
Eötvös Loránd University

Faculty of Informatics

Algorithms and Applications Department

# Post-Quantum Signed Banking Transaction Protocol

*Supervisor:*

Szűcs Ádám István

MSc, PhD candidate

*Authors:*

Richárd Antal, Nagy

Krisztofer Zoltán, Horvát

Cybersecurity MSc

*Budapest, 2023*

# EÖTVÖS LORÁND UNIVERSITY

**FACULTY OF INFORMATICS**

# Thesis Registration Form

**Student's Data:**

**Student's Name:** Nagy Richárd Antal

**Student's Neptun code:** V7BFDU

**Course Data:**

**Student's Major:** programtervező informatikus, mesterképzés (MA/MSc)

I have an internal supervisor

***Internal Supervisor's Name:*** *Szűcs Ádám István*

*Supervisor's Home Institution:* ***Department of Computer Algebra***

*Address of Supervisor's Home Institution:* ***1117, Budapest, Pázmány Péter sétány 1/C.***

*Supervisor's Position and Degree:* *MSc, PhD candidate*

**Thesis Title:** Post-Quantum Signed Banking Transaction Protocol

**Topic of the Thesis:**

*(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis. )*

**More and more online purchases revolve around subscription models with the implicit trust that the vendors will take the correct amount of money each time. Canceling subscriptions usually involves a tedious process of answering surveys and finding the buttons hidden on the vendor's page. They might store those banking details and use them afterward, making them a target for hackers. With card payments, it is trivial to steal funds or impersonate users in case of a data breach.**

**This thesis aims to create a new protocol for a unified cardless banking standard. In this standard, tokens are issued for vendors allowing them to use each for pre-approved transactions. Post-quantum encryption standards and exchange schemes for future-proof operations secure these transaction verifications. This standard will replace bank cards with a QR code and the customer's mobile authenticator. Furthermore, vendors can deactivate the issued tokens in case of a breach or if users would like to unsubscribe from services.**

**In this thesis, I will compare our work to other authentication protocols and improve it based on different solutions for general use cases. Finally, I will build the final protocol design and create a banking site for demonstration purposes.**

Budapest, 2022. 12. 01.

# EÖTVÖS LORÁND UNIVERSITY

**FACULTY OF INFORMATICS**

# Thesis Registration Form

**Student's Data:**

**Student's Name:** Horvát Krisztofer Zoltán

**Student's Neptun code:** NFJXDY

**Course Data:**

**Student's Major:** programtervező informatikus, mesterképzés (MA/MSc)

I have an internal supervisor

***Internal Supervisor's Name:*** *Szűcs Ádám István*

*Supervisor's Home Institution:* ***Department of Computer Algebra***

*Address of Supervisor's Home Institution:* ***1117, Budapest, Pázmány Péter sétány 1/C.***

*Supervisor's Position and Degree:* *MSc, PhD candidate*

**Thesis Title:** Post-Quantum Signed Banking Transaction Protocol

**Topic of the Thesis:**

*(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis. )*

**More and more online purchases revolve around subscription models with the implicit trust that the vendors will take the correct amount of money each time. Canceling subscriptions usually involves a tedious process of answering surveys and finding the buttons hidden on the vendor's page. They might store those banking details and use them afterward, making them a target for hackers. With card payments, it is trivial to steal funds or impersonate users in case of a data breach.**

**This thesis aims to create a new protocol for a unified cardless banking standard. In this standard, tokens are issued for vendors allowing them to use each for pre-approved transactions. Post-quantum encryption standards and exchange schemes for future-proof operations secure these transaction verifications. This standard will replace bank cards with a QR code and the customer's mobile authenticator. Furthermore, vendors can deactivate the issued tokens in case of a breach or if users would like to unsubscribe from services.**

**In this thesis, I will be researching and comparing other banking protocols with their advantages and shortcomings to our own. I will use this knowledge to improve and analyze our protocol for breaking issues. I will also prepare a vendor website for demonstration.**

Budapest, 2022. 12. 01.

# Contents

# Chapter 1

# Introduction

## 1.1 History

As the online purchase of goods and services becomes the norm, online fraud is proving to be an ever-increasing business for criminals. According to the Nilson Report[1], $25.58 billion was lost to credit card and ATM fraud in 2020. This figure is larger than the median GDP of countries, meaning that credit card fraud is larger than the entire economy of more than half of the countries in the world.

Trading happens with the implicit trust that, given one's banking information, the vendors will take the correct amount of money each time while not storing the card details without authorization. Canceling subscriptions usually involves a tedious process of answering surveys and finding the suitable options hidden on the vendor's page. They might also store those banking details and use them afterward, increasing them as a target for potential hackers. With card payments, stealing funds or impersonating users in case of a breach is trivial.

Money started to move digital increasingly in the 1960s with the invention and adoption of magnetic stripe digital cards issued by banks. This technology was further improved with PINs and wireless chips. Nowadays, people use the numbers on their cards that are entered into websites to authorize transactions. This method has a range of flaws regarding security, starting from using only a couple of guessable numbers to having to enter them into input fields into any websites that may or may not have nefarious code running on them.

Credit card information is also susceptible to data breaches. In the event of a

breach in the vendor's system, all credit card information may be lost and sold on the Internet. It may then be purchased by bad actors and used to purchase usually physical goods. The attackers have secured the items when the fraud becomes apparent, and a chargeback is issued. This is also an issue if a person loses their wallet with their credit card. There will be a bit more details on the security of credit cards in the Credit Card Entropy (5) section.

With this in mind, we can identify several issues with the current online way of managing purchase authorization:

1. Credit cards have relatively low entropy compared to modern cryptographic standards, which are considered secure.

2. Entering information manually to websites might lead to phishing attacks if a website is hijacked or a similar website is shown.

3. Breaches of vendor data might lead to exposed credit card numbers, which may be used and require ordering a new card to prevent.

4. Card details are used multiple times; any website might use it to charge it multiple times.

5. The buyer might not immediately be aware of the precise charge they are authorizing and lack the control to limit it by the transaction.

6. Man-in-the-middle attacks can be performed relatively easily against credit card purchases.

7. Credit card purchases inherently lack the option to use multiple authentication factors as the implementation or lack thereof depends on the issuer bank.

8. The customers might not have an immediate way to block transactions from specific vendors or cancel subscriptions on the bank's side.

9. Entering credit card details repeatedly might lead to frequent users saving them in insecure locations for easy pasting.

With this project, we aim to provide a solution or significant improvement to all the issues mentioned above. The goal is to create an opinionated transaction

protocol that uses modern encryption standards and commonly used techniques, such as multi-factor authentication, to eventually provide a standard solution to replace bank cards altogether.

We will start with a simple implementation of a token-based transaction protocol. Then, by identifying flaws in the given procedures, we improve the protocol incrementally, eventually arriving at a state that we consider a good compromise between security and convenience for the users.

## 1.2 Credit Card Entropy

A bank card consists of 16 digits for the card, an expiry date (composed of 2 + 4 numbers), and a three-number CVV for authorization.

$$10^{16} * (10)^{2+4} * 10^3 = 4 * 10^{16} \implies \lceil log_2(1 * 10^{16}) \rceil = 54 bits$$

Fifty-four bits of entropy is already considered low, but from the calculation, it can be apparent that it's overly generous in many cases. For instance:

- It is reasonable to assume that we know the first six digits of the card number since it identifies the issuer bank. Also, the last four digits can legally be shown on purchase receipts.
- The sum of the digits on the card is further validated with Luhn's algorithm. This divides the options by a further 10-fold.
- Credit cards are typically issued for three to four years; this reduces the combinations to up to 48, but this can be further reduced if the attacker is aware of any past purchases with a date.
- The CVV can be calculated, but it's entirely random for our purposes since it requires multiple parameters that attackers might not have access to, such as the corresponding account number.

Considering these restrictions on the original calculations, the two possible entropies can be calculated by refining the formula and splitting it into cases when we know the final four numbers and another one where we do not.

Entropy knowing the last four digits:

$$\frac{10^6}{10} * (12 * 4) * 10^3 = 4.8 * 10^9 \implies \lceil log_2(4.8 * 10^9) \rceil = 33 bits$$

Entropy not knowing the last four digits:

$$\frac{10^{10}}{10} * (12 * 4) * 10^3 = 4.8 * 10^{13} \implies \lceil log_2(4.8 * 10^{13}) \rceil = 46 bits$$

In conclusion, a credit card's absolute entropy is between 33 and 46 bits. This is considerably lower than we consider acceptable nowadays, especially in the future. If we take AES encryption as an example, because of its wide use, nothing below a 128-bit key size is considered secure, and 256-bit is recommended. It is also important to note that with each additional bit, the complexity doubles.

## 1.3 Solution Proposal

Our proposal is a new, unified cardless banking standard where vendors are issued a token, allowing them to use each for pre-approved transactions. In this thesis, we are creating our standard solution to this problem and substantiating it with a demo system with guidelines for implementation.

The protocol combines post-quantum and classical elliptic curve-based digital signatures to verify the transaction details from the vendor and provider sides. The token also contains the public keys from both providers for verification to facilitate further negotiation about the state of the token, such as revocation or modification of recurring data.

In the authentication flow, these tokens would replace credit cards generally used for online services. This dramatically increases the entropy for a valid credit card address and reduces the chances of phishing attacks, as customers would never directly come in contact with such tokens, thus being unable to disclose them to attackers.

All transactions will involve the generation of secure tokens that include all details needed for a successful transfer and ensure that the customer knows the trade and all the related information. The validation of such transactions will be accomplished using a mobile authenticator application provided by the user's bank. This

application will use available methods, such as passwords, SMS tokens, or biometrics, to validate the user, then provide the user with the necessary information, such as transaction cost and recurrence, to validate it.

These guardrails create a three-way handshake between the vendor, the bank, and the customer, where all parties must agree to the transaction. The protocol will also involve manipulating tokens in the background, such as revocation from both sides. A vendor could revoke all their tokens in case of a breach, whereas a customer could revoke it once they want to cancel their subscription.

Implementing this protocol would be required on both the vendor's and the bank's sides, along with some changes to the vendor's website and mobile application. Some of this, however, could be mitigated since, in case the vendor already uses a 3rd party payment provider, only the given provider would have to make changes and provide some extra API endpoints to their vendors.

Altogether, this protocol would replace the entire payment verification flow, so significant changes to the infrastructure would be necessary. This, however, does not mean that it has to happen all at once. The recommended way to roll out these changes would be to classify it as a separate payment method and implement it. A different option next to "Pay with card" could use some form of "Pay with STP" or any familiar name.

# Chapter 2

# Cryptography

## 2.1 Overview

Preparing for post-quantum cryptography is paramount due to the potential threat that quantum computers pose to current cryptographic systems. Quantum computers can solve mathematical problems, such as factoring large numbers, exponentially faster than classical computers, rendering many currently used cryptographic systems vulnerable to attacks. As such, the development of post-quantum cryptography, which is resistant to attacks by quantum computers, has become a priority. Without proper preparation, sensitive information, such as financial transactions and personal data, could be compromised, leading to severe consequences for individuals, organizations, and governments.

