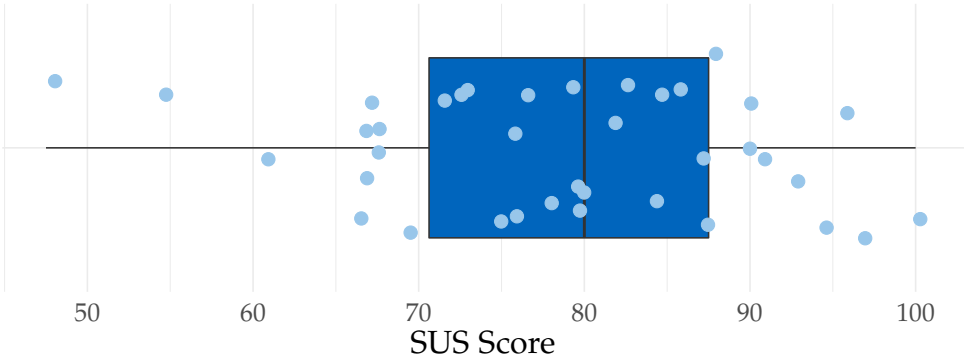

Figure 7.8.: Boxplot of SUS ratings for MetaMask

distributions. The picture does not show an extraordinary allocation though we find a tendency of the data concentrating towards the upper end of the scale. We see that the data spreads from 47 to 100; the interquartile ranges from 70.62 to 87.50. The mean value is 78.62.

To our knowledge, there is no other work that reports a SUS rating for MetaMask yet, which is why we compare it with a benchmark that Sauro provides in [50]. His meta-study evaluates other works, which use the SUS, and calculates an average value for customer-facing (B2C) products at 74.0. We test the null hypothesis that the true mean of the SUS score of the population is equal or lower than the average score of B2C products at a significance level of $\alpha = 0.05$. With student's t-test, we can reject the null hypothesis because the p-value is 0.01057. Thus, we can conclude that the overall design of MetaMask is more satisfying than already tested consumer-facing applications are.

The user study also collects feedback on the TeSC warnings' design, which the next section discusses.

7.3.3. Participants' Testimony

To understand why the participants ignored the TeSC error, we ask them to formulate their understanding of the exception and why they decide to continue with the transaction. Some participants trust the imagined friend so much that they ignore any warnings:

> "I trusted Alice's advice" - Participant 8

Others do not take the error seriously, as it is an experiment and an extraordinary situation. Sotirakopoulos already observes this problem in the context of the research of browser warnings [70]. It is not a real situation. Thus the participants do not feel the

same obligation to guard themselves against fraud:

> "I continued as that's no real ether. Myself wouldn't if that were real money" - Participant 17

One other effect, which might let the user ignore the error, applies to all participants exposed to the original MetaMask in the first run. These participants have already transacted to the fraudulent address the first time without noticing a problem, and now they see an unexpected TeSC error when they repeat the experiment. We do not tell them after the first round that they lost ether with the fraudulent transaction. Thus, they learn that this action is working as expected. Reeder et al. describe a similar user behavior in browsers: when they ask survey respondents why they ignored HTTPS warnings, 34.3 % of them argue that they visited the website previously and learned to trust the page. [34] The following quote is an example of this effect in our study:

> "Since I knew what sort of transaction I was making, I decided to continue to make the transaction by adding the website to the whitelist." - Participant 2

It is also interesting to see what those participants who cancel the transaction understand of the TeSC error. The responses vary between a general lack of comprehension of the situation, which leads to dismissing the transaction, and a thorough apprehension of the entire protocol.

Not every user has to understand the issue. A feeling of insecurity seems to be enough for some of our participants to distrust the transaction. The following answer to the question, why s/he canceled the transaction, shows this.

> "It felt unsafe." - Participant 13

However, some participants who have no prior knowledge of the TeSC protocol understand the error's technical reason. An example is the following statement to the question what the participant thinks the cause of the error to be:

> "The signer/endorser of the contract is not the same as the owner of the domain I'm connected to" - Participant 21

These testimonials supplement the previous analysis. They provide an example of what might drive the participants to a decision and why the prototype does work as expected in some instances or does not in others.

Generally, this chapter shows that a MetaMask wallet augmented with domain name based authentication is better suited to prevent users from transacting to a fraudulent address than the original open-source product. Thus, we affirm the third research question. The proposed concept enhances the user's security. During the experiment, we identify an issue with the decision that MetaMask does not enforce TeSC for every transaction. Therefore, the prototype does not show an error to three testers. The

chapter also discusses some limitations with the study design. The sample might not represent the entire population, and we do not cover all use cases and error scenarios. The sequential treatments within the same experiment might bias the participant's behavior. This bias reduces the external validity of the results. Some of the participant's testimonials show that effect. The next chapter discusses the limitations of this thesis and further work that we could not cover here.

8. Conclusion

This thesis shows that our concept enhances the users' security when they interact with Ethereum. Our design concepts accomplish to warn users of potentially dangerous transactions. We see that most of our experiment's participants adhere to the newly introduced security warnings in MetaMask. The prototype can authenticate Ethereum addresses, which are TeSC compliant. We see that significantly more participants detect a fraudulent Ethereum address if they use the authenticating MetaMask compared to the original wallet application. During the implementation, we identify one major problem. A pure browser-based certificate retrieval is only possible in Firefox. All other browsers, which MetaMask supports, do not provide access to the certificate. The limitation on one browser reduces the applicability of the proposed solution.

This chapter summarizes the thesis and contemplates its results. In the context of our research questions, we reiterate decisions and findings in section 8.1. We reflect upon the limitations of our methods and their indications on the results in section 8.2. We conclude this thesis with a discussion on open topics, which future works may investigate.

8.1. Research Questions

We structured the research process with three overarching questions. This section summarizes all results according to the research questions.

RQ1 How can the indication of domain name-based authentication be designed for MetaMask?

We analyze Firefox, Edge, and Google Chrome to identify existing design concepts for such an indication. Additionally, we review the research literature about the security warnings in browsers. The literature highlights the inefficiency of passive warnings and suggests interrupting the user with active negative warnings on possible security risks. We see a passive indication for secure scenarios in all browsers, but we notice an active interruption if the authentication fails. A remarkable case is the HTTPS downgrade. The browser does not stop the user from accessing a website that uses an older and insecure protocol. Based on these findings, we design a two-stages warning page, which MetaMask shows in case of an authentication error. The first stage provides a high-level warning, while the second stage explains the error in detail. In case of a protocol

downgrade, we propose a heuristic approach to determine the current transaction risk. Depending on the heuristic, we either interrupt the user or show a passive warning indication. If the authentication is successful, MetaMask displays the associated domain name next to the transaction receiver address in our design concept. We identify three use cases, which are candidates for transaction receiver authentication.

