Publicly Auditable Privacy Revocation in Practice

Wilmer Nilsson and Patrik Kron
wlmr@pm.me and dic15pkr@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Paul Stankovski Wagner

Advisor: Joakim Brorsson

Examiner: Thomas Johansson

July 7, 2021

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

PAPR Credentials is the name of a not yet published credential scheme that provides conditional privacy for its users [1]. This means that an administrator can choose to revoke a user's anonymity. Doing so, however, the administrator must broadcast her intention, as the success of the privacy revocation depends on the help of other users, since the administrator does not have access to sufficient information to do it alone.

In this thesis, we provide a PAPR Credentials implementation that we use to examine system characteristics. We focus on how different system parameters impact three properties: the computational load for an arbitrary user, revocation delay, and when revocations begin to fail due to the number of previous revocations.

On top of PAPR, we construct an example application called PAPR Money, in which the users can transfer bitcoin [2] to one another pseudonymously. In case of suspected money laundering or terrorist financing, the underlying PAPR Credentials scheme facilitates the ability to revoke anonymity.

We have also produced the first implementations of Schoenmakers' *Publicly Verifiable Secret Sharing scheme* (PVSS) [3] in Python that we are aware of. Schoenmakers' PVSS is a crucial component of PAPR Credentials.

Our results show how the computational load changes with different sets of system parameters and how the system parameters are key to determining both revocation delay and revocation failure.

Acknowledgements

We want our supervisor, Paul Stankovski Wagner, to know that he has all our gratitude for the feedback he provided us with, which ultimately led to this thesis becoming readable.

We also want to thank doctoral student Joakim Brorsson, who was always there for us, just a Slack-message away. His expertise, explanations and help were essential to the successful completion of this thesis.

Without the two of you, we would never have made it. So, once again, thank you.

Popular Science Summary

How often do authorities misbehave? This is tricky to answer as most systems lack transparency. Hence, as a user, you can neither know if the system is abused by its users nor by its owners. As an end-user, you are, however, forced to take it or leave it. The only incidents of misbehaviour we know of have required whistleblowers with authority enough to see what is really going on, e.g. Edward Snowden. Whistleblowers are few and far between, and one can only ponder what service providers are up to. If the service provider does not publish the service's source code—how can the end-user trust the service provider? How can the end-user trust that her data is not being sold, for instance?

One form of questionable conduct is the non-justified deanonymization of users in systems that allow for conditional privacy. To exemplify such a case of malpractice in a practical setting, we imagine a conditionally anonymous digital currency platform, where the bank providing the service has the power to revoke its customers' privacy. They claim only to use this power when suspecting money laundering or terrorist financing. One particularly problematic aspect of such a closed system is that it has no built-in protection mechanism that stops the bank from deanonymizing users arbitrarily.

PAPR Credentials [1] is an anonymous credential scheme with a twist, namely that it prevents authorities from carrying out covert revocations. A user wanting to obtain a valid anonymous login PAPR credential splits up its real identity into n shares in such a way that less than some number of k shares reveals nothing about the real identity. The user and authority then agree upon n randomly selected users and, for each selected user, encrypt a share with the selected user's encryption key. Hence, the authority needs the help of other users to revoke a user's anonymity, which forces it to broadcast every revocation request, making it impossible to revoke users under the radar.

PAPR Credentials has never been implemented. In this thesis, we implement the scheme and examine its feasibility by measuring the duration of setup given different parameters, how many users can be revoked before revocations no longer are reliable, as well as how users' usage patterns affect the time it takes until a requested revocation completes. We also provide an example application of PAPR Credentials called PAPR Money. PAPR Money is a digital currency platform much like the one described above, within which trading of bitcoin [2] is facilitated, on top of PAPR Credentials.

Table of Contents

List of Figures

Chapter 1

Introduction

The term security refers to protection from outside forces. Confidentiality, integrity and availability are central aspects of a secure system when regarding data. To increase security, a cryptographic system might, for instance, require the user to identify herself with several metrics to robustly authenticate that the user is whom she claims to be.