Since the two primary cryptographic functions used for the digital signature in this project are hashing and asymmetric encryption, we need to determine a viable compromise between future proofing and usability. The algorithms have to be:

1. widely available as libraries for commonly used server programming languages,
2. reasonably fast to run, even for thousands of executions per second,
3. provides high-entropy signatures but still allows the token exchange over HTTP

## 2.2 Hashing

Hashing is a popular method for data encryption and secure data transmission over the internet. Engineers widely use it for data integrity checks, digital signatures, and password protection. Although researchers have discovered some cryptographic attacks on hashing algorithms over the years, most modern hash functions are still considered safe due to their resistance to collision attacks, preimage attacks, and others. Furthermore, new and more secure hashing algorithms continue to be developed and evaluated.

The security of a hash function relies on its ability to consistently produce a unique output for each input and the complexity of reversing the process. While hash functions are not infallible, they are still widely used and trusted by security experts because of their reliability and ease of use. This broad use has led to various algorithms being developed in the industry, utilizing multiple mathematical and computational challenges.

Experts generally consider hashing as a safer type of cryptography regarding quantum resistance. The primary method of attacking hashes with quantum functions is Grover's algorithm. It is a "black box" search algorithm to find the input of such a function with high probability. Even though this algorithm has existed since 1996, researchers have never demonstrated it to break modern hashing standards.

Let us define a hashing algorithm as:

$$H : \{0,1\}^* \to \{0,1\}^n$$

Where the number of input bits can be zero or arbitrarily large, but the output bits are fixed for each function type. This means that for each output, there are infinite inputs. The difficulty of finding two inputs that produce the same result is why hashing is considered secure. If a hashing algorithm has a demonstrated pair of distinct inputs that produce the same output, it is no longer considered secure.

In our case, collision resistance is the property that will be vital for the method to be considered. Collision resistance can be defined as selecting any $m \neq m'$ message pair that is insurmountably hard such that $H(m) = H(m')$. This rule makes it so that modifying any signed token will change the hash and thus invalidates the

signature, and forging a new similar token such that the hashes match is almost impossible.

### 2.2.1 Algorithms

The Secure Hashing Algorithms (used as SHA) family members are some of the most used hashing algorithms nowadays. These algorithms were published by the National Institute of Standards and Technology of the United States. The two commonly used generations of this family are the SHA-2 and SHA-3. SHA-2 improved on its predecessors in the resistance against collision and length extension attacks. The 256-bit version of the hashing algorithm is widely used in many libraries still to this day, but engineers increasingly prefer the 512 version because of the higher speeds on modern systems.

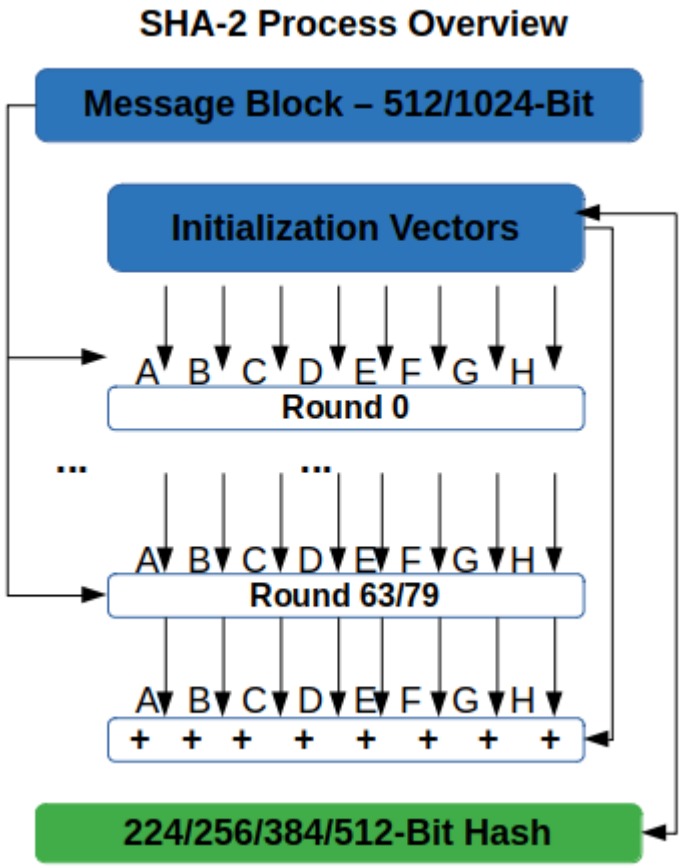


Figure 2.1: A simplified diagram that illustrates how the SHA-2 hashing algorithm works by Code Signing Store

The SHA-3 family, however, uses a new underlying problem to generate the

hashes. It is also comparable in speed and provides similar collision resistance. SHA-3 is a subset of a broader cryptographic family called Keccak. It relies on a principle called sponge construction. This dissimilarity to SHA-2 functions makes it an excellent pair for SHA-2 because of the underlying disjunct problem it's based on. If one were cryptographically broken, the other would likely still be secure.

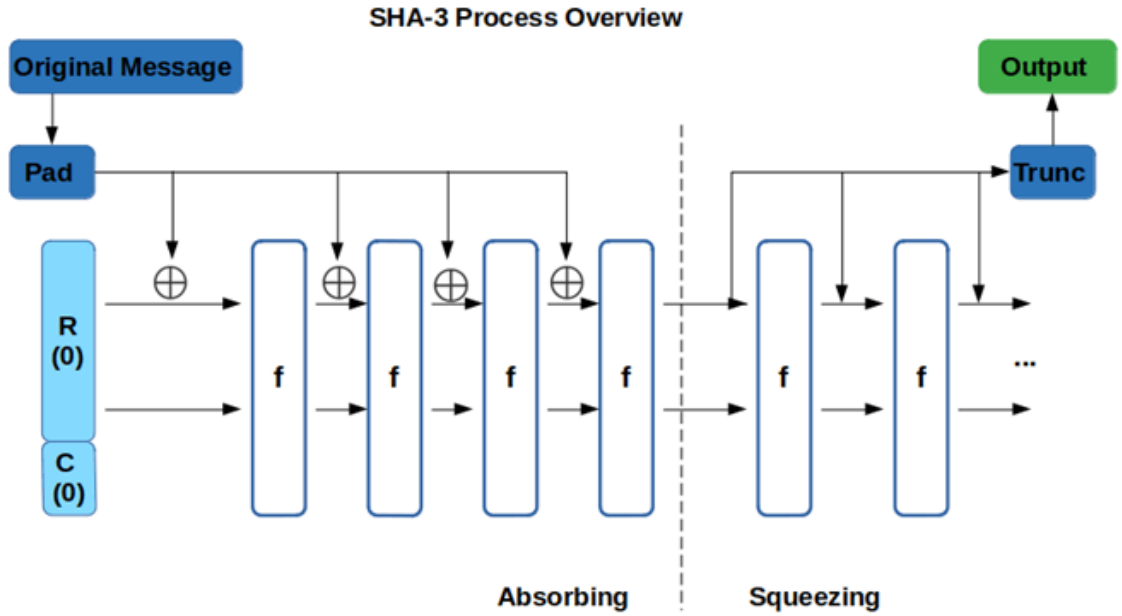


Figure 2.2: A simplified diagram that illustrates how the SHA-3 hashing algorithm works by Code Signing Store

The 512-bit variant of these functions was chosen because of their inherent computational advantage on modern systems. On bigger than a few bytes of data and a 64-bit processor, 512-bit hashing methods consistently outperform their 256-bit counterparts by up to 50%. To modify a token, an attacker must make a fraudulent token with the identical hashes of two distinct hashing operations. The likelihood of such is considered unattainable, even for a single hash; thus, increasing performance this way will not significantly increase the chance of a double collision, such that brute forcing becomes feasible.

### 2.2.2 Post-Quantum Hashing

Focusing on the previously selected hashing methods SHA-512 and SHA3-512, introducing quantum threats does not break the methods. Still, the most efficient

way to attack these systems is a preimage attack. SHA2 and SHA3, however, have shown excellent capability at being preimage resistant, even against current quantum algorithms. This, in practice, means that an attacker possessing significant quantum computational capacity might be faster at testing potential outcomes and iterating them. However, finding a collision will still take an insurmountable amount of time, especially one small enough to fit the token size.

This does not mean that such quantum algorithms will not be discovered. However, using the pair of secure hashing algorithms with disjunct underlying constructions will increase their chances of proving reliable long-term.

Hashing methods are generally widely considered safe against quantum computing, so we will not spend as much time validating this property as we will doing so with the public portion of the asymmetric key part of the digital signature.

### 2.2.3 Conclusion

The final choice for calculating the checksums for the digital signature is using a mix of SHA2-512 and SHA3-512 functions in a rotating fashion. The vendor and the provider both have to sign once. Thus they also have to hash once. This is true if we neglect to count the verification process, which is not consequential.

All tokens must have one signature signing the checksum using one and the other using the other hashing method. This can change for each token, so the initiator will choose one of the tokens, while the other will answer with its pair.

This will ensure that if one of the hashing methods, consisting of different underlying problems, is cracked, the tokens remain valid, and changing them fraudulently will remain impossible. This will allow some time to update the protocol with a new hashing method and allow time to invalidate the old tokens.

## 2.3 Digital Signatures

Digital signatures are cryptography primitives for verifying the integrity and authenticity of messages, documents, smart cards, and any other data that can be expressed as bits. A digital signature scheme contains three methods: Gen, Sign, and Vrfy:

- The Gen method is used for generating the key pair for calculating the signatures, $Gen(1^n) = (pk, sk)$
- The Sign method creates a signature of a given message using the private key, $Sign(m, sk) = \sigma$
- The Vrfy method verifies the correctness of the signature using the public key $Vrfy(m, \sigma, pk) = \{0, 1\}$

One useful feature of modern digital signature schemes is non-repudiation. It states that anyone with access to the message, the signature, and the signer's public key can verify that the signature was done with the signer's private key, known only by the signer. More specifically, the signer cannot later deny that they signed the message. A valid signature is therefore considered proof.

### 2.3.1 Popular Schemes

Digital signatures are based on complex one-way mathematical computations. These problems are relatively easy to execute in one direction but get exponentially challenging to reverse, given the bit length of the algorithm. The most commonly used problems are:

- the factorization problem (RSA),
- the discrete logarithm or DLOG problem (DSA, Schnorr, ElGamal),
- the elliptic curve DLOG problem (ECDSA, EdDSA).

Regrettably, these algorithms are susceptible to attacks by large-scale quantum computers. Using Shor's algorithm, most schemes can be broken, meaning signatures could be forged knowing only the public key. Public key cryptographic functions are the primary schemes easily broken using quantum algorithms. Because of this, post-quantum variants were considered for this thesis to make the banking protocol future-proof.