1. The user enters the receiver manually in the wallet,
2. a web application triggers a transaction,
3. or the transaction is an ERC-20 token exchange.

In all cases, MetaMask shows a confirmation page before it submits the transaction. We identify this page as a central place where authentication information may be helpful for the user.

RQ2 What is a feasible architecture concept to authenticate addresses in MetaMask?

The adoption of TeSC provides the means to authenticate Ethereum addresses based on domain names. We propose a verification algorithm for this protocol, which facilitates the authentication in MetaMask. The algorithm results in an erroneous state or a successful authentication state. With the implementation of this algorithm, we identify a limitation of the browser API. Only Firefox supports accessing the TLS/SSL certificate when a web extension sends a request to a domain name. Using TeSC authentication in other browsers requires secondary resources, for example, a surrogate server that is able to retrieve the certificates. The following section 8.2 discusses this approach further. The thesis outlines the implementation of the algorithm in a Firefox web extension. Our prototype shows that the design concept of RQ1 is technically feasible.

RQ3 Does the application of domain name-based authentication improve the user's security while interacting with Ethereum?

We conduct a scenario-based usability study with 40 participants to evaluate the security improvements of the augmented MetaMask. The prototype determines successfully whether the simulated transaction scenario is legitimate or potentially dangerous. The interface shows the correct indication and informs the user about the authentication state. We observe a significant improvement in the participant's ability to identify a fraudulent transaction with the authenticating MetaMask. Thus, we argue that such an authentication enhances the user's security. We are confident that our proposed solution facilitates a securer transaction environment.

8.2. Limitations

We also identify limitations in our research approach, which may reduce the explanatory power of our results. They concern the variety of the analyzed set of browsers, which

shapes our design proposal, and they affect the experiment's design.

This thesis compares TLS/SSL warning concepts in several browsers to form an underlying design inspiration. We consider different browsers because the literature highlights that there is not one browser, which stops the user from accessing fraudulent websites entirely. Thus, we identify the best practices in the most often used browsers. Due to technical limitations, we cannot include Apple's Safari in our analysis, though it is one of the three most often used browsers. This exclusion limits the variability of our design proposition. Since it runs in one specific operating system with a distinct design concept, we assume that Safari might have contributed additional insights. We hope that the following works will be able to include Safari in their analysis.

The usability experiment does only cover a specific branch of the second use case from section 5.2.3. Other cases have not been evaluated in an experiment yet. Additionally, we induce only a Downgrade Attack and test no other error scenario with participants. Due to limitations in terms of resources and number of participants, we could not investigate other scenarios in an experiment. Further work is required, which evaluates the user behavior on a more exhaustive set of scenarios.

During the entire study, it is always clear for the attendants that they are not in a real situation. The lack of reality reduces the external validity of the results. Some of the participant's testimonials show that effect, too. We expect that a field study would result in a better security performance of today's MetaMask than an artificial experiment. Thus, we interpret the performance observations as a lower bound of their real values.

Finally, we could not implement all use cases in the prototype. We focus on authenticating receivers of transactions, which either the user triggers manually in the wallet or a web application submits. Due to time constraints, we could not implement the resolution of a manually entered domain name to an Ethereum address via the TeSC registry. We have shown that the registry is accessible from MetaMask, so we assume that a subsequent development effort can implement such a feature without issues. The third use case requires a TeSC verification of an ERC-20 token transaction. This case demands executing the authentication algorithm twice: for the token receiver address and the token contract address. In general, we can show that the algorithm serves its intended purpose of authentication, but our prototype does not support the duplicated execution for this particular case yet.

Though these limitations might bias this thesis's results, we are confident that they do not invalidate the overall insights. We have to consider that this is a relatively new topic in research, and there are several aspects, which subsequent works should investigate in more detail. We provide an outline of relevant topics in the next section.

8.3. Further Work

This thesis discusses an approach to enhance the user's security by augmenting MetaMask with domain name-based authentication. During this work, some topics arose, which we cannot cover in this thesis.

This work focuses on preventing resource misspending. Transaction receiver authentication facilitates this aspect. Thus, we only focus on authentication when the user changes the blockchain's state. However, we identify another indirect security risk, which we believe authentication could mitigate. What happens if the user retrieves information from unverified blockchain addresses and acts upon that information? In general, it is essential in terms of information security to authenticate data sources. Subsequent work should discuss whether such authentication is necessary in blockchain, and they should investigate cases when such a mechanism is applicable. Additionally, it is not apparent how existing applications can integrate a reliable solution to authenticate on-chain information sources. The proposed solution in this thesis cannot prevent the side effects of reading unverified information from an unknown entity in Ethereum.

The Ethereum Name Service (ENS) facilitates the creation of domain-like names for Ethereum addresses. We suggest combining this service with TeSC. We think that the TLS/SSL-based protocol could contribute legitimate identity information to an ENS address. One problem is that ENS creates new names, which cannot be mapped to the existing domain names of the internet. We propose two approaches how to integrate both services. One suggestion is to create new certificates for ENS names. TeSC does not explicitly require the use of the internet's PKI. However, this raises the question of who the authorities are that sign these certificates. A different approach requires enhancing ENS. The naming service does currently not allow creating ENS names with existing top-level domains (e.g., addresses ending with *.com* or *.org*). This approach prevents that addresses impersonate well-known identities from the internet. We propose investigating whether this restriction can be lifted if a TeSC verification evaluates during the ENS creation process whether the existing internet domain name endorses the Ethereum address.

Finally, future works should investigate server-based certificate retrieval. This retrieval is required for chromium browsers because they do not provide the necessary means to access the certificates for a given domain name. We propose here two approaches that could be interesting to investigate further in follow-up works.

One option is that researchers build a new server application, which works as a surrogate for the certificate request. It accepts a domain name, retrieves the certificate from this domain server, and returns it to MetaMask. With this approach, the question arises who is responsible for evaluating the certificate according to the TLS validation algorithm in RFC 5280 [57]. If the server evaluates the certificate, the user must trust the server completely because s/he has no control over the evaluation process. If the server returns

a certificate to MetaMask, which is not validated, the control over the evaluation resides in the client. However, it requires the implementation of the RFC 5280 verification algorithm in MetaMask. Additionally, MetaMask has no access to the trusted certificate authorities of the browser. Thus, the wallet must ship with an exclusive set of CAs that MetaMask's publishers need to preselect. We suggest a third approach for evaluating the certificate on the client-side, which uses the browser's set of CAs. MetaMask sends an additional request to the domain and awaits a successful completion of the HTTPS handshake. However, this requires a redundant request to the endorsing domain name. The response of the domain is not relevant for the certificate evaluation. Only the HTTPS handshake has to succeed. If the request fails due to a network error, there is no further information about what renders the certificate invalid. All three methods are not ideal for establishing the vital trust in a certificate.