In the same context, privacy, on the other hand, regards how secret one's data is. If an external entity cannot deduce anything by looking at the data, then it is private.

If a user is required to identify herself, privacy falters, as the system knows who is responsible for all data.

Clearly, there is a trade-off between these two concepts, and real-world systems are often required to find a suitable middle ground.

One example of where such a middle path needs to be achieved can be found in cryptographic systems where authorities have the ability to revoke privacy [4, 5]. It might also be necessary for legal reasons. For example, if a user chooses to break any of the agreed-upon rules, an authority might reveal the identity of said user to the real-world authorities.

The problem with these applications is that there has been no way of knowing, as a user, if one's privacy has been breached. This creates a lack of trust in the privacy revoking authorities. There is nothing technical that stops the privacy revoking authority from misusing their abilities. Therefore, the authority could recall the privacy of whomever they want, without the public or the users knowing.

A common strategy for coping with the above-mentioned problem is to enlist a trusted third party (TTP) that acts as a supervisor of the authority. However, even a TTP cannot be completely trusted as the TTP and the authority might be colluding. The lacking trust is still present even if we have multiple TTP's, since those could still collude.

A potential solution to this lack of trust is to cryptographically force the authorities to publicly announce every privacy revocation. Fortunately, a new yet unpublished paper addresses this issue and presents a solution without using a single TTP.

1.1 PAPR Credentials

"PAPR Credentials - Conditional Privacy with Publicly Auditable Privacy Revocation" [1], is the title of a paper introducing the notion of Publicly Auditable Privacy Revocation. A credential scheme with this property must provide both conditional privacy of its users as well as public verifiability of correct behaviour of the privacy revocation authority. Two suggested constructions of a credential scheme with this property are provided, a pseudonymous and an anonymous. The former is the one covered in this thesis and will be briefly explained in the following paragraphs. From now on, when writing PAPR, we mean the pseudonymous credential scheme with the PAPR property presented in the paper [1].

A user of the pseudonymous version of PAPR will have a *pseudonym*, also known as a *credential*, which they can use when authenticating to services that use PAPR. This pseudonym proves to another user, or a third party, that she, in fact, is a valid user, i.e., her credential corresponds to one of the enrolled real identities, but it does not tell the other user who she is. If a user were to break any law whilst authenticated with their pseudonymous credential, the issuing authority could create a revocation request, where a select number of other users need to participate in order for the revocation to complete. If enough users answer, the identity is revealed (for the authority), and legal action can be taken. The system users can see how many revocations have been made since the authority must publish each credential that she revokes in order to receive answers.

PAPR is the first construction that is able to cryptographically force the authority to publicly announce every privacy revocation, and it does so without any trusted third parties.

The main idea behind PAPR is to not let the authority alone have access to sufficient data for revocations. This will force the authority to ask for help with privacy revocation requests, in effect forcing the authority to announce every revocation request. The following describes how it is done.

When a user sets out to create a credential and get it acknowledged by the authority, the user and the authority cooperatively select n random users. These users are known as the user's *data custodians*. The user's real identity is then chopped up into n shares, which are distributed to the custodians, each share encrypted with a particular custodian's public encryption key. The chopping up of the real identity is carried out in such a way that with less than k shares, nothing can be said of the real identity. This part of the scheme is called *data distribution*.

1.1.1 What are privacy revocations good for?

The privacy revocation is performed by an authority to look up a pseudonym of a user and receive the user's real identity. A privacy revocation can be carried out if a user does not follow the rules or even breaks the law.

In PAPR, an authority needs to broadcast whom they wish to revoke the privacy of, since it is the users that hold the decryption keys for other users' encrypted identities. This makes sure an authority cannot misuse their ability to revoke users' privacy without it being known she does so.

1.1.2 Use Cases

Digital Currency

When paying with cash, transactions are anonymous. Paper money cannot easily be traced, nor is there anything linking the holder to her bills.

These days we are drifting further and further from anonymity as people use card payments to an increasing extent. Why is this? The answer is probably simple; card payments offer more convenience.