### 2.3.2 Post-Quantum Signatures

Quantum computers did not yet reach the levels necessary to break modern pre-quantum algorithms. Still, sensitive data created now could be relevant years later when quantum computers will be a considerable threat. For this reason, the National Institute of Standards and Technology (NIST) started the Post-Quantum Cryptography Standardization [2] to establish alternatives to the beforementioned algorithms. The standardization is still in progress, but in 2022 some algorithms were selected as the first standardized PQ algorithms. This announcement contained three digital signature schemes:

- CRYSTALS-Dilithium https://pq-crystals.org/dilithium/index.shtml
- FALCON https://falcon-sign.info
- SPHINCS+ https://sphincs.org

Dilithium and Falcon are based on lattices; meanwhile, Sphincs is based on a hash function, both unaffected by quantum computers. All three above have their original versions implemented in C but also have WebAssembly wrappers for JavaScript. Furthermore, each is under permissive open-source licenses like the Apache 2.0 and the MIT licenses.

### 2.3.3 Comparison

The table below contains different versions of the three standardized schemes with relevant data about their key and signature sizes, security levels, and performance.

| Method | Security level | Public key size (bytes) | Private key size (bytes) | Signature size (bytes) | Keygen (CPU cycles) | Sign (CPU cycles) | Verify (CPU cycles) |
|---|---|---|---|---|---|---|---|
| Crystals Dilithium 2 | 1 128-bit | 1 312 | 2 528 | 2 420 | 116 511 | 342 726 | 112 506 |

| Method | Security level | Public key size (bytes) | Private key size (bytes) | Signature size (bytes) | Keygen (CPU cycles) | Sign (CPU cycles) | Verify (CPU cycles) |
|---|---|---|---|---|---|---|---|
| Crystals Dilithium 3 | 3 192-bit | 1 952 | 4 000 | 3 293 | 191 331 | 534 254 | 180 350 |
| Crystals Dilithium 5 | 5 256-bit | 2 592 | 4 864 | 4 595 | 307 765 | 610 807 | 417 971 |
| Falcon 512 | 1 128-bit | 897 | 1 281 | 690 | 24 656 358 | 1 085 984 | 183 949 |
| Falcon 1024 | 5 256-bit | 1 793 | 2 305 | 1 330 | 74 741 342 | 2 204 927 | 359 553 |
| Sphincs SHA256-128f Simple | 1 128-bit | 32 | 64 | 17 088 | 3 088 404 | 72 191 077 | 8 962 488 |
| Sphincs SHA256-192f Simple | 3 192-bit | 48 | 96 | 35 664 | 3 920 103 | 119 085 653 | 12 269 960 |
| Sphincs SHA256-256f Simple | 5 256-bit | 64 | 128 | 49 856 | 10 771 651 | 220 637 782 | 12 168 947 |

Sources

- Digital Signatures for the Future: Dilithium, FALCON, and SPHINCS+ - https://medium.com/asecuritysite-when-bob-met-alice/digital-signatures-for-the-future-dili
- PQC Digital Signature Speed Tests - https://asecuritysite.com/pqc/pqc_sig

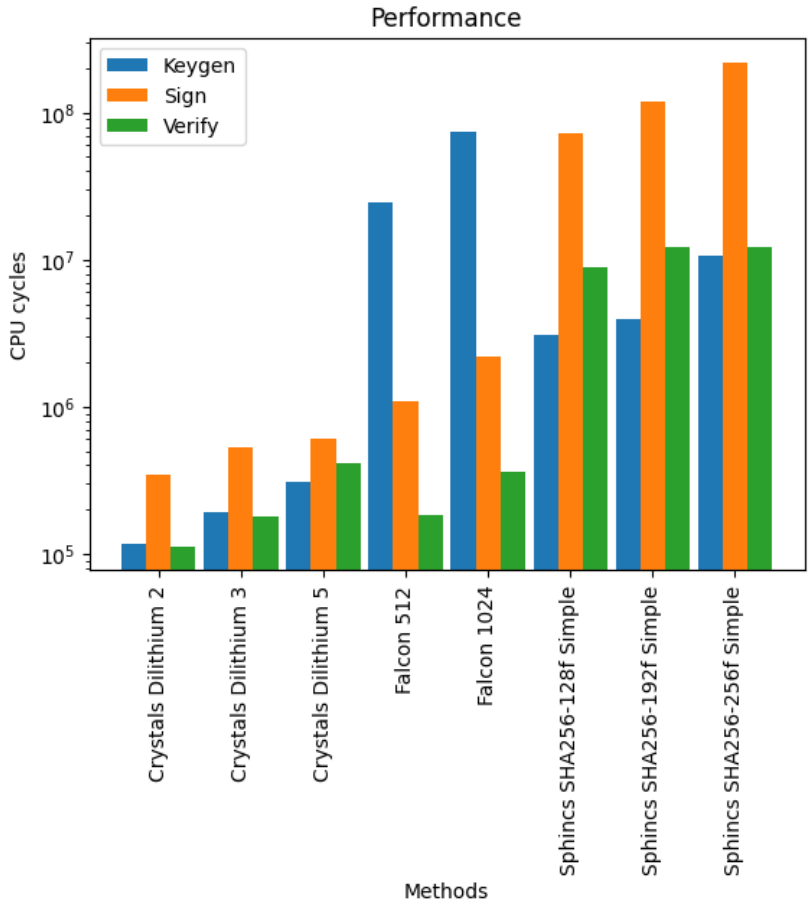


Figure 2.3: Performance comparison chart between the three PQ signatures

As seen above, Dilithium is the fastest algorithm in all three aspects, with moderate key and signature sizes. At the same time, Falcon performed more poorly, especially in key generation, but has noticeably smaller key and signature sizes. Finally, Sphincs operates with tiny key sizes, even smaller than a similarly secure RSA. Likewise, both its performance and signature size are inadequate for our use.

We also need to consider that NIST recommends using PQ algorithms in hybrid mode, combining them with a well-established pre-quantum algorithm. This is necessary because the algorithms mentioned above are not fully standardized yet, so they are most likely quantum-safe, but the certainty of this may not meet industry standards. For this reason, already-proved robust algorithms should be used for redundancy reasons [3].

To make our protocol as future-proof as possible, we decided to use the highest versions of the digital signature schemes (NIST security level 5, AES-256).

The following essential aspect for us is the key and signature sizes. PQ signature

| Method | Token size (KB) |
|---|---|
| Crystals Dilithium 5 | 15 |
| Falcon 1024 | 7 |
| Sphincs SHA256-256f Simple | 101 |

Table 2.2: Comparing the token size with the various signatures

schemes performed here much more poorly than the pre-quantum ones; meanwhile, our token contains two public keys and two signatures. The token's size with different algorithms would be approximately as seen in table 2.2.

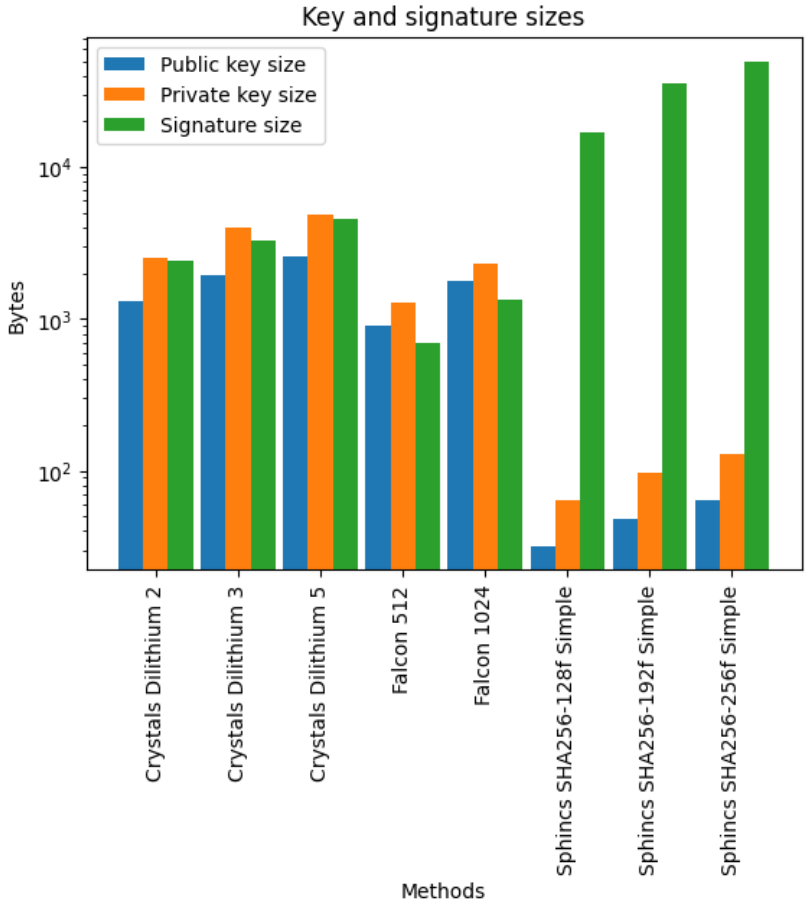


Figure 2.4: Signature size comparison chart between the three PQ signatures

Token size equals the transaction content (1 KB), two public keys, and two signatures. With this comparison alone, we ruled out Sphincs as a candidate because of the relatively big size of the signature compared to the other options.

### 2.3.4 Falcon Signature

The name Falcon stands for Fast Fourier lattice-based compact signatures over NTRU. As its name states, the algorithm is a lattice-based digital signature scheme. This class of problems is believed to be secure under large-scale quantum computers, unlike factorization and discrete logarithm-based algorithms.

The idea behind lattice-based problems is that an N-dimensional lattice can be generated by choosing any N linearly independent vectors (points in the lattice) as the generator. Given arbitrary generator vectors and a target point not on the lattice, it is usually hard to find which linear combination of the vectors gives back the closest lattice point to the target point. At the same time, if the length of the generator vectors is minimal, then finding the correct linear combination is more feasible. Naturally, the problems the algorithms above are based on can vary from this. This is just a high-level overview of the problem class.

Falcon combines the GPV framework[4] with NTRU lattices and Fast Fourier Sampling[5]. The design rationale of Falcon revolves around optimizing for the size of the public key and the signature, the scheme parameters commonly transported over networks. This property is optimal for keeping the token size of STP minimal.

At its core, Falcon instantiates the GPV framework. The framework describes a general lattice-based digital signature scheme without specifying the exact lattices and samplers used. In GPV, the different parts of the scheme are defined as the following:

- The public key is a full-rank matrix $A \in Z_q^{n \times m}$ $(m > n)$
- The private key is a matrix $B \in Z_q^{m \times m}$, which generates a lattice $\Lambda_q^{\perp}$ perpendicular to $\Lambda$ modulo $q$. Also $B \times A^T = 0$
- For message $m$ the signature is $s$ in such a way that $sA^T = H(m)$, where $H : 0, 1^* \to Z_q^n$ is a hash function
- Knowing $s$, $A$ and $m$ the verification is trivial

The GPV framework requires a class of lattices and a trapdoor sampler to be instantiated. In the case of Falcon, NTRU lattices are used since they can reduce the public key's size by a factor of $O(n)$ while speeding up some essential parts

of the algorithm by a factor of $O(n/log\ n)$. NTRU lattices are described by four polynomials $f, g, F, G \in Z[x]/(\phi)$, where $\phi = x^n + 1$ and $n = 2^k$. These polynomials satisfy the equation

$$fG - gF = q \mod \phi$$

provided, that $f$ is invertuble modulo $q$, we can define $h := g \cdot f^{-1} \mod q$. This way the original polynomials $\begin{bmatrix} f & g \\ F & G \end{bmatrix}$ and $\begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix}$ generate the same lattice. Knowing this GPV can be instantiated in such a way:

$$A = \begin{bmatrix} 1 & h \end{bmatrix}$$

$$B = \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$$

The public key $A$ is equivalent to knowing polynomial $h$, so the public key is relatively short.