Additionally, a new server application introduces a single point of failure, which becomes an interesting object for attacks. Thus, we highlight another approach instead of building a new server application. We expect that the existing Certificate Transparency (CT) logs can be used for certificate retrieval. CT is a publicly available log of all certificates that participating certificate authorities have published. Its purpose is to monitor the issuance of certificates and to detect suspicious behavior. [71] Most of the major certificate authorities support this registry. Therefore, we expect that MetaMask can find all relevant certificates in the registry. The main problem is finding the correct certificate efficiently because the transparency log is a Merkle tree. Thus, searching for a certificate for a specific domain is not self-evident and must be investigated further. However, we expect that CT logs can overcome the trust issue, which a third-party server raises. We conclude that additional research is required to find a secure and reliable solution for certificate retrieval, which all the browsers support.

A. Tables for Use Cases

A.1. Use Case 1: Transaction from a Web Application

	Use Case 1
Description	Web application triggers transaction
Trigger	Web application sends transaction via ethereum.request() with one of the following RPCs: 1. eth_sendTransaction, or 2. eth_signTransaction
Preconditions	1. User is logged in MetaMask 2. The web application is allowed to use the MetaMask API
Postconditions	The user sees the authentication of the recipient in the confirmation screen
Normal Flow	1. The web application sends the transaction RPC 2. The TeSC protocol authenticates the recipient's address 3. smart contract belongs to the same domain as the current web page 4. A passive indication is shown in the user interface
Alternative Flow I	1. & 2. Ditto 3. smart contract authenticates to a different domain than current web page 4. An action message is shown to the user to verify the FQDN itself
Exceptions	All TeSC protocol errors
Assumptions	User understands the context, where the transaction originates from

Table A.1.: Use Case 1 as described in section 5.2.2

A.2. Use Case 2: Transaction Configuration in MetaMask

	Use Case 2
Description	User transfers Ether manually
Trigger	User presses the send button in the UI
Preconditions	1. User is logged in in MetaMask 2. User knows the recipient's address information
Postconditions	The user sees the authentication of the recipient in the confirmation screen
Normal Flow	1. The user enters a FQDN for the recipient 2. The FQDN resolves unambigously to a smart contract 3. The TeSC protocol authenticates the recipient's address 4. The recipient's address is successfully authenticated
Alternative Flow I	1. The user enters a FQDN for the recipient 2.a The FQDN does resolves ambiguously to multiple smart contracts 2.b The user selects the correct smart contract 3. & 4. Ditto
Alternative Flow II	1. The user enters a different addressing format 2. The TeSC protocol authenticates the recipient's address successfully 3. The confirmation screen displays the FQDN 4. Ditto
Exceptions	All TeSC protocol errors FQDN was not found in the registry
Assumptions	none

Table A.2.: Use Case 2 as described in section 5.2.3

A.3. Use Case 3: ERC-20 Transaction

For more clarity the use case is split into two tables: One for the MetaMask-only branch and the other ERC-20 token transferrals from web pages.

	Use Case 3 b
Description	Web application triggers ERC-20 token transaction
Trigger	Web application sends transaction via ethereum.request() with one of the following RPCs: 1. eth_sendTransaction, or 2. eth_signTransaction Receiver is an ERC-20 contract
Preconditions	1. User is logged in MetaMask 2. The web application is allowed to use the MetaMask API
Postconditions	The user sees the confirmation screen for token exchanges
Normal Flow	1. The web application sends the transaction RPC 2. MetaMask recognizes a contract Transaction 3. The TeSC protocol authenticates the token's contract address 4. The TeSC protocol authenticates the token's recipient address 5. The recipient's account belongs to the same domain as the current web page 4. Show the authenticated FQDN
Alternative Flow I	1. The web application sends the transaction RPC 2. MetaMask recognizes a contract Transaction 3. The TeSC protocol authenticates the token's contract address 4. The TeSC protocol authenticates the token's recipient address 5. The recipient's contract belongs to a different domain than the current web page 6. Show an action message: the user must verify the contract 7. Show the authenticated FQDN
Exceptions	- All TeSC protocol errors during token contract authentication - All TeSC protocol errors during token recipient authentication
Assumptions	User understands the context, why a transaction confirmation is shown

Table A.4.: Use Case 3b as described in section 5.2.4

	Use Case 3 a
Description	User swaps ERC-20 token from MetaMask
Trigger	User presses the send button in the UI
Preconditions	1. User is logged in MetaMask 2. The token is already configured and authenticated 3. The token is currently selected in MetaMask
Postconditions	The user sees the confirmation screen for token exchanges
Normal Flow	c.f. Normal flow in table A.2
Alternative Flow I	c.f. Alternative flow I in table A.2
Alternative Flow II	c.f. Alternative flow II in table A.2
Exceptions	c.f. Exceptions in table A.2
Assumptions	Token contract has been authenticated during its configuration

Table A.3.: Use Case 3a as described in section5.2.4

B. Flow Diagrams of Use Cases

The following diagrams are derived from the flows that are also described in the tables in the appendix A. For the last use case in appendix A.3 the diagram is omitted, because it does not differ from figure B.2.