How can we marry the anonymity of cash with digital currency convenience? This is not a new question, and solutions have been suggested since the 1980s [6].

Bitcoin [2] is a contemporary solution. It does not link one's identity with the payment, thus allowing for transactions to be made pseudonymously, although the transactions themselves can be linked in most cases. Authorities, worried that cryptocurrencies could be used for money laundering, have required exchanges to demand user information, effectively rendering Bitcoin's pseudonymity useless. Other more sophisticated solutions exist. Monero [7] is one such example. This cryptocurrency is completely anonymous. However, that makes the authorities even more concerned about the risk for money laundering and terror financing. In the EU, several requirements must be met before being allowed to issue any form of anonymous payment alternative [8].

Since PAPR allows for users to remain pseudonymous, it can be used for cash-like payments. With some added information on each customer, PAPR could meet the EU requirements since PAPR is conditionally private and the authority can always, with the help of other users, revoke a user's privacy if there is a suspicion of illegal activities.

Vehicular Ad-Hock Networks

Another use case for PAPR can be found in Vehicular Ad-Hock Networks (VANETs), where cars communicate with each other (and maybe also with infrastructure). In these systems, enrolled vehicles can broadcast their authenticated positions to nearby vehicles. The regulating authorities have specific requirements for such networks. Among these are that the networks require privacy, user accountability and pseudonymity [1].

Currently proposed VANET systems introduce two authorities that need to cooperate to revoke a vehicle's identity. These two authorities have no public accountability, meaning they are not forced to reveal which revocations they have made. Such a system also requires trust in these two authorities [1].

A solution involving PAPR requires no trust in authorities, as the authority in a PAPR-based system is forced to be transparent.

1.2 Problem Statement

One concern with PAPR is that it has never been implemented. This thesis remedies that concern by implementing a PAPR Credentials scheme. The implementation is then used to run simulations. With these simulations, we aim to answer a few questions:

1.2.1 Research Questions

The following research questions have guided this thesis work. How user patterns, threshold number and committee size affect PAPR in terms of

- *user computational load* - The issuance of credentials can be particularly taxing for certain parameters. We want to measure how long the duration of the credential issuance procedure is, for a user, for different values of committee size and threshold number.

- *how many revocations are possible* - In a PAPR Credentials system, where several revocations already have been carried out, there might be a point where future revocations become unsuccessful. This is due to the fact that previously revoked users also act as custodians for other users. Once a user is revoked, she has no incentive for helping out with future revocations. If too many custodians of a specific user already have been revoked, that user is effectively unrevokable. Is this a problem in practice, and how do we set the parameters so that the risk of this happening is negligible?

- *delay of revocations* - Once a revocation request is published, its subsequent success relies on the help of the custodians actually logging in and helping out. How long time does it take from the revocation request being published until enough custodians have submitted their decrypted shares and the revocation can complete?

These research questions are specifically addressed and answered in Chapter 5.

1.3 Contributions

This thesis mainly contributes the following:

- *An implementation of PAPR*: We provide the first-ever implementation of PAPR Credentials. Doing so includes implementing the two main subcomponents: Shoenmakers' publicly verifiable secret sharing scheme (*PVSS*) [3] and the keyed-verification anonymous credentials scheme, by Chase, Meiklejohn and Zaverucha, that we refer to as *CMZ* [9].

- *A Schoenmakers' PVSS implementation in Python*: As far as we are aware, this has not been done before. There exists an implementation in Go [10].

- *A proof of concept of a pseudonymous bitcoin trading application on top of PAPR*: We present a proof-of-concept application of PAPR in the form of an application called PAPR Money, capable of sending bitcoin within a network of PAPR users.

- *A practical analysis of PAPR*: We provide a practical analysis of PAPR in terms of the research questions outlined above.

1.4 Outline

Chapter 2 provides the background knowledge needed to understand PAPR Credentials and the thesis in general. Chapter 3 provides a more detailed explanation of PAPR Credentials, its parts and how each part works.