Creating a short signature is more delicate. To achieve this first preimage, $c_0 \in Z_q^m$ is calculated in a way that satisfies $c_0 A^T = H(m)$. After this, $v \in \Lambda_q^\perp$ can be calculated using the matrix $B$. Finally let the signature be $s = c_0 - v$, so $sA^T = c_0 A^T - vA^T = c - 0 = H(m)$. Knowing $B$, we can minimize $c_0 - v$ to make the signature as short as possible.

To find $v \in \Lambda_q^\perp$ without leaking any information about $B$, a trapdoor sampler is required, randomly sampling the shifted lattice $c_0 + \Lambda_q^\perp$. The creators of Falcon chose the relatively new Fast Fourier Sampler, which combines the quality of Klein[6] with the efficiency of Peikert's algorithm[7]. At the same time, the FFS's randomness breaks one of GPV's requirements, which states that two signatures of the same hash can never be made public simultaneously. To combat this, a random seed $r \in \{0,1\}^k$ is appended before the message in hashing, and it is also appended to the signature to make the verification possible. With this built-in mechanism, we can avoid replay attacks and any issues where the provider may be forced to generate two transactions with identical transaction IDs.

The more detailed specification of Falcon can be found in the Falcon Specification article[8].

| $n$ | Classical | Quantum |
|---|---|---|
| 512 | 120 | 108 |
| 1024 | 277 | 252 |

Table 2.3: The entropy of Falcon signatures

### 2.3.5 The Entropy of Falcon

Our protocol aims to be used in place of credit card data, providing more rigorous security and much higher entropy. To calculate the entropy of the tokens, we propose in chapter 3, first the exact entropy of the digital signatures is required since they are a crucial part of the token.

Falcon has two sets of parameters $n = 512$ and $n = 1024$, corresponding to NIST level I and V security. Security level I is equivalent to an AES-128, basically 128 bits of entropy. At the same time, level V is for AES-256 or 256 bits of entropy. The Falcon Specification[8] presents us with more detailed results in section 2.5. Based on this, the exact signatures entropies are in table 2.3. Naturally, forging signatures takes different computations on classical and quantum computers, equivalent to having different entropy levels in these two scenarios.

As mentioned before, post-quantum signatures are recommended to be combined with a pre-quantum signature in hybrid mode. Ed25519 is a well-established classical digital signature scheme since it is widely used for its minor key and signature sizes. Ed25519 is at NIST security level I, meaning 128 bits of entropy. This, combined with Falcon1024, results in a $277 + 128 = 405$ bits of entropy per signature on classical machines. Ed25519, based on the elliptic curve discrete logarithm problem, is broken under large-scale quantum computers, so its entropy in this scenario is negligible. We conclude that a Falcon1024-Ed25519 combination still has an entropy of 252 bits in the quantum system.

### 2.3.6 Conclusion

Finally, we decided to use Falcon 1024 for our protocol because of the relatively small signature size, good performance, and sufficient quantum resistance. At the same time, we acknowledge Falcon's shortcomings compared to Dilithium. However,

key generation is not commonly executed, so its speed is less relevant than other factors. Also, Falcon's signing process is 3.6 times slower, but the verification is 1.16 times faster. We accept these performance compromises to keep the token's size as small as possible.

It is vital to note that all of these tokens will have to be exchanged over the network multiple times for the generation, then once more for use. The TCP protocol allows for a maximum of 64 Kb of data transferred in a single packet, which is considerably smaller because of the data added by the higher layers. All extra sizes added to the token will require more packets to be transferred.

## 2.4 Symmetric Encryption

### 2.4.1 Overview

Symmetric encryption schemes are used if both entities can access the same secret key. This usually happens if the encrypting and decrypting parties are the same or if two entities pre-negotiate a symmetric key.

Pre-negotiation is advantageous because it is considerably faster than public key cryptography and works with practically unlimited data. We use symmetric encryption in the protocol where both parties can access the same data because of the pre-negotiated token.

Asymmetric encryption schemes are only used in our case for digital signatures because public key encryption is susceptible to quantum algorithms. In contrast, symmetric key cryptography is affected, but not nearly to the same extent. These algorithms use a certain bit-length to determine how secure the encryption is. Quantum algorithms can dynamically reduce the complexity by half of the bit length. This means AES 256 encryption becomes the same complexity as AES 128 encryption, given a quantum computer with the same processing power as a regular computer.

This is not a small decrease since every extra bit of entropy doubles the cracking complexity. Despite this being a major hindrance, the solution is quite simple. It is doubling the bit length.

Since the 128-bit Advanced Encryption Scheme is considered secure in 2023, AES-256 will give the same security by introducing adversaries possessing quantum computers. Because of this, we opted to keep using the tried and tested AES standard for symmetrically encrypting data in transit.

With the introduction of HTTPS to the protocol, packets are already encrypted between the endpoints. Because of this, we are not using this to add a layer of encryption to all the JSON objects; instead, we make sure that certain sensitive pieces of information can only be decrypted by the party for whom it is intended. This is important because the data may travel through TLS terminating load balancers, or the DNS cache may be poisoned, sending the request to the invalid server.

### 2.4.2 Conclusion

We are using AES-256 to do symmetric encryption on data that is considered sensitive when we haven't yet verified the identity of the receiving server. This helps facilitate one-round trip remediation and revision steps.

## 2.5 Transit Encryption

### 2.5.1 Overview

The protocol we opted to use is HTTPS, which is discussed further in chapter 3. HTTPS is the backbone of the modern internet, so there is a high interest in preparing it for quantum algorithms.

HTTPS uses TLS as the standard for encrypting the communication between the server and the client without a third party to manage the asymmetric key exchange. This consecutively uses public key cryptography, the primary cryptographic method susceptible to quantum attacks. The currently used RSA-based key exchange is going to be insufficient later on.

Because of this, we considered creating a custom protocol to exchange messages over TCP, from the initial key negotiation to the transfer of symmetrically encrypted packets containing the tokens and other details of the transaction. While designing this protocol, we realized that we needed more and more features, getting ever

closer to the already existing TLS scheme but with quantum-resistant algorithms. Considering this and the fact that it is already implemented on all web servers we consider essential for this project, we opted to consider future improvements to the protocol.

### 2.5.2 Conclusion

A solution already exists to improve TLS versions 1.2 and 1.3, according to a Microsoft article on Post-Quantum TLS [9] through the Open Quantum Safe (OQS) project https://openquantumsafe.org/.

The project is funded by major players in the server space, including Microsoft, Amazon, IBM, and Cisco. Considering that HTTPS is the most important protocol for the internet, especially for websites we aim to target with this protocol, we are confident that HTTPS will receive significant updates and be prepared for quantum computers.

# Chapter 3

# Communication Protocol

The STP (Signed Transaction Protocol) protocol consists of the tokens that are generated, signed, and used for money transfers, also the underlying communication protocol that facilitates all exchange steps.

These must be validated on each step and protected against man-in-the-middle attacks. It must be sufficiently challenging to impersonate a bank or vendor and use commonly available technologies.

## 3.1 Communication Architecture

### 3.1.1 WebSockets

In the first version, we opted to use WebSockets. A two-way communication protocol over HTTP(S) because of its wide adoption, ease of use, native compatibility with web browsers, and the convenient transfer of consecutive messages. This, however, was deemed challenging because of several factors.

Having to keep a channel open for an extended period while we allow the user to see the transaction details in their application and decide to approve or deny it introduces several problems. First, web systems have limited connections they can open concurrently. This is not an inherent limitation of the WebSocket standard but rather the underlying hardware and operating system. Linux systems limit the number of files that can be in an opened state at once. Each alive connection opens up another file for buffering, which exhausts system resources. This issue could mean

that bad actors could reasonably open thousands of connections simultaneously and cause an outage with a denial of service attack on the vendor or the bank.

Another potential issue is consistency. If a connection were to fail mid-transaction, it would mean that either it would have to be restarted from the beginning of the protocol and would have to implement some recovery state, where it waits for the connection to continue and make sure that both the vendor and the bank's systems are in a consistent state. This could introduce significant complexity to the protocol.

Because of this, we looked for a way to minimize the number of messages exchanged and the duration in which the sockets have to be open.

### 3.1.2 HTTP/HTTPS

In the second iteration, considering the learnings from WebSocket, we opted to create a system with multiple round trips. In the HTTP protocol, a single request is followed by a single response. Because of this limitation, reducing the number of round-trips is crucial for reliability. We set a maximum of two round trips as our goal. The first is the negotiation phase, where the two systems exchange information. The second is the confirmation phase, where the bank allows or denies the token request based on the user's input and other factors, such as sufficient funds. This is the phase where the bank signs the token, validating it.

HTTPS is a great alternate protocol to WebSockets because it has many of the same advantages as WebSockets have, but with a few limitations:

- HTTPS is widely used in web communication, so it's well-tested and secure,
- nearly all languages and web frameworks provide ways to communicate using HTTPS over TCP, it is already available for all online stores and websites,
- communicates in round trips, with one party sending a request, then the other sending a response. Communication is stateless between the trips,
- it supports transferring any data type.

The Quantum-resistance of the HTTPS protocol is discussed in the Cryptohtaphy 2.5 section.