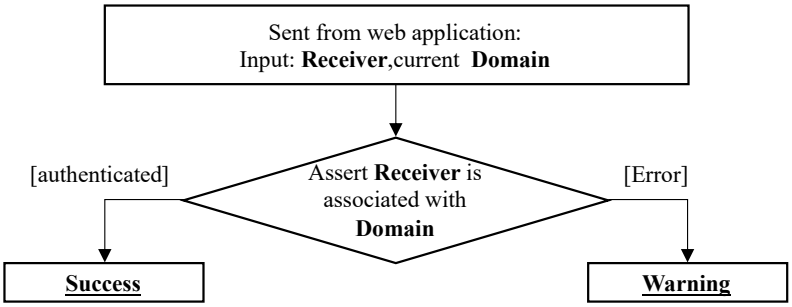

Figure B.1.: Flow of use case 1 in section 5.2.2

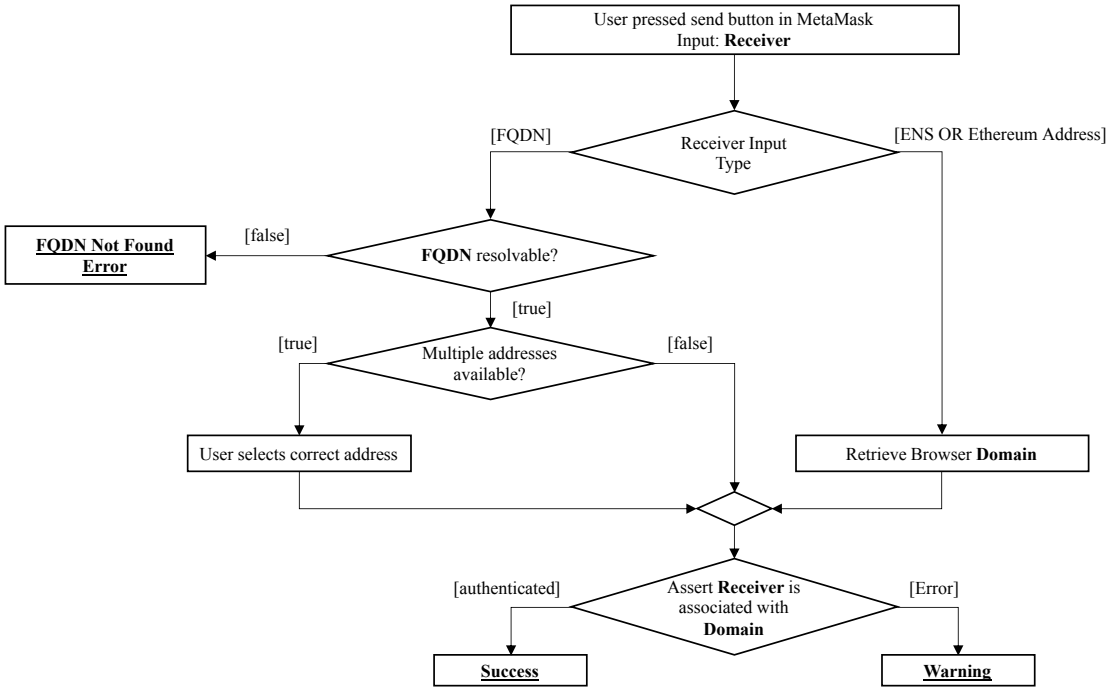

Figure B.2.: Flow of use case 2 in section 5.2.3

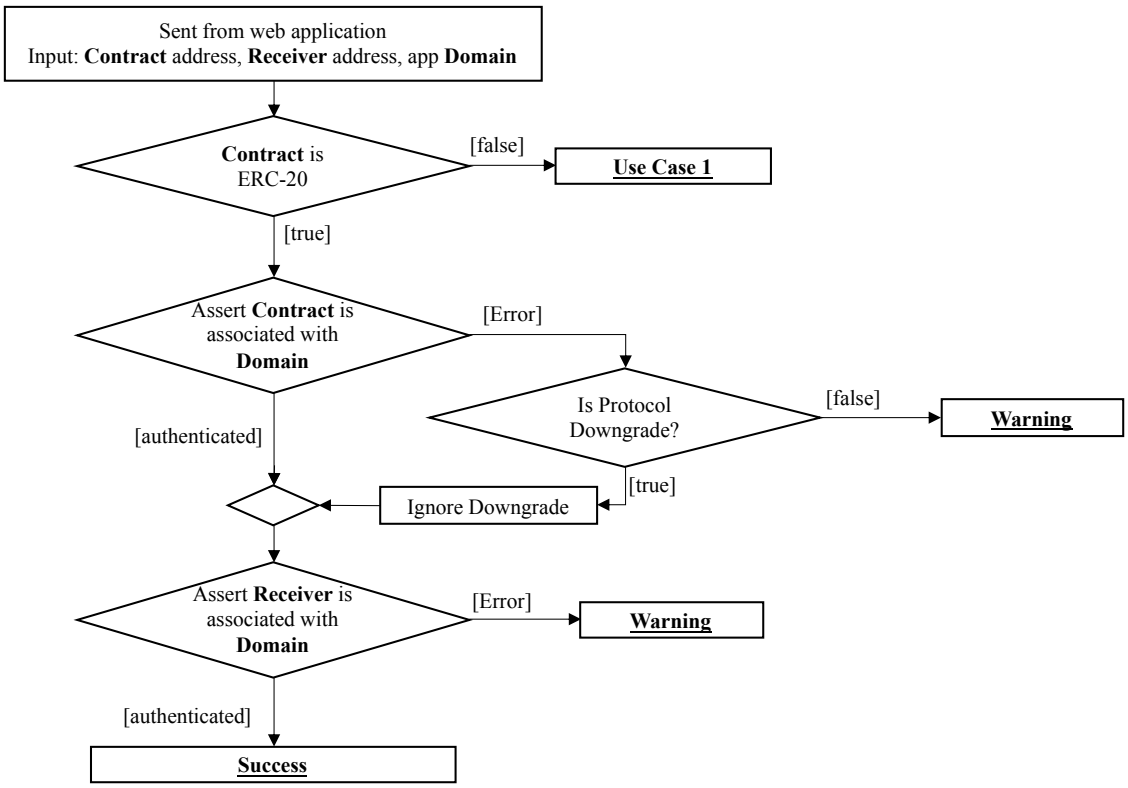

Figure B.3.: Flow of use case 3b in section 5.2.4

C. Error Scenarios in RFC 5280

The following table details where each error is discussed in the RFC 5280.

RFC 5208	**Page**
AnyPolicy uncompliance	84
Chain broken	19 ff, 73, 80
Chain depth error	39, 80, 87
Invalid signature	18, 80
Key usage mismatch	44 f, 87
Name constraint error	40, 80
Non-compliance	General Uncompliance with RFC 5280
Policy constraint	43, 73, 84
Policy mismatch	32 ff, 73, 84
Revoked	45 ff, 80
Unknown extension	26, 87, 88
Untrusted anchor	73
Validity expired	22 f
Version error	19, 80
Wrong subject	23 ff, 35 ff

Table C.1.: Certificate errors defined in RFC 5280 and the page of their occurrence

D. EV-Certificate Indication

Figure D.1 displays Firefox's dialog with an EV-certificate; Figure D.2 shows the altered dialogs of Chrome and Edge.