Chapter 4 explains our implementation of PAPR Credentials, PAPR Money and our simulations on our PAPR Credentials implementation. In Chapter 5 we analyse the results of our simulations on the basis of our questions mentioned above. Lastly, in Chapter 6 we wrap up the thesis, conclude it and discuss open questions left to explore.

Chapter 2

Theoretical Background

To understand this thesis in general, and PAPR Credentials in particular, we need to introduce a couple of topics. ElGamal encryption, the Elliptic Curve Digital Signature Algorithm (ECDSA), Schnorr's Zero-knowledge proof, Shoenmakers' Publicly Verifiable Secret Sharing scheme (PVSS), an anonymous credential scheme, the Pedersen commitment scheme and two electronic payment systems.

The anonymous credential scheme provides the functionality that allows users to enroll themselves in a PAPR Credentials system. A user enrolls by submitting her real identity. She is then given an anonymous token, which proves that she is one of the enrolled users, but not which one. Throughout the anonymous credential scheme's procedures, ElGamal encryption and Pedersen commitments both play an essential part, and so does Schnorr's zero-knowledge proof.

Shoenmakers' PVSS is yet another key component of PAPR Credentials. It is this component that strips the authority of the possibility of revoking users' privacy by herself.

PAPR Credentials rely on digital signatures for its authentication procedure; thus, we provide an overview of the Elliptic Curve Digital Signature Algorithm.

As our PAPR Money application deals with real value transactions, we deemed it important to verify the plausibility of such a system, with regards to the law. Hence a primer on the EU laws regarding money laundering is provided. We also contribute a short history lesson on electronic payment systems.

Below we will make use of the notation $x \xleftarrow{\$} S$, where S is a set. If we assume $S = \{1, 2, \ldots, 10\}$, then we say that x is randomly assigned a value from 1 to 10.

2.1 ElGamal Encryption

ElGamal is an asymmetric encryption scheme with several similarities to the Diffie-Hellman key distribution scheme. The security of ElGamal is based on the Diffie-Hellman problem and the intractability of the discrete logarithm problem. ElGamal involves three steps, namely *key generation*, *encryption* and *decryption*. The three steps are explained below.

2.1.1 Key Generation

Alice wants Bob to send her an encrypted message, and so she generates a key and sends it to Bob according to the following steps.

1. Generate a description of a cyclic group G of prime order q and find a generator g.
2. $x \xleftarrow{\$} \{1, \ldots, q-1\}$
3. compute $h = g^x \pmod q$

Alice's public key is (G, q, g, h). She sends her public key to Bob. Alice's private key is x, which she keeps secret.

2.1.2 Encryption

Bob receives Alice public key (G, q, g, h) and encrypts his message m as follows.

1. compute $s = h^y \pmod q = g^{xy} \pmod q$
2. compute $c_1 = g^y \pmod q$
3. compute $c_2 = m \cdot s \pmod q = m \cdot g^{xy} \pmod q$

Bob now sends Alice the encrypted message (c_1, c_2). It should be noted that, if an enemy, Eve, somehow were to compromise the plaintext message, m, and the ciphertext, (c_1, c_2), then she could easily determine the shared key $s = g^{xy}$, since $c_2 \cdot m^{-1} = s$. To remedy this, a new y is chosen for each new message, and hence forward secrecy is achieved.

2.1.3 Decryption

1. compute $s = c_1^x \pmod q = g^{xy} \pmod q$
2. compute $s^{-1} \pmod q$
3. compute $m = c_2 \cdot s^{-1} \pmod q$

2.2 ECDSA and Digital Signatures

By signing a message, m, be it a file or simply a number, Alice uses her private key d_A along with a signing algorithm. Anyone can then use the signature to verify that Alice sent the message m, and that it has not been tampered with, given that they know Alice public key, Q_A.

The Elliptic Curve Digital Signature Algorithm is a signing algorithm based on the Digital Signature Algorithm (DSA), that, instead of using modular exponentiation, uses elliptic curve cryptography. As with most algorithms working with elliptic curves, the key size is smaller than a DSA key for the same level of security [11].