## 3.2 Token Exchange

### 3.2.1 Token Negotiation Protocol

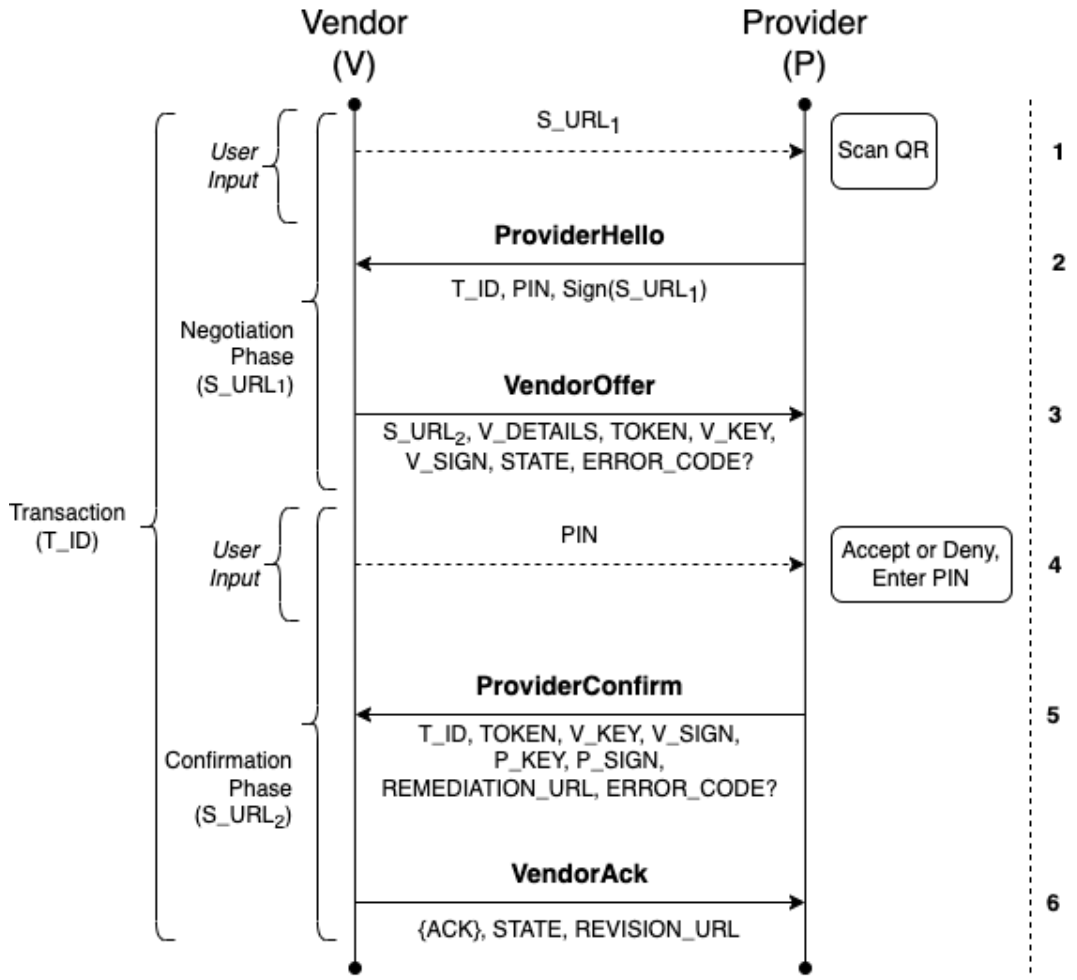


Figure 3.1: A flowchart of the sample HTTPS communication between the vendor and the provider

Please refer to figure 3.1. It shows the transaction protocol from the perspective of the vendor and the provider. The user input is collected twice, first to initiate the transaction, then to authorize it on the provider's side after any applicable authentication.

Detailing each step:

1. When initiating the checkout process, the vendor generates a signed URL $S\_URL_1$. This is displayed on the website in the form of a QR code. The user opens their banking application and scans the QR code.

2. During the ProviderHello, the provider connects to the signed endpoint using HTTPS protocol, sends a unique transaction identifier T_ID, random pin code, and signs the S_URL$_1$ with its private key. This key will be used to validate the provider's identity to the vendor. The vendor has to verify the signature with the attached public key. TODO: pubkey not attached

3. The vendor replies to the HTTPS request with a VendorOffer containing a token with the transaction details, signed by its private key, and some information on the vendor to display to the user. For example, the vendor may also reply with an error because of an unsupported protocol version. This concludes the negotiation phase of the transaction.

4. In the confirmation phase, the user is presented with the vendor information and the transaction details in the application, where they may choose to approve or deny the transaction. If they decide to approve, they must enter the PIN displayed on the website. This will prevent any attacker from opening a session and expecting the user to authorize a fraudulent transaction, similar to MFA fatigue.

5. After receiving the confirmation from the user, in the ProviderConfirm step, the bank signs the token in the user's name with its key. Since the transaction has not introduced any customer ID, the bank will associate the transaction ID with the user's account. This way, the token remains anonymous for the vendor. TODO: customer data stored in token instead

6. After receiving the signed token, the vendor saves it and associates it with the given transaction. It sends a VendorAck message to conclude the confirmation phase if all the signatures and data are successfully validated. It also sends a revision URL, where a notification can be sent if the token is revoked on the bank's side.

The transaction consists of two round trips initiated by the provider. If the connection fails during the first round trip, the entire transaction is reset and has to be restarted. Since there are no active connections between the phases, it is a safe zone for requests to fail. Suppose a request does not reach the destination from the

vendor's side. It may retry later. The same applies if the second confirmation fails with anything other than an error code from one of the parties. If it fails and an error code is received, the entire transaction is considered a failure, and the signatures are invalidated. It may be retried if the request or the response timed out and did not produce an error.

Please refer to the attached repository for more details on the possible error messages and responses.

### 3.2.2 Remediation Protocol

Token remediation is the process of the vendor contacting the provider and asking for modifications to the token. Since all the tokens are validated by a digital signature, a new token signature is generated for each remediation.

The remediation session starts with the vendor contacting the specified REMEDIATION_URL of the provider. The vendor sends this URL during token negotiation. The vendor prepares all required data for the transaction.

In previous versions of the remediation protocol, we have used two different round trips. The first validated the initiating party, while the other described the intent and updated the token. After switching to HTTP from WebSockets, the round trips would have been separated into two HTTP request-response pairs. Both pairs had to be authenticated, so leaving them separate provided no additional advantage.

The two round trips were eventually merged, and the initiator encrypted the token to protect user data. This also speeds up the process and makes mass remediation simpler.

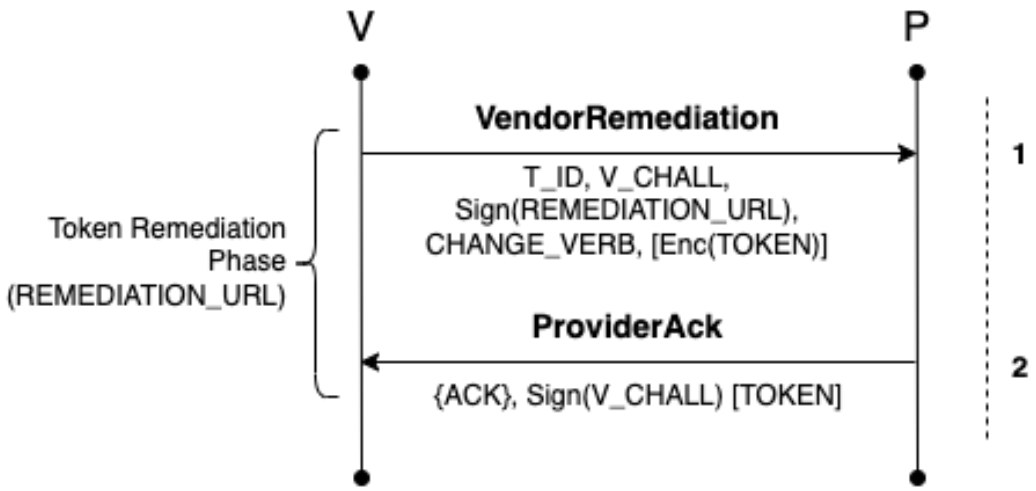


Figure 3.2: A flowchart of the remediation protocol initiated by the vendor

(1) To start the remediation process, the vendor will contact the provider on the pre-signed URL acquired during token negotiation. In this step, the vendor will sign the remediation URL using the private key pair of the public key in the token. This will initially verify that the token's creator started the remediation process. The vendor also includes the instructions about the change they intend to make alongside the token data encrypted by the hash of the previous token version. This is sufficiently random since the tokens contain the signatures, which effectively count as a nonce. The vendor also includes a challenge that the provider will have to sign with their key in the token for the vendor to accept the response as legitimate.

The formula for calculating the sent token ($T_e$) is

$$T_e = Enc_{H(Sign_p(T))}(T \times T') \tag{3.1}$$

Where $T$ is the currently valid token, and $T'$ is the modified and signed token by the provider. The two tokens are concatenated JSON objects as strings.

Conversely, decryption is done by the pre-saved hash of the token by the provider as

$$T \times T' = Dec_{H(Sign_p(T))}(T_e) \tag{3.2}$$

(2) Upon receiving the remediation request, the provider will verify the signature, decrypt the tokens and determine if the proposed modifications are allowed. This logic is up to be determined by the bank.

Three distinct responses are valid by the provider.

- Accepted - The provider authorized modifications immediately, and the newly signed token was returned. The token is only valid after the provider returns it.
- Denied - The remediation request was denied for any of the specified reasons. For example, it can be because of an unauthorized change or an invalid signature. The previous token is not revoked if the vendor's identity was improperly established because of an invalid signature.
- Pending - Some modifications may require user interaction. For example, if the vendor seeks to increase the monthly subscription cost, the provider may prompt the user to re-authorize the subscription. This is not supported on one-time tokens.

For more information on the precise responses, please refer to the protocol design documentation in the attachment.

With these safety measures in mind, the protocol does the validation and the negotiation in a single step. With these signatures, the data is protected against man-in-the-middle attacks. For example, the single-round negotiation makes it fast to remediate many tokens in case of a breach. After remediation, the previous token is invalidated by the bank. Since the signatures will change upon any modification to the data in the token, the resulting object is effectively a new token.

### 3.2.3 Revision Protocol

Token revision is the process of the provider contacting the provider and issuing a modification to the token. The provider revision can be classified into two main categories.

- Revocation - The provider issues a revocation order to the vendor
- Pending resolving - The provider sends the newly signed token to the vendor or informs it about the request being denied.

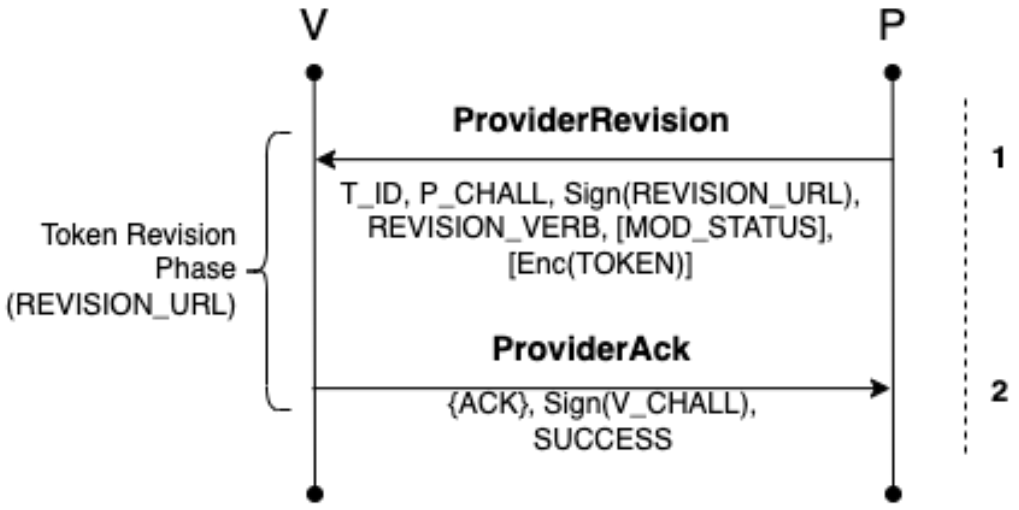


Figure 3.3: A flowchart of the revision protocol initiated by the provider

In case of a revocation order, the provider has already revoked the token, and it can no longer be used for transactions. The vendor does not have to accept a revocation; it is only informed about the state change. This can be because a user has decided to cancel a subscription in their banking app, so the bank has invalidated the corresponding token. It's important to note that the bank does not have to inform the vendor that the token may no longer be used. This prevents vendors from negotiating with an invalid prevision URL or denying the revocation altogether.