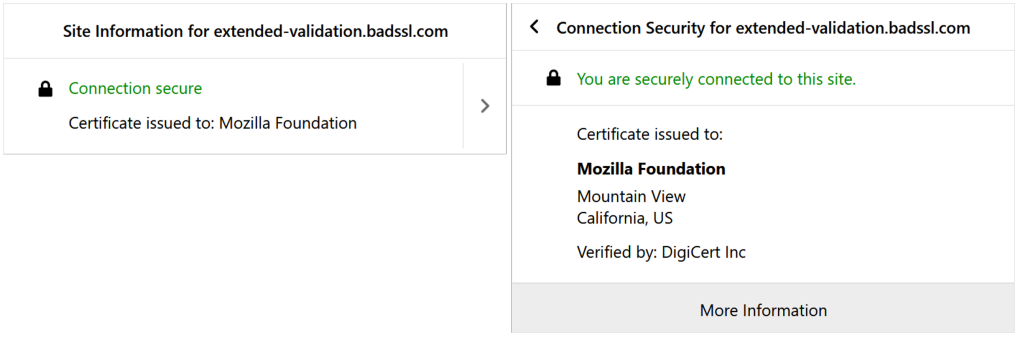

Figure D.1.: The popup with an EV-certificate in Firefox

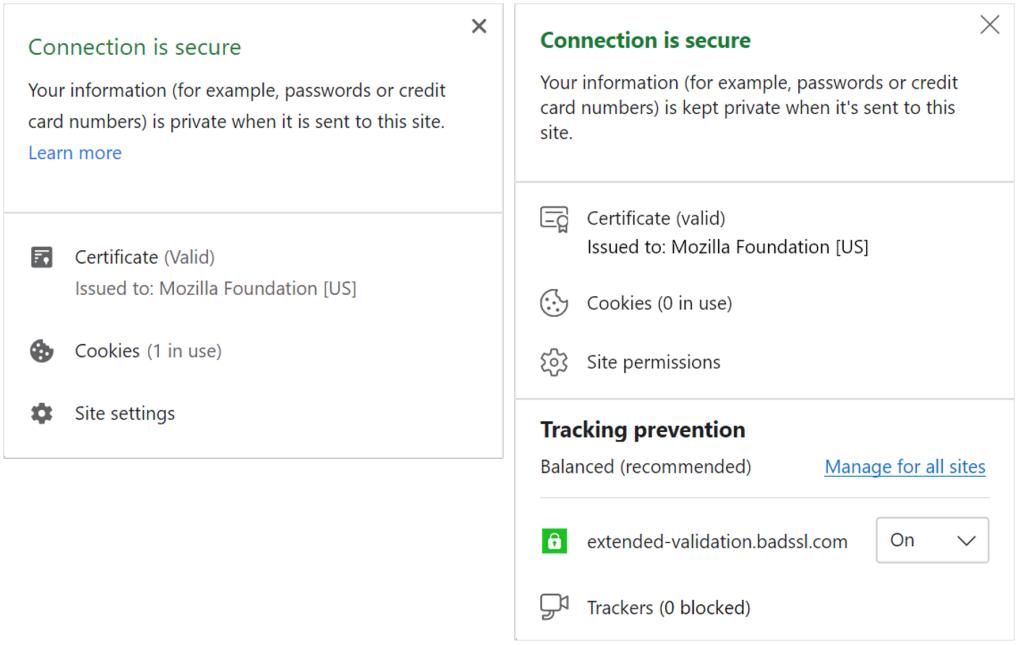

Figure D.2.: The popup with an EV-certificate in Chrome and Edge

E. Analysis of Firefox

	Yellow Frame	Two stages with proceed	Headline	Server caused error	Client caused error	Scenario: Proceed is clicked	User resolvance	Technical cause
Any Policy Error			x	x			x	x
Chain broken			x	x			x	x
Chain depth error			x	x			x	x
Invalid Signature			x	x			x	x
Key usage mismatch			x	x			x	x
Name constraint error			x	x			x	x
Revoked			x	x			x	x
Unknown extension			x	x			x	x
Untrusted anchor	x	x	x			x	x	x
Validity expired	x	x	x	x	x	x	x	x
Version error	x	x	x	x		x		x
Wrong subject	x	x	x			x	x	x
Policy constraint	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Policy mismatch	No example available	No example available	No example available	No example available	No example available	No example available	No example available	No example available

Table E.1.: User interface elements in Firefox on certificate errors

F. Analysis of Chrome

Chrome error code	**RFC Error**	**Stage 2 explanation text**
Date Invalid	Validity expired	This server could not prove that it is expired.badssl.com; its security certificate expired 2,061 days ago. This may be caused by a misconfiguration or an attacker intercepting your connection. Your computer's clock is currently set to Wednesday, December 2, 2020. Does that look right? If not, you should correct your system's clock and then refresh this page. Proceed to expired.badssl.com (unsafe)
Authority Invalid	Chain broken, Untrusted anchor, Invalid signature, Chain depth error	This server could not prove that it is untrusted-root.badssl.com; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection. Proceed to untrusted-root.badssl.com (unsafe)
Revoked Certificate	Revoked	revoked.badssl.com normally uses encryption to protect your information. When Google Chrome tried to connect to revoked.badssl.com this time, the website sent back unusual and incorrect credentials. This may happen when an attacker is trying to pretend to be revoked.badssl.com, or a Wi-Fi sign-in screen has interrupted the connection. Your information is still secure because Google Chrome stopped the connection before any data was exchanged. You cannot visit revoked.badssl.com right now because its certificate has been revoked. Network errors and attacks are usually temporary, so this page will probably work later.

Table F.1 continued from previous page

Chrome error code	RFC Error	Stage 2 explanation text
Weak signature algorithm	deprecated hashing algorithm	You attempted to reach sha1-intermediate.badssl.com, but the server presented a certificate signed using a weak signature algorithm (such as SHA-1). This means that the security credentials the server presented could have been forged, and the server may not be the server you expected (you may be communicating with an attacker). Proceed to sha1-intermediate.badssl.com (unsafe)
Certificate Invalid	Key Usage, Name constraint, Anypolicy, Unknown extension, Version error, Policy Constraint,	testingInvalidCert.example.com normally uses encryption to protect your information. When Google Chrome tried to connect to testingInvalidCert.example.com this time, the website sent back unusual and incorrect credentials. This may happen when an attacker is trying to pretend to be testingInvalidCert.example.com, or a Wi-Fi sign-in screen has interrupted the connection. Your information is still secure because Google Chrome stopped the connection before any data was exchanged. You cannot visit testingInvalidCert.example.com right now because the website sent scrambled credentials that Google Chrome cannot process. Network errors and attacks are usually temporary, so this page will probably work later.
Common Name Invalid	Wrong host	This server could not prove that it is wrong.host.badssl.com; its security certificate is from *.badssl.com. This may be caused by a misconfiguration or an attacker intercepting your connection. Proceed to wrong.host.badssl.com (unsafe)

Table F.1.: Certificate validation errors in Chrome with technical explanation assigned to RFC errors from table 2.1

G. Error Messages in TeSC

Error Type	Overridable	Error message
Endorsement Expired	Yes	The address 0x0 could not prove its identity. The identity information expired. It was valid until 2.956 days ago. This may be caused by a misconfiguration or you're sending the transaction to the wrong address. Your computer's clock is currently set to Wednesday, December 2, 2020. Does that look right? If not, you should correct your system's clock and then retry the transaction. Proceed with transaction (unsafe).
No certificate Error	No	The address 0x0 could not prove its identity. No certificate was found for the domain example.com. This may be caused by a network error or you're sending the transaction to the wrong address. You cannot execute this transaction. Network errors and attacks are usual temporary, so this will probably work later.
Endorsement Tampered	No	The address 0x0 could not prove its identity. The signature of the identity proof is broken. This means that someone has altered the information and it cannot be trusted anymore. You cannot execute this transaction. Please contact the owner of this address.