ECDSA has two functions:

- $sign(curve, d_A, message) \rightarrow signature$
- $verify(curve, signature, Q_A, message) \rightarrow bool$

2.2.1 Sign

Consider the method
$sign(curve, d_A, message) \rightarrow signature$,
where the *curve* parameter is a description of a specific elliptic curve that includes a generator g, and the integer order of g, n. Let H be a cryptographic hash function.

1. $e = H(message)$
2. $k \xleftarrow{\$} \{1..n-1\}$
3. $(x_1, y_1) = k \times g$
4. $r = x_1 \mod n$ (if $r = 0$ then go back to step 2)
5. $s = k^{-1}(e + rd_A) \mod n$ (if $s = 0$ then go back to step 2)
6. return the signature (r, s)

2.2.2 Verify

Consider the method
$verify(curve, signature, Q_A, message) \rightarrow bool$,
where the *curve* parameter is a description of a specific elliptic curve that includes a generator g, and the integer order of g, n. Let H be a cryptographic hash function, and Q_A be the public counterpart of the private key used to create the signature.

1. $e = H(message)$
2. $u_1 = es^{-1} \mod n$
3. $u_2 = rs^{-1} \mod n$
4. $(x_1, y_1) = u_1 \times g + u_2 \times Q_A$
5. The signature is valid if r is equal to x_1 and if so the algorithm returns *true*, otherwise *false*

Explanation:

$$(x_1, y_1) = es^{-1} \times g + rs^{-1}d_A \times g = (es^{-1} + rs^{-1}d_A) \times g = (e + rd_A)s^{-1} \times g$$

$$(e + rd_A)s^{-1} \times g = (e + rd_A)(e + rd_A)^{-1}(k^{-1})^{-1} = k \times g$$

2.3 Schnorr's Zero-Knowledge Proof

In his paper from 1990 [12], C.P. Schnorr presents both an identification scheme and a signature scheme for smart cards. The protocol is built upon Chaum, Evertse and Graaf's protocol [13]. This protocol features a type of zero-knowledge proof (ZKP) that is utilised in PAPR.

The protocol consists of an identity (I, v) signed by a key authentication centre (KAC), where I is an identification string, and v is a public key of a user.

Computing v is done from the users private key s such that $v = \alpha^{-s}$, where α is a publicly known generator of the group $\mathbb{Z}_p$, and p is a publicly known prime. The last piece of the puzzle is the publicly available prime q, which is chosen so that $q|p-1$.

To become registered in the system, a user must first go to a KAC, bringing her public key v. The KAC will generate an identity string I (containing name, address, ID-number etc.). The KAC then proceeds by digitally signing the tuple (I,v). The part of the protocol which is relevant to PAPR, however, is the one associated with authentication, namely the identification scheme. It works as follows.

2.3.1 Identification Scheme

When proving her identity, Peggy will convince Victor that she knows the discrete logarithm with base α of v^{-1}, i.e. $-\log_\alpha v = s$. In other words, the prover, Peggy, will prove that she knows the private key associated with her public key.
Peggy begins by sending her identity (I,v) with the KAC signature to Victor. She then picks a random number $r \xleftarrow{\$} \{0, \ldots, q-1\}$, computes the commitment $C = \alpha^r \pmod p$ and sends C to Victor. Victor responds with random number $e \xleftarrow{\$} \{0, \ldots, 2^t - 1\}$ to Peggy (where t determines the t-bit-security of the scheme). Peggy sends $y = r + s \cdot e \pmod q$. This looks like a random value to Victor but when used as an exponent for α, it can aid Peggy in proving her identity, as you will see in the next step.

To verify the identity, Victor checks that $C = (\alpha^y \cdot v^e) \pmod p$. If this is true then Peggy must know s such that $v = \alpha^{-s}$, hence the prover must have access to Peggy's private key. This works as $\alpha^y \cdot v^e \pmod p = \alpha^r \cdot \alpha^{se} \cdot \alpha^{-se} \pmod p = \alpha^r \pmod p$. This method of authentication allows Peggy to prove that she is in possession of her private key, without revealing it to Victor. Such proofs are generally called zero-knowledge proofs.