The provider will answer with a reason if a pending request is denied. At this point, the authenticity of the proposal was established; it cannot be rejected because of an invalid signature. Instead, the user denied it manually, or the bank's fraud prevention system must have flagged the request. In both cases, the initiator token becomes invalid, and the token under remediation never gets signed by the provider.

### 3.2.4 Summary

We have defined three token negotiation and revision/remediation sub-protocols to standardize cryptographically safe transaction authorization. These protocols are implemented on each vendor checkout system that supports STP transactions. Because of the asymmetric key-based signature system, the front-facing web server may be disjunct from the transaction service.

This allows for separation to a different server, service, or serverless function. This follows modern standards in cloud computing where such tools may be implemented alongside supplementary Cloud HSM modules for maximal transactional safety.

This protocol allows transactions to be authorized even without the bank having all the information, similar to how JSON Web Tokens work.

Conditional symmetric encryption in one-round negotiations provides end-to-end safety even when the TLS is terminated early or the messages are directed to an unauthorized party.

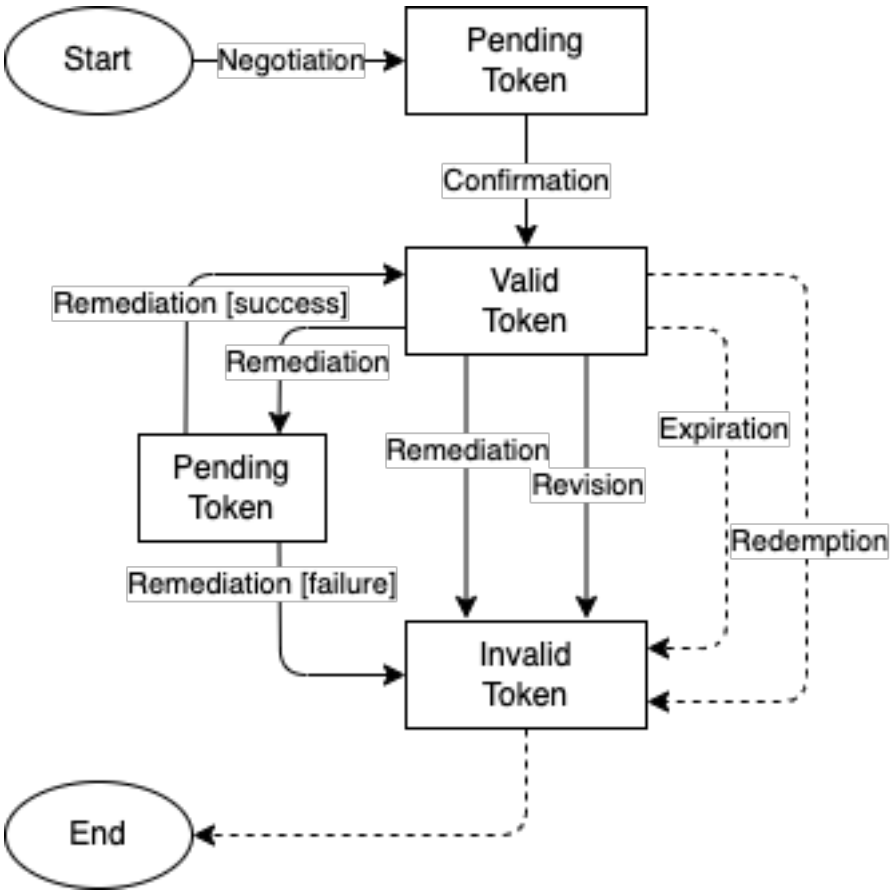


Figure 3.4: Lifecycle of an STP token

The token will always move between valid states; eventually, it will be invalidated by some action or timed out. In this chapter, we have defined all the methods that navigate the token between these states with the help of both the vendor and the provider's participation.

## 3.3 Example

This section covers an example of communication for each sub-protocol. Since all message exchanges use HTTPS requests with JSON payloads, all data will consist of a request and the response. Request objects are denoted as

```
--> [initiator]
```

and

```
<-- [receiver]
```

to show if the given package is the request or the response, as well as the party who initiated it.

The examples only show a single negotiation with sample data, which might not always accurately simplify the object. For instance, ... is often used if the data in the given field is too large and unnecessary to display. Please refer to the attached demo project for complete example code samples.

Please refer to the attached repository for more details on the possible error messages and responses.

### 3.3.1 Token Negotiation

(1) The user presses the pay button on the website, and the pre-signed URL is generated and presented as a QR code.

```
stp://vendor.com/api/stp/request/CF45D22C-28B8-41E7-AC78-
   B6C81580F575
```

(2) The user scans the code, and the bank requests the transaction details from the vendor.

```
--> provider
{
  "version": "v1"
  "bank_name": "mybank"
  "bic": "ABCDHUBP001"
  "random": "gG7HD+JRhc2FszVdPfGR/LF6KTgXxjE1bHXdEU8F"
```

```
  "transaction_id": "013CCF92-1313-434B-A838-6BE3D9645DD1
    "
  "customer": "juastf89234r2bewiohfwf6qw"
  "url_signature": "SWvwHAstKjI7tRGdJqaUT5eA5mljMP2HAzKqo
    8fw..."
  "verification_pin": 1234
}
```

The vendor receives the request from the bank and pairs it to the open transaction using the URL signature. It creates the vendor offer, including the details of the purchase and some information about the vendor.

```
<-- vendor
{
  "success": true,
  "confirmation_id": "AD3621AD-2D2D-4BF7-A3EE-A71F054B684
    7",
  "response_url": "stp://vendor.com/api/stp/response/AD36
    21AD-2D2D-4BF7-A3EE-A71F054B6847",
  "vendor": {
    "name": "My Vendor",
    "logo_url": "https://myvendor.com/images/logo.png",
    "address": "Washington, Imaginary st. 184"
  },
  "transaction": {
    "amount": 14.99,
    "currency_code": "USD",
    "recurrence": "monthly"
  },
  "token": ...
}
```

(3) The pin code is displayed on the vendor's site. That pin will verify the transaction is the same as the one on the screen. This is used to combat attacks

similar to "MFA fatigue", where the user might assume a request looks right and authorize it. This is especially likely if the user makes a lot of smaller transactions and has to authenticate often. Since this protocol is not explicitly designed for end-user use only, scenarios where representatives at companies have to authorize these transactions, are likely.

The format of entering the pin code or the precise length is not defined explicitly. Some vendors may prefer providing a pin pad with a four-digit code, while others may opt to use a two-digit pin with a couple of combinations displayed on the screen. These methods should give users sufficient uncertainty that they would not press a button randomly and potentially allow for a fraudulent transaction.

(4) If the user approves the transaction and enters the pin, the token will be signed by the provider and returned in a successful HTTP request.

```
--> provider
{
    "allowed": true,
    "token": ...,
    "remediation_url": "stp://provider.com/api/stp/change
       /26A50373-6290-4EDD-8F8D-ED22C5DE9299"
}
```

As the answer to the signed token, the vendor verifies that they and the signature are correct and return a successful response. The response also contains a revision URL, to which the vendor will have to initiate all revision requests regarding this token.

```
<-- vendor
{
    "success": true,
    "revision_url": "stp://vendor.com/api/stp/revision/70
       60129E-4FBC-48CD-AA2A-9FB2E718777E"
}
```

A shortened example of a filled and validated signed token:

```
{
```

```
    "metadata": {
        "version": 1,
        "alg": "sha512,sha3512",
        "enc": "sha512,aes256",
        "sig": "falcon1024,ed25519"
    },
    "transaction": {
        "id": "013CCF92-1313-434B-A838-6BE3D9645DD1",
        "expiry": "2023-04-12T20:03:12.477Z",
        "created_at": "2023-04-12T20:03:12.477Z",
        "provider": "REVOLT21",
        "amount": 12.66,
        "customer": "juastf89234r2bewiohfwf6qw",
        "currency": "USD",
        "recurring": {
            "period": "monthly",
            "next": "2023-04-12T20:03:12.477Z",
            "index": 0
        }
    },
    "signatures": {
        "vendor": "oKbfJohV4Nq5rCeWM74uKnFyniAV2Ae9Sbr3
           Fwdr2H6OEuVzYpJjGYFFOZ+5...",
        "vendor_key": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8
           AMIIBCgKCAQEA4iAzt4C4T16wclcCbo9pXZn...",.
        "signed_at": "2023-04-12T20:03:12.477Z",
        "provider": "L5oaGF/zyMxmY4r6bZfU/ow5TPoMzvL5
           xqUjc7//nDiKCzlmdXmE...",
        "provider_key": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8
           AMIIBCgKCAQEAuLJInsXWreKLvBbQSDGCk..."
    }
}
```

Aside from the transaction section, each token contains a metadata object that details the protocols and encryption schemes used for each token creation step. While these methods are mostly fixed because the protocol is highly opinionated, it still serves a purpose in some cases.

Suppose a library does not have an entire implementation of the STP protocol covered, having methods for all versions. In that case, they may still perform more straightforward tasks, such as verifying the signature on the token by reading the appropriate values, the calculating them on the data.

Some parts of the algorithm, however, may vary. This token's metadata.alg section details using sha-512 and sha3-512 in this order. This means that the vendor has used sha-512, while the provider has used sha3-512 to calculate the checksum for their signatures.

The signature section contains the public keys of the vendor and the provider. These serve as ways to fingerprint the vendors and providers and to perform validity checks on the token. The signed_at field is set by the provider upon signing the token, as this step is the last one in the token signing flow.

### 3.3.2 Token Remediation

In cases when the vendor seeks to modify, refresh, or revoke a token later, this can be achieved using a token remediation request. The vendor can send a request to the remediation URL agreed upon turning the token negotiation phase.

An example of a token modification request

```
--> vendor
{
    "transaction_id": "013CCF92-1313-434B-A838-6BE3D9645
       DD1",
    "challenge": "vKvqQA8BvPmXT/QogPve6briG6dvivo3EXx3p1
       mj",
    "url_signature": "ldRSWRYYWFwdkpHQ3gwVGZYZnhyWVNPWE1
       YRlZ0eG9S...",
    "remediation_verb": "MODIFY",
    "original_token": ...,
```

```
    "modified_token": ...
}
```

The provider may respond with success, failure, or request pending. This example shows a successful response. The vendor will validate the signed response to the challenge and the resulting signed token.

```
<-- provider
{
    "success": true,
    "response": "kV2tD6iuApOm8uspBat+KsXG+fP4Eb+
       HHkAYOeqmAUeB...",
    "token": ...
}
```

In cases where the modification returns as pending, a token field is omitted, and instead, the modification_status field is included and set to PENDING. In such cases, the provider will later send a token revision request to confirm or deny the request.

Please refer to the attached repository for more details on the possible error messages and responses.