TeSC protocol errors: Table G.1 continued from previous page

Protocol Downgrade Attack	Yes	*(address.domain == null)* ? [The address at 0x0 has no proof of identity.] : [The address at 0x0 belongs to the wrong domain.] *IF (domain!= null)* [The address 0x0 is associated with the domain example.com.] *IF (referencepage.addresses != null)* [The website in your browser is app.uniswap.com. It owns the following Ethereum addresses: - 0x01 - 0x02] MetaMask prevents you from sending the transaction to a different domain than the current website, because an attacker might have altered the Ethereum address and may try to steal your funds. As a default MetaMask requires same-domain transactions. You can whitelist the current browser page if you want to allow payments to other domains, too. Whitelist domain and proceed.
Exclusiveness	No	The address 0x0 could not prove that it belongs to example.com. This domain can only be used for one address, but our records show that the domain is used also with these other Ethereum addresses: - 0x01 - 0x02 You cannot execute this transaction. Please contact the owners of this domain and the addresses.

TeSC protocol errors: Table G.1 continued from previous page

General Error	No	The address 0x0 could not prove its identity. The smart contract's configuration is not correct. It looks like the owner of the address tries to prove its ownership, but something went wrong during the configuration. This means that the identity cannot be verified. You cannot execute this transaction. Please contact the owner of this address with this error code and try again later.

Table G.1.: Error messages during TeSC protocol execution

Certificate Error Type	**Overridable**	**Error message**
Non-Compliance	No	The Ethereum address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received an unusual and incorrect certificate. This may happen when an attacker is trying to pretend to be example.com, or your network connection is not correct. You cannot execute this transaction. Network errors and attacks are usual temporary, so this will probably work later.
Chaining Error	Yes	This address could not prove that it belongs to example.com. The identity proof is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection. Proceed with transaction (unsafe)

TeSC certificate errors: Table G.2 continued from previous page

Tampered Error	No	This address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received a certificate with a broken signature. The transaction cannot be executed, because an attacker might have altered the identity certificate. You cannot execute this transaction. Please contact the owners of example.com to inform them of this problem.
Usage Error	No	This address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received a certificate that must not be used for authentication. This may happen due to misconfiguration, or when an attacker is trying to pretend to be example.com. You cannot execute this transaction. Please contact the owners of example.com to inform them of this problem.
Subject Error	Yes/No	This address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received a certificate that is not valid for example.com. The certificate can only be valid for the following names: *.badssl.com This may be caused by a misconfiguration or an attacker intercepting your connection. *IF (error==wrong host)* [Proceed with transaction] *IF (error==nameconstraint)* [You cannot execute this transaction. Please contact the owners of example.com to inform them of this problem.]

TeSC certificate errors: Table G.2 continued from previous page

Validity State Error	Yes/No	This address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received a certificate that is not valid. *IF (error==revoked)* [You cannot executed this transaction, because the certificate has been revoked. Please contact the owners of example.com to inform them of this problem] ELSE [The certificate expired xxx days ago. This may be caused by a misconfiguration or an attacker intercepting your connection. Your computer's clock is currently set to Wednesday, December 2, 2020. Does that look right? If not, you should correct your system's clock and then refresh this page. Proceed with transaction]
General Error	No	This address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received an unusual and incorrect certificate. This may happen when an attacker is trying to pretend to be example.com, or your network connection is not correct. You cannot execute this transaction. Network errors and attacks are usual temporary, so this will probably work later.
Hash Error	Yes	This address could not prove that it belongs to example.com. When MetaMask tried to connect with this domain, it received a certificate signed using a weak signature algorithm (such as SHA-1). This means that the security credentials the server presented could have been forged, and the server may not be the server you expected (you may be communicating with an attacker). Proceed with transaction (unsafe).

Table G.2.: Certificate error messages during TeSC execution

List of Figures

List of Tables

Bibliography

[1] W. Zhao. "$7 Million Lost in CoinDash ICO Hack - CoinDesk". In: *CoinDesk* (July 17, 2017). URL: https://www.coindesk.com/7-million-ico-hack-results-coindash-refund-offer (visited on 09/05/2020).

[2] U. Gallersdörfer and F. Matthes. *AuthSC: Mind the Gap between Web and Smart Contracts*. 2020. URL: https://arxiv.org/pdf/2004.14033.

[3] A. M. Antonopoulos and G. A. Wood. *Mastering Ethereum. Building smart contracts and DApps*. 2018.

[4] M. Becze, H. Jameson, and et al. *EIP-1: EIP Purpose and Guidelines*. 2015. URL: https://eips.ethereum.org/EIPS/eip-1 (visited on 03/31/2021).

[5] F. Vogelsteller and V. Buterin. *EIP-20: ERC-20 Token Standard*. 2015. URL: https://eips.ethereum.org/EIPS/eip-20 (visited on 11/14/2020).

[6] Etherscan. *Token Tracker*. 2020. URL: https://etherscan.io/tokens (visited on 11/14/2020).

[7] C. Reitwießner, N. Johnson, F. Vogelsteller, et al. *ERC-165 Standard Interface Detection*. 2018. URL: https://eips.ethereum.org/EIPS/eip-165 (visited on 03/23/2021).

[8] IEC International Electrotechnical Commission, ed. *Information technology - Open Systems Interconnection - The Directory. Part 8: Public-key and attribute certificate frameworks*. ISO/IEC 9594-8. International Organization for Standardization, 2017.

[9] R. Housley, W. Ford, W. Polk, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. RFC Editor, 1999. URL: https://tools.ietf.org/html/rfc2459 (visited on 04/10/2021).

[10] D. Cooper, S. Santesson, S. Farrel, et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC Editor, 2008. URL: https://tools.ietf.org/html/rfc5280 (visited on 01/07/2021).

[11] M. E. Acer, E. Stark, A. P. Felt, et al. "Where the Wild Warnings Are". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Ed. by B. Thuraisingham. New York, NY: Association for Computing Machinery, 2017, pp. 1407–1420.