2.4 Schoenmakers' Publicly Verifiable Secret Sharing scheme

A secret sharing scheme is an important cryptographic primitive that can be used when a dealer wants a secret split up and distributed among a set of n participants, in such a way that no group less than k of those participants, can reconstruct the distributed secret. In the most simple secret sharing schemes, there are no ways to verify that

(i) the dealer is handing out valid shares, or

(ii) that the participants are returning valid shares in the reconstruction process.

Hence, the notion *of verifiable secret sharing* (VSS) was introduced. In VSS schemes, the above two problems are taken care of in the sense that the participants can verify their own shares. Publicly verifiable secret sharing (PVSS) is a secret sharing scheme that allows *anyone* to verify that shares are generated correctly.

In the PVSS scheme proposed by Shoenmakers [3], no private communication channels can be used to distribute the shares, as everything needs to be publicly verifiable. Therefore, all communication takes place over a public channel. This makes it possible for everyone to verify that both the dealer and all participants behave correctly.

Shoenmakers' PVSS scheme is used to share a secret with n participants, in such a way that a subset of at least k of the n participants can cooperate to retrieve the secret (a (k,n)-threshold scheme), but any group with less than k participants receives no information about the secret. The scheme also provides proof that all shares of a secret are valid and that they are, in fact, shares of the same secret, in such a way that anyone can verify the proof. The scheme is also enabling the shareholders to prove that, at the reconstruction process, they are providing a valid share. In the paper, Shoenmakers also presents an electronic voting scheme based on this publicly verifiable secret sharing scheme, which we will not use in this paper.

2.4.1 PVSS in Practice

The dealer picks a random polynomial $p(x)$, where α_j is in $\mathbb{Z}_q$ and $j \leq t-1$. The secret which is to be distributed is α_0.

$$p(x) = \sum_{j=0}^{t-1} \alpha_j x^j = \alpha_0 + \alpha_1 x^1 + \alpha_2 x^2 + \ldots + \alpha_{t-1} x^{t-1}$$

All C_j are commitments.

$$C_j = g^{\alpha_j} \qquad \forall j \in 0 \leq j < t$$

Encrypting $p(i)$ we get the encrypted shares Y_i:

$$Y_i = y_i^{p(i)} \qquad \forall i \in 1 \leq i < n$$

The dealer wants to prove that the commitments, C_j, and the encrypted shares, Y_i, were constructed from the same secret. The dealer will therefore use a zero-knowledge proof that we call *DLEQ*, which we explain in Section 2.4.2. To show that the commitments and shares are constructed from the same secret, a helper variable, X_i, is constructed as follows:

$$X_i = \prod_{j=0}^{t-1} C_j^{i^j}$$

Now we can rewrite is as:

$$X_i = \prod_{j=0}^{k-1} C_j^{i^j} = C_j^{i^0} \cdot C_j^{i^1} \cdot C_j^{i^2} \cdot C_j^{i^3} \cdot C_j^{i^4} \cdot \ldots \cdot C_j^{i^{k-1}} = g^{\alpha_0^{i^0}} \cdot g^{\alpha_1^{i^1}} \cdot g^{\alpha_2^{i^2}} \cdot g^{\alpha_3^{i^3}} \cdot \ldots \cdot g^{\alpha_{k-1}^{i^{k-1}}} =$$

$$= g^{\alpha_0 + \alpha_1^i + \alpha_2^{i^2} + \alpha_3^{i^3} + \ldots + \alpha_{k-1}^{i^{k-1}}} = g^{\alpha_0 + i\alpha_1 + i^2\alpha_2 + i^3\alpha_3 + \ldots + i^{k-1}\alpha_{k-1}} = g^{p(i)}$$

By using $DLEQ(g, X_i, y_i, Y_i)$, the dealer generates a zero-knowledge proof showing that the exponent α_i in $Y_i = y_i^{\alpha_i}$ is the same as the exponent α_i of $X_i = g^{\alpha_i}$ for all i and thus that the encrypted shares are consistent.