### 3.3.3 Token Revision

In cases where the tokens are revoked on the provider's side, those may send a token revision request.

```
--> provider
{
    "transaction_id": "013CCF92-1313-434B-A838-6BE3D9645
       DD1",
    "challenge": "1dFMqKhh3TGUwkBfW6FmZhE+fURLNcaec0LArkG
       9",
    "url_signature": "ldRSWRYYWFwdkpHQ3gwVGZYZnhyWVNPWE1
       YRlZ0eG9S...",
    "revision_verb": "REVOKE"
```

```
}
```

Since the vendor has no authority to overrule a revocation in the revision request, the vendor will always respond with success unless the signatures are invalid. Even if the vendor replies with a failure, the token is still revoked.

```
<-- vendor
{
    "success": true,
    "response": "kV2tD6iuApOm8uspBat+KsXG+fP4Eb+
       HHkAYOeqmAUeB..."
}
```

The provider may also use the token revision to follow up on a modification-type token remediation request.

```
--> provider
{
    "transaction_id": "013CCF92-1313-434B-A838-6BE3D9645
       DD1",
    "challenge": "1dFMqKhh3TGUwkBfW6FmZhE+fURLNcaec0LArkG
       9",
    "url_signature": "ldRSWRYYWFwdkpHQ3gwVGZYZnhyWVNPWE1
       YRlZ0eG9S...",
    "revision_verb": "FINISH_MODIFICATION",
    "modification_status": "ACCEPTED",
    "token": ...
}
```

This is an example of an accepted modification request. Note that the token field is not an object but rather a base64 string because the token is encrypted with the hash of the token ID.

Please refer to the attached repository for more details on the possible error messages and responses.

### 3.3.4 Summary

This section demonstrated one or more examples for each type of transaction supported by the protocol. It, however, is not a comprehensive list of all possible messages that can be sent and received.

For engineers implementing this solution, we always recommend employing strict error-checking measures and ensuring that all fields contain only the information allowed.

Please refer to the attached code repository for live examples or a more in-depth look at the communication.

# Chapter 4

# Demonstration

To demonstrate the capabilities of STP we created a set of applications. These can serve as an example implementation of the token negotiation, remediation, and revision procedures.

It is important to note that these applications are written for demonstrational purposes. At best, they serve as guidelines and not as strict implementational standards.

## 4.1 Implementation details

### 4.1.1 Technologies used

The demonstration consists of five packages, which together create:

- A Vendor application
- A Provider/Bank application
- And an automated test for these

We intended to make these applications as simple as possible, and the tech stack we chose mirrors that well. The Vendor and the Provider's applications consist of a back-end and a front-end. We use NodeJS with the express library for the back end to create a REST API. We decided to use the basic HTML and vanilla Javascript combination for the front-end to keep the implementation simple. In some places, dynamic content rendering was necessary; for this purpose, we chose

the EJS templating language. We chose pnpm as the package manager for NodeJS, a more performant variant of the classic npm. Finally, for our automated end-to-end test, our choice was the Selenium Webdriver instantiated with the Google Chrome browser.

### 4.1.2 Cryptography implementations used

Our demonstration used several cryptography algorithms to ensure the protocol's security. This section discusses in detail which implementations are used where.

Where it was possible, we leveraged NodeJS's built-in crypto library. This library generates cryptographically secure random data, like the token negotiation UUIDs and the token remediation and revision challenges. Furthermore, this default implementation is used for symmetric AES encryption and SHA-512 hashing.

Digital signatures are a vital component of STP. As discussed before, we chose Falcon1024 as the post-quantum digital signature scheme. However, post-quantum digital signature schemes are suggested to be combined with well-established pre-quantum schemes since the new standardization process is still in progress. Unfortunately, NodeJS does not provide a built-in implementation for post-quantum schemes yet. For this reason, it was necessary to look for alternatives. In search of an adequate implementation, we created the following criteria:

- The implementation needed to be relatively fast. WebAssembly implementations were preferred over vanilla Javascript ones
- It had to be easily integrated with our demonstration. For this reason, many other programming languages were excluded

Finally, we decided to go with `superfalcon`, which conveniently combines an Emscriptem WebAssembly wrapper for the C implementation of Falcon and another Emscriptem WebAssembly wrapper for `libsodium`'s Ed25519 implementation. It also provides a fast SHA512 implementation, which uses a native implementation if available. These combined seemed a good fit for our use case.

Unfortunately, `superfalcon` does not support other hashing algorithms, like SHA3-512. Since SHA3 also plays a key role in our protocol, we had to implement it ourselves. A `superfalcon-hash-wasm` package was created, which forks

the original `superfalcon` but replaces the `fast-sha512` with the `hash-wasm` package. `hash-wasm` is another npm package that provides hand-tuned WebAssembly wrappers for the most popular hash functions. Using this, `superfalcon`'s interface was extended to support multiple hashing algorithms for the identical Falcon1024-Ed25519 digital signatures. With `superfalcon-hash-wasm` most functions take an extra `hashType` parameter, which denotes whether SHA-512 or SHA3-512 should be used.

The `superfalcon-hash-wasm` package is not published yet on npm, but we may post it in the future after extending it with more hashing options.

### 4.1.3 The structure of the demonstration

The demonstration structure includes but is not limited to, the provider and vendor applications as the main parties of STP. Both applications follow a similar design, which was refactored multiple times throughout the thesis to handle the increasing complexity of the protocol. These applications are separated into a NodeJS Express back-end and a vanilla HTML and Javascript front-end.

Since these applications have a lot in common, these functionalities were implemented in a `common` package to reduce code duplication. Furthermore, both main components also use the custom-mentioned Falcon implementation, the `superfalcon-hash-wasm` package.

Finally, the end-to-end test of our demonstration is separated into another package. More about it in the 4.1.5 section.

### 4.1.4 Differences between the demonstration and production

#### Token negotiation

At first, on the vendor's side, we created a form to set the details of the transaction to be negotiated, so we could easily demonstrate different use cases. After selecting the amount, currency, and recurrence, we generate an STP URL. In a production scenario, the vendor would fix the transaction details, and the customer would be presented with the STP URL encoded as a QR code right at the beginning.

The STP URL is copied to the provider's application in the demonstration. Again, in a real-world use case, the QR code would be scanned by the bank's mobile application.

After entering the URL, the customer must type the verification PIN generated by the provider and shown in the vendor's app. STP supports a broader range of PINs, but we used a classic 4-digit variant here. The provider can choose how to enter the PIN code: they can present the customer with numeric input, multiple choices, or another method. In the demonstration, we use a simple numeric input.

#### Token remediation and revision

On all occasions, except the pending modification, the remediation and the revision happen without customer interaction. In the demonstration, these procedures can be initiated manually.

Both the remediation and revision happen in a single HTTPS request-response round trip. These are implemented in the `remediateToken` and `reviseToken` functions at the vendor's and provider's side, respectively.

### 4.1.5 Testing

We believe in the importance of testing as the primary method of assuring software quality. For this reason, we included tests in our demonstration, despite not having such rigorous quality requirements due to the demonstration nature of our implementation.

#### Unit testing

First, we aimed to test the basic functionalities of the commonly used utilities since these are a core part of many other parts of our demonstration. For this reason, unit tests were created for the `common` package. These unit tests cover all util functions with multiple input data to achieve excellent code coverage.

The Mocha library executes the unit tests, making it easier to perform automated tests reliably while generating reports of these executions. It is widely used in the industry and comes with many more functionalities. It is fair to say that we just

scratched the surface of this library, but we did not want to complicate our tests any more than was necessary.

Finally, the original `superfalcon` implementation has unit tests to ensure the quality of the package. We kept these tests in our `superfalcon-hash-wasm` package and tailored them to fit our use case better. Here we test for the different `hashType`-s used in the implementation. Furthermore, we moved the testing framework behind these tests from jest to Mocha to have a more unified testing environment.

#### End-to-end testing

The primary purpose of the testing was to provide us with a quick way to verify the implementation of new functionalities while checking whether we caused any regression issues. We deemed end-to-end tests to be the best-fitting candidate to fulfill this purpose.

As mentioned, we used Selenium Webdriver to power these end-to-end tests. Selenium provides a suite of tools for automating web browsers. Its API is available in many popular programming languages such as Java, Python, Javascript, etc. Furthermore, the framework supports all major browsers, such as Chrome/Chromium, Firefox, Internet Explorer, Edge, and Safari. For this reason, Selenium is still a widely used framework for all browser automation-related tasks, including but not limited to end-to-end testing. Please refer to figure 4.1 to better understand how Selenium is used in our use case.

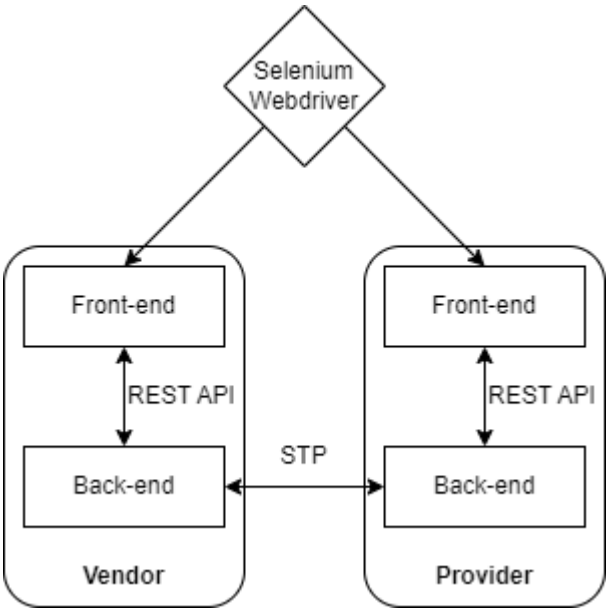


Figure 4.1: The mechanism behind the end-to-end test

We deemed Mocha an appropriate framework to execute our end-to-end tests as well. An important note here is that using the `--no-timeouts` argument was necessary to tolerate inconsistencies in the execution time caused by the nature of UI automation.

Our end-to-end tests consist of test groups, each containing test cases. The structure of the test we implemented is the following:

- Token negotiation
  - Non-recurring token

    Negotiation of a 1 USD one-time token
  - Recurring token

    Negotiation of a 1 USD monthly token
  - Recurring SHA3-first token

    Negotiation of a 1 USD annual token with the `sha3512,sha512` hash suit
- Token revocation
  - Revocation by vendor

    Executing a REVOKE remediation after a successful negotiation
  - Revocation by provider

    Executing a REVOKE revision after a successful negotiation

- Token refreshing
  - Single refresh

    Executing REFRESH remediation after a successful negotiation. Both parties receive a valid refreshed token with a recurrence index of 1
- Token modification
  - Instant accept

    Executing a MODIFY remediation after a successful negotiation. The provider accepts the request immediately
  - Delayed accept

    Executing a MODIFY remediation after a successful negotiation. The provider returns a PENDING request. A FINISH_MODIFICATION revision is executed after the user input

This way, test cases cover all token negotiation, remediation, and revision procedures to ensure the quality of our created demonstrations.

## 4.2 Benchmarks

To properly utilize STP's capabilities, many parties need to support it, providers and vendors alike. For this reason, it is essential to make the protocol performant even in a large-scale use case. So in this section, we discuss the performance results of the STP demonstration we created to provide insight into the protocol's scalability.