[12] U. Gallersdörfer. *Smart Contract X.509 Identity Standard*. URL: https://github.com/TeSC-app/TeSC-EIP (visited on 04/10/2021).

[13] P. Herrmann, T. A. Ma, D. Stübing, and M. Manov. *TLS-based Authentication Management on Blockchain*. Munich: Seba Lab Practical Course, 2021. URL: https://github.com/TeSC-app/TeSC-api-server.

[14] O. Avellaneda, A. Bachmann, A. Barbir, et al. "Decentralized Identity: Where Did It Come From and Where Is It Going?" In: *IEEE Communications Standards Magazine* 3.4 (2019), pp. 10–13.

[15] D. Reed, M. Sprony, D. Longley, et al. *Decentralized Identifiers (DIDs) v1.0*. Ed. by D. Reed, M. Sporny, and M. Sabadello. Version W3C Working Draft. W3C, 2020. URL: https://www.w3.org/TR/2020/WD-did-core-20201108 (visited on 11/27/2020).

[16] J. Bernal Bernabe, J. L. Canovas, J. L. Hernandez-Ramos, et al. "Privacy-Preserving Solutions for Blockchain: Review and Challenges". In: *IEEE Access* 7 (2019), pp. 164908–164940.

[17] N. Johnson. *ENS Is Upgrading — Here's What You Need to Do*. Ethereum Name Service. 2019. URL: https://medium.com/the-ethereum-name-service/ens-is-upgrading-heres-what-you-need-to-do-f26423339fcf (visited on 11/26/2020).

[18] M. Inoue. *State of the ENS 2020*. Ethereum Name Service. 2020. URL: https://medium.com/the-ethereum-name-service/state-of-the-ens-2020-cd8afa19f59d (visited on 11/26/2020).

[19] V. Buterin and A. Van de Sande. *EIP-55: Mixed-case checksum address encoding*. Ethereum Improvement Proposals. 2016. URL: https://eips.ethereum.org/EIPS/eip-55 (visited on 11/25/2020).

[20] A. Garba, Z. Guan, A. Li, and Z. Chen. "Analysis of Man-In-The-Middle of Attack on Bitcoin Address". In: *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications*. Institute for Systems and Technologies of Information, Control and Communication. Porto, Portugal: SCITEPRESS - Science and Technology Publications Lda, 2018, pp. 388–395.

[21] E. Dachtler, P. Szilágyi, D. Seago, et al. *Blockies*. Version 0.1.1. Github: Ethereum, 2014. URL: https://github.com/ethereum/blockies (visited on 11/26/2020).

[22] A. E. Salih, M. Elsherif, M. Ali, et al. "Ophthalmic Wearable Devices for Color Blindness Management". In: *Advanced Materials Technologies* 5.8 (2020), p. 1901134.

[23] T. Dierks and C. Allen. *The TLS Protocol. Version 1.0*. USA: RFC Editor, 1999. URL: https://tools.ietf.org/html/rfc2246 (visited on 10/27/2020).

[24] E. Rescorla. *HTTP Over TLS*. RFC Editor, 2000. URL: https://tools.ietf.org/html/rfc2818 (visited on 04/10/2021).

[25] R. Dhamija, J. D. Tygar, and M. Hearst. "Why phishing works". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Ed. by R. Grinter. ACM Special Interest Group on Computer-Human Interaction. New York, NY: ACM, 2006, p. 581.

[26] J. Sobey, R. Biddle, P. C. van Oorschot, and A. S. Patrick. "Exploring User Reactions to New Browser Cues for Extended Validation Certificates". In: *Computer Security - ESORICS*. Ed. by S. Jajodia and J. Lopez. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 411–427.

[27] Max-Emanuel Maurer, Alexander De Luca, and Tobias Stockinger. "Shining Chrome: Using Web Browser Personas to Enhance SSL Certificate Visualization". In: *Human-Computer Interaction – INTERACT*. Ed. by P. Campos, N. Graham, J. Jorge, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 44–51.

[28] C. Bravo-Lillo, L. F. Cranor, J. Downs, et al. "Improving Computer Security Dialogs". In: *Human-Computer Interaction – INTERACT*. Ed. by P. Campos, N. Graham, J. Jorge, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 18–35.

[29] C. Bravo-Lillo, L. F. Cranor, J. Downs, and S. Komanduri. "Bridging the Gap in Computer Security Warnings: A Mental Model Approach". In: *IEEE Security & Privacy Magazine* 9.2 (2011), pp. 18–26.

[30] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. "The SSL landscape. A thorough analysis of the x.509 PKI using active and passive measurements". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. Ed. by P. Thiran. ACM Special Interest Group on Measurement and Evaluation. New York, NY: ACM, 2011, p. 427.

[31] D. Akhawe and A. P. Felt. "Alice in warningland: A large-scale field study of browser security warning effectiveness". In: *22nd USENIX Security Symposium*. Ed. by S. King. Berkeley, Calif.: Usenix Association, 2013, pp. 257–272.

[32] *HTTPS-Verschlüsselung im Web*. Google Ireland Limited. URL: `https://transparencyreport.google.com/https/overview` (visited on 11/18/2020).

[33] A. P. Felt, R. Barnes, A. King, et al. "Measuring HTTPS Adoption on the Web". In: *Proceedings of the Second Workshop on Real, Large Distributed Systems*. USENIX Association. 2017, pp. 1323–1338.

[34] R. W. Reeder, A. P. Felt, S. Consolvo, et al. "An Experience Sampling Study of User Reactions to Browser Warnings in the Field". In: *Engage with CHI*. Ed. by R. Mandryk and M. Hancock. New York, New York: The Association for Computing Machinery, 2018, pp. 1–13.

[35] C. Thompson, M. Shelton, E. Stark, et al. "The Web's Identity Crisis: Understanding the Effectiveness of Website Identity Indicators". In: *Proceedings of 28th USENIX. Security Symposium*. 2019, pp. 1715–1732.

[36] L. Jelovčan, S. Vrhovec, and A. Mihelič. "A literature survey of security indicators in web browsers". In: *Elektrotehniški vestnik* 87.1-2 (2020), pp. 31–38.

[37] A. P. Felt, A. Ainslie, R. W. Reeder, et al. "Improving SSL Warnings". In: *Proceedings of the 33rd Annual CHI Conference on Human Factors in Computing Systems* (). Ed. by J. Kim. New York, NY: ACM, 2015, pp. 2893–2902.

[38] M. Stojmenović, T. Oyelowo, A. Tkaczyk, and R. Biddle. "Building Website Certificate Mental Models". In: *Persuasive technology*. Ed. by J. Ham, E. Karapanos, P. P. Morita, and C. M. Burns. Vol. 10809. Lecture Notes in Computer Science 10809. Cham: Springer, 2018, pp. 242–254.