2.4.2 Proof of Equal Exponents

Chaum and Pedersen [14] present a protocol that shows how to prove, in zero-knowledge, that two bases, a_1 and a_2, share the same exponent, w. We call the protocol *DLEQ*, as that is what it is referred to in Shoenmakers' PVSS [3]. The protocol *DLEQ* takes the parameters g_1, h_1, g_2 and h_2. We want to show that we know an α such that $h_1 = g_1^\alpha$ and $h_2 = g_2^\alpha$. This interactive procedure is carried out in the following way.

1. Prover: Let $a_1 = g_1^w$ and $a_2 = g_2^w$, $w \xleftarrow{\$} \mathbb{Z}_q$
2. Verifier: Take $c \xleftarrow{\$} \mathbb{Z}_q$
3. Prover: $r = w - \alpha c \pmod q$
4. Verifier: Verify that $a_1 = g_1^r \cdot h_1^c$ and that $a_2 = g_2^r \cdot h_2^c$

In Shoenmakers' PVSS, a non-interactive version of *DLEQ* is used. Instead of choosing c randomly, two non-interactive methods for choosing c are used. The methods are presented below.

The first method is used when the dealer wants to show that her distributed shares are consistent with her commitments. The dealer repeatedly calls $DLEQ(g, X_i, y_i, Y_i)$, for each $0 \leq i \leq n$. In this method, the same c is used for all i. As c we use a hash of all X_i, Y_i, a_{1i} and a_{2i}.

The second method is used when a participant wants to prove that she has correctly decrypted a share. Then $DLEQ(G, y_i, S_i, Y_i)$ is called. In that method c is the hash of y_i, Y_i, a_1 and a_2, for that participant's index, i.

2.5 Anonymous Credentials

The building block in PAPR that allows for anonymous credentials is a scheme due to Chase, Meiklejohn and Zaverucha, which we refer to as CMZ [9].

In their paper, the authors present a new type of message authentication code (MAC), called an algebraic MAC. This type of MAC is based on algebraic group operations (such as modular exponentiation), instead of the more conventional block cipher. They introduce this type of MAC as it integrates seamlessly with cryptographic primitives such as zero-knowledge proofs, since such proofs are often also based on group operations.

Following the introduction of the algebraic MAC, the authors show how it can be used to generate keyed-verification anonymous credentials. In PAPR, only one key is used for verification, although CMZ can be extended to use multiple keys.

CMZ requires that the credential issuer trusts the credential verifier to the degree where they can exchange a secret key. In PAPR, the verifier and the issuer are the same entity; hence this is not an issue.

The two most interesting procedures of the anonymous credential protocol are arguably the credential issuance and credential verification procedures. These also happen to be vital to PAPR and are therefore described in some detail below.

2.5.1 Credential Issuance

A credential certifies that the owner has access to some set of attributes. An attribute can, e.g., be a name or an age. In the paper by Chase, Meiklejohn and Zaverucha [9], two types of credential issuance methods are presented.

The first type of credential issuance requires sharing the underlying attributes with an issuer. As described in the paper, the issuer calculates the algebraic MAC using the attributes and the issuer's secret key as input. The MAC can then be used as a credential.

The second type of issuance allows the user to keep secret some, or all, of the attributes. This second type is made possible by the homomorphic properties of ElGamal. Firstly a user generates an ElGamal keypair $(d,\gamma := g^d)$, which she then uses to encrypt every attribute m_i as $(g^{r_i},g^{m_i}\gamma^{r_i})$. These are sent to the verifier along with a proof of knowledge of r_i and m_i. It is now time for the homomorphic encryption magic.