### 4.2.1 Methodology

The goal is to estimate how STP would work on the vendors' and providers' servers. A performance test was conducted on the demonstration applications to gather the necessary data. In the test, the server time between the incoming request and the outgoing response was measured using middleware with the help of NodeJS' `process.hrtime` function. Five measurements of each procedure with varying token properties were taken to combat instabilities mainly originating from network usage.

| Procedure | Min time (ms) | Avg time (ms) | Max time (ms) |
|---|---|---|---|
| Negotiation phase, provider* | 26.75 | 32.83 | 70.35 |
| Negotiation phase, vendor | 11.39 | 14.22 | 23.22 |
| Confirmation phase, provider* | 17.73 | 21.74 | 28.40 |
| Confirmation phase, vendor | 1.66 | 3.46 | 6.29 |

Table 4.1: Token negotiation benchmarks

After taking the measurements, the minimum, maximum, and average server times were recorded and published in the 4.2.3 subsection.

### 4.2.2 Environment

The performance test was executed on an ASUS Predator Triton 300SE machine (Intel Core i7-11370H, 16 GB RAM, NVMe SSD) running Windows 11. The servers ran on Ubuntu 22.04 under WSL2 on the same machine and communicated through the loopback network interface (localhost).

It is important to note that using WSL, the performance is slightly degraded. For more information about the performance difference between native running and WSL2, please read the following article: https://www.pugetsystems.com/labs/hpc/wsl2-vs-linux-hpl-hpcg-namd-2354

### 4.2.3 Results

The measurements are separated based on the three main procedures of STP. The benchmarks of the token negotiation procedure can be found in table 4.1, while the results of the token remediation and revision can be inspected in table 4.2 and table 4.3 respectively. All measurements include both the request validation and the processing of the request. The measurements marked with an asterisk (*) are the request sender's times and have the other party's time calculating the response.

### 4.2.4 Conclusion

The token negotiation takes 54.47 ms for the provider, meaning a provider server with similar capabilities can negotiate 18.36 tokens every second. Looking into other

| Procedure | Min time (ms) | Avg time (ms) | Max time (ms) |
|---|---|---|---|
| Revocation, vendor* | 25.69 | 26.91 | 30.06 |
| Revocation, provider | 11.73 | 12.09 | 13.21 |
| Refreshing, vendor* | 50.34 | 58.70 | 83.98 |
| Refreshing, provider | 23.92 | 25.89 | 28.56 |
| Accepted modification, vendor* | 53.21 | 60.77 | 85.62 |
| Accepted modification, provider | 24.68 | 26.73 | 28.45 |
| Pending modification, vendor* | 42.58 | 49.56 | 70.62 |
| Pending modification, provider | 14.43 | 16.30 | 18.05 |

Table 4.2: Token remediation benchmarks

| Procedure | Min time (ms) | Avg time (ms) | Max time (ms) |
|---|---|---|---|
| Revocation, provider* | 25.10 | 27.96 | 31.93 |
| Revocation, vendor | 11.31 | 12.43 | 13.77 |
| Finish modification, provider* | 43.54 | 45.18 | 47.30 |
| Finish modification, vendor | 12.57 | 13.76 | 14.97 |

Table 4.3: Token revision benchmarks

payment providers' statistics, it seems like a pleasing result. For example, Paypal handles around 193, while Visa executes about 1700 transactions per second with a much larger computing capacity. Of course, in a real-world use case, the provider and vendor servers would not run on the same machine, which would cause some extra latency. To counter this, the servers could execute multiple negotiations parallel, unlike our demonstration. From these results, we conclude that the protocol's performance is adequate for the use case we designed it for.

## 4.3 Potential improvements

Our implementation works well as a demonstration, but we acknowledge that improvements could be made, as with every software. This section lists a couple of possible improvements.

### 4.3.1 QR codes instead of URLs

To demonstrate the capabilities of STP, the demonstration displayed URLs for the client to copy over to the other application. This approach is prone to user errors, thus, inadequate for a production use case. A more elegant method would be to display a QR code on the vendor's site and read it with the bank's mobile application since most banks already have one. Implementing this in the demonstration would undoubtedly increase its quality, but we did not deem it necessary for this thesis.

### 4.3.2 Publishing STP packages

Creating implementations of STP in different popular server-side languages and publishing these with quality documentation to popular package repositories would make it much easier for any vendor or provider to adapt the protocol. Publishing STP packages to npmjs for Javascript and to pip for Python would be a great start. These packages could also be used as an example for other parties in an improved demonstration version.

# Chapter 5

# Conclusion

In this thesis, we have created a protocol to improve how online purchases and subscriptions are conducted while increasing security considerably. The proposed protocol would replace bank cards issued for individual bank account holders for an extended time with one-time use tokens that are digitally signed. This solution makes the act of online transacting more convenient and considerably increases each transaction's security.

In the Introduction 1 section, we listed core issues with bank cards that make their use less efficient and less secure. We will show how this work improves or even solves some or all of them.

- "Credit cards have relatively low entropy compared to modern cryptographic standards considered secure." As detailed in the Credit Card Entropy section, a credit card has between 33 and 46 bits of entropy. Nowadays, a minimum of 128 must be considered secure, which is insufficient. To generate a fake transaction, the attacker would either have to defeat both SHA-512 and SHA3-512 cryptographic functions or defeat the Falcon digital signature scheme, brute force a duplicate hash, that is, 1024 bits (2*512), or brute force two Falcon digital signatures that are between 512 and 810 bits of entropy. Assuming that the cryptographic schemes are not broken, an attacker's lowest complexity to brute force given quantum capabilities is 512 bits. We did not include the additional encrypted account identifier because we assume the attacker obtained at least one token. Given these numbers, the worst case entropy of our token is $2 * 10^{140}$ times as complex as the best case of the bank card.

- "Entering information manually to websites might lead to phishing attacks if a website is hijacked or a similar website is shown." With the STP protocol, the token is never shown to the user, making it impossible for them to share with attackers.

- "Breaches of vendor data might lead to exposed credit card numbers, all of which may be used and require ordering a new card to prevent." The vendor can revoke the lost tokens when they realize a breach or the provider when they get notified. The tokens are ephemeral, so 'ordering' new ones is unnecessary or can even be done automatically.

- "Card details are used multiple times; any website might use it to charge it multiple times." Tokens are ephemeral and used only once, with a new one generated for all transactions. Thus no two websites will possess the same token.

- "The buyer might not immediately be aware of the precise charge they authorize and lack the control to limit it by the transaction." The user has to authorize a transaction on their mobile phone. The bank renders the data, so the vendor may not present false information. The user will know the actual charge and whether the vendor can charge the card multiple times.

- "Man-in-the-middle attacks can be performed relatively easily against credit card purchases." The protocol is end-to-end cryptographically secured, so no party can modify the data without the transacting partners realizing it. This works even in cases where the TLS tunnel is terminated early, for example, in a microservice scenario.

- "Credit card purchases inherently lack the option to use multiple authentication factors as the implementation or lack thereof depends on the issuer bank." Mobile device authentication is a multi-factor verification mechanism using the approve button and a pin code.

- "Customers might not have an immediate way to block transactions from specific vendors or cancel subscriptions on the bank's side." The STP protocol allows for the deletion of valid tokens. If the bank provides an interface with

active subscription tokens and a revoke button for each, their customers can cancel any subscriptions from a centralized location.

- "Entering credit card details repeatedly might lead to frequent users saving them in insecure locations for easy pasting." There is no data to save on your device that might get compromised. The provider application login information is kept securely in the app vault, and it should never interact with any web applications or be exposed in a file.

From the user's perspective, the potentially unfavorable effect of using this protocol is that they have to have a mobile phone with an active internet connection. Nowadays, in Western societies, this is almost a given already.

With this in mind, we have achieved our initial goal of creating an end-to-end secured digital signature-based post-quantum-ready transaction protocol.

# Chapter 6

# Contributors

## 6.1 Authors

This is a shared work of Krisztofer Horvath Zoltan and Richard Antal Nagy. We decided to work together on this project because it was apparent from the beginning that it would require research and testing on multiple fronts of cryptography, networking, and identity verification schemes. This initial complexity was further exaggerated by the repeated iteration we've made to the protocol because of our constant cross-checking of each other's work and adversarial testing of the protocol.

Krisztofer was working on the following sections of the work:

- Digital Signatures / Post-Quantum algorithms
- Demo application implementation and cryptographic library extensions
- Application benchmarking and protocol integration testing
- Token schema and considerations

Richard was working on the following sections of the work:

- Owner of the initial concept
- Hashing / Symmetric encryption and their post-quantum contexts
- Communication protocol and network security
- Adversarial testing of application

- Documenting our findings and managing the overhead of sharing our progression

Shared work:

- Iterating on the concept with additional features
- Determining and quantifying risks to the protocol
- Cross-checking decisions

## 6.2 Special Thanks

We want to thank our supervisor Ádám István Szűcs for the assistance in both the thesis itself and the administration process.

We want to thank Ádám Nagy for his assistance in the networking aspects of the thesis, giving ideas and alternatives, and challenging the various security aspects of the protocol.

We want to thank Mohammed B. M. Kamel for his assistance in the cryptography aspects of the thesis, especially related to post-quantum cryptography.

We want to thank Attila Kovács for providing us with helpful contacts and associates who helped challenge aspects of the protocol and iterate on it.

We also thank our teachers and lecturers who deepened our knowledge and helped us create this thesis.

# List of Figures

# Bibliography


[1] David Robertson. "Nilson Report". In: *https://nilsonreport.com* (2021).

[2] NIST. "Post-Quantum Cryptography Standardization". In: *https://csrc.nist.gov/projects/post-quantum-cryptography* (2016).

[3] NIST. "NIST's Hybrid Mode Approach to Post-Quantum Computing". In: *https://utimaco.com/news/blog-posts/nists-hybrid-mode-approach-post-quantum-computing-why-crypto-agility-crucial* (2019).

[4] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. *Trapdoors for Hard Lattices and New Cryptographic Constructions*. Cryptology ePrint Archive, Paper 2007/432. `https://eprint.iacr.org/2007/432`. 2007. URL: `https://eprint.iacr.org/2007/432`.

[5] Léo Ducas and Thomas Prest. *Fast Fourier Orthogonalization*. Cryptology ePrint Archive, Paper 2015/1014. `https://eprint.iacr.org/2015/1014`. 2015. URL: `https://eprint.iacr.org/2015/1014`.

[6] Philip Klein. "Finding the closest lattice vector when it's unusually close". In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. 2000, pp. 937–941.

[7] Chris Peikert. "An efficient and parallel Gaussian sampler for lattices". In: *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer. 2010, pp. 80–97.

[8] Pierre-Alain Fouque et al. "Falcon: Fast-Fourier lattice-based compact signatures over NTRU". In: *Submission to the NIST's post-quantum cryptography standardization process* 36.5 (2018).

[9] Microsoft. "Post-Quantum TLS". In: *https://www.microsoft.com/en-us/research/project/post-quantum-tls/* (2021).