[39] C. L. X. Yi, Z. F. Zaaba, and M. A. I. M. Aminuddin. "Appraisal on User's Comprehension in Security Warning Dialogs: Browsers Usability Perspective". In: *Advances in Cyber Security* (2020). Ed. by M. Anbar, N. Abdullah, and S. Manickam. Communications in Computer and Information Science. Singapore: Springer Singapore, 2020, pp. 320–334.

[40] Hevner, March, Park, and Ram. "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1 (2004), p. 75.

[41] A. R. Hevner. "A Three Cycle View of Design Science Research". In: *Scandinavian Journal of Information Systems*. Vol. 19. 2007.

[42] A. Herwix and C. Rosenkranz. "Making Sense of Design Science in Information Systems Research: Insights from a Systematic Literature Review". In: *Designing for a Digital and Globalized World* (2018). Ed. by S. Chatterjee, K. Dutta, and R. P. Sundarraj. Cham: Springer International Publishing, 2018, pp. 51–66.

[43] K. E. Wiegers and J. Beatty. *Software requirements*. 3. ed. Best practices. Redmond, Wash.: Microsoft Press, 2013.

[44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns. Elements of resuable object-oriented software*. England: Pearson education Limited, 1995.

[45] J. Sauro and J. R. Lewis. *Quantifying the user experience. Practical statistics for user research*. 2nd edition. Cambridge: Morgan Kaufmann, 2016.

[46] A. Parush. *Conceptual design for interactive systems. Designing for performance and user experience*. First edition. Waltham, MA: Morgan Kaufmann, 2015.

[47] S. Strahringer. "Konzeptuelle Modellierung von IS". In: *Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon*. Ed. by N. Gronau, J. Becker, N. Kliewer, et al. 11th ed. GITO, 2013. URL: www.enzyklopaedie-der-wirtschaftsinformatik.de (visited on 03/03/2021).

[48] J. Nielsen. *Usability engineering*. Interactive Technologies. Amsterdam: Kaufmann, 2010.

[49] J. Brooke. "Sus: a 'quick and dirty' usability scale". In: *Usability evaluation in industry* 189 (1996).

[50] J. Sauro. *A practical guide to the system usability scale. Background, benchmarks & best practices*. Denver Col.: Measuring Usability LLC, 2011.

[51] *Market share comparison of the leading browser families for internet usage in Germany from January 2009 to September 2021*. StatCounter. 2021. URL: https://www-statista-com.eaccess.ub.tum.de/statistics/461879/browsers-market-share-germany/ (visited on 04/09/2021).

[52] L. Garron, A. King, J. Burton, et al. *badssl.com*. Github: The Chromium Project, 2020. URL: https://github.com/chromium/badssl.com (visited on 03/05/2021).

[53] *HTTPS Interception Weakens TLS Security*. Cybersecurity & Infrastructure Security Agency. 2017. URL: https://us-cert.cisa.gov/ncas/alerts/TA17-075A (visited on 12/02/2020).

[54] A. Vance, J. L. Jenkins, B. B. Anderson, et al. "Improving Security Behavior Through Better Security Message Comprehension: fMRI and Eye-Tracking Insights". In: *Information Systems and Neuroscience* (2019). Ed. by F. D. Davis, R. Riedl, J. vom Brocke, et al. Cham: Springer International Publishing, 2019, pp. 11–17.

[55] W. Dormann. *Effects of HTTPS and SSL inspection on the client*. Version Vulnerability Analysis. CERT Coordination Center. 2017. URL: vuls.cert.org (visited on 12/02/2020).

[56] B. Kaduk, B. Edlinger, D. Benjamin, et al. *OpenSSL. Cryptography and SSL/TLS Toolkit*. Version 3.0. OpenSSL Software Foundation, 2021. URL: https://www.openssl.org/.

[57] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC Editor, 2018. URL: https://tools.ietf.org/html/rfc8446 (visited on 04/10/2021).

[58] B. Smith. *Implement certificate policy constraints extension*. Ed. by D. Keeler. 2014. URL: https://bugzilla.mozilla.org/show_bug.cgi?id=975877 (visited on 03/11/2021).

[59] J. Hodges, C. Jackson, and A. Barth. *HTTP Strict Transport Security (HSTS)*. Ed. by RFC Editor. 2012. URL: https://tools.ietf.org/html/rfc6797 (visited on 04/10/2021).

[60] C. Ward. *JSON RPC API. Ethereum Wiki*. URL: https://eth.wiki/json-rpc/API (visited on 11/02/2020).

[61] A. Akers, D. Finlay, E. Marks, et al. *RPC API. MetaMask Docs*. URL: https://docs.metamask.io/guide/rpc-api.html (visited on 11/02/2020).

[62] D. Finlay, E. Miño, E. Marks, et al. *contract-metadata*. Version 1.17.0. Github: Metamask, 2020. URL: https://github.com/MetaMask/contract-metadata (visited on 11/16/2020).

[63] D. Finlay, A. Davis, M. J. Miller, et al. *metamask-extension*. Version 9.0.3. Github: Consensus Software Inc., 2020. URL: https://github.com/MetaMask/metamask-extension (visited on 03/18/2021).

[64] *MetaMask. Download Page*. ConsenSys Software Inc. URL: https://metamask.io/download.html (visited on 03/17/2021).

[65] *chrome.webRequest. Extension API Reference*. Google Ireland Limited. 2021. URL: https://developer.chrome.com/docs/extensions/reference/webRequest/ (visited on 03/17/2021).

[66] *WebExtensions API. webRequest*. Mozilla. 2021. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest (visited on 03/17/2021).

[67] Y. Qing. *fidm/x509*. Github: Federated Identity Management, 2019. URL: https://github.com/fidm/x509.

[68] D. Abramov, A. Clark, M. Erikson, et al. *Redux*. 2021. URL: https://redux.js.org/ (visited on 03/18/2021).

[69] A. Agresti and B. A. Coull. "Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions". In: *The American Statistician* 52.2 (1998), p. 119.

[70] A. Sotirakopoulos, K. Hawkey, and K. Beznosov. "On the challenges in usable security lab studies". In: *Proceedings of the Seventh Symposium on Usable Privacy and Security*. Ed. by L. F. Cranor. ACM Special Interest Group on Computer-Human Interaction. New York, NY: ACM, 2011, p. 1.

[71] *Certificate Transparency. How CT works*. Google Ireland Limited. 2021. URL: https://certificate.transparency.dev/howctworks/ (visited on 03/17/2021).