Without the need or ability to decrypt the ciphertext, the issuer modifies the ciphertext in such a way that the underlying plaintext is altered in a meaningful way. In essence, what previously was a set of ciphertexts of g^{m_i}, has now become one ciphertext of $u' = g^{bx_0}\prod_1^n (g^{m_i})^{bx_i}$ where b is randomly chosen from $\mathbb{F}_p$, by the issuer, and the x_i's are part of the issuer's secret key. The ciphertext is then further altered by multiplication with an encryption of 0 before being sent back to the user, along with $u = g^b$ and zero-knowledge proofs that the credential has been generated correctly, using the actual secret keys.

The user can now verify the proofs. If they add up, according to publicly available commits of the issuer's secret keys, she can decrypt the ciphertext and obtain her credential (u,u'). This credential can then be used as a pseudonym.

2.5.2 Credential Verification

Now that the user has a credential, (u,u'), she can use this as proof that she is in possession of one of the registered keypairs. She will use this to authenticate herself anonymously. The authentication procedure focuses mainly on verifying that the user's credential certifies attributes that match the attributes in a set of Pedersen commitments published by the user. If they do, the user is authorised as "someone with access to some registered keypair".

2.6 Pedersen Commitment Scheme

The Pedersen commitment scheme is a commitment scheme that is computationally binding and information-theoretically concealing. The committer wants to commit x. The committer calculates $B_a(x) = h^x \cdot g^a$, where h and g are verifiably random, and a is a blinding value that makes sure that the commitment remains concealed even to an adversary with unlimited computing ability [11].

One way to generate g (a similar method can be used to generate h) is to do as follows [11]:

Let H be a cryptographic hash function.

1. $r \leftarrow \mathbb{Z}$
2. $f \leftarrow H(r) \in \mathbb{F}_p^*$
3. $g \leftarrow f^{(p-1)/q} \pmod{q}$

Repeat until $g \neq 1$, return (r, g)

2.7 Electronic Payment Systems

The following sections explain previous versions of electronic payment systems, how they work, and their drawbacks.

2.7.1 E-Cash

Digital payments are often inherently nonprivate, since a third party (for example, a bank) usually handles the transactions and have full visibility in who pays whom. Chaum [6] proposed an electronic cash scheme in which the third party cannot determine the payee, time or amount of payments made by an individual.

First, Alice deposits money at the bank and asks for e-cash. She receives e-cash that she can give to Bob, which he later can cash in. For this to work, multiple problems have to be solved. How can Bob know that the e-cash he receives is issued by the bank? How can Bob know that the e-cash he receives has not already been spent? How can we make sure that the bank cannot trace who paid whom upon receiving the e-cash?

Chaum solved all of these problems by letting Alice create the e-cash by generating a random number, "blinding" the number and sending the blinded value to the bank. The bank uses a *blind signature* to sign the e-cash and sends it back to Alice. Alice can now remove the blinding factor of the bank's answer in such a way that the signature is now valid for the underlying random number. This signature is her banknote. She sends her note to Bob. When Bob receives it, he can verify the bank's signature and trust that the banknote is valid. When Bob then exchanges the note at the bank, the bank knows that it is valid since it has its signature on it, and they have not received it earlier. Since the bank signed the blinded value earlier, they have not seen the unblinded value before, and can not connect it back to Alice. One problem remains, however. Nothing hinders Alice from spending the same coin multiple times (double-spending) before those who received it cash it in. Thus for this scheme to work, Bob must talk to the bank as soon as he receives the note to verify that it has not yet been double-spent, before carrying out his transaction with Alice.

Camenish et al. [15] presented a paper that solved the above problem where the receiver, Bob, had to contact the bank, directly after being paid by Alice, to ensure that Alice had not spent the money beforehand. The paper describes a scheme in which Bob can contact the bank at any point in the future to cash in the coins. If the coins are valid, i.e. not yet cashed in, all is good. If they turn out invalid, Alice can be identified, and all transactions she has done with the current wallet will be linked. As long as Alice does not double spend, she will, however, remain unidentified.

2.7.2 Bitcoin

E-cash relies on a third party (for example, a bank) that has to be trusted to, for example, not issue more e-cash than have been deposited with them. E-cash also relies on the third party to hinder or at least detect double-spending. This is one of the problems that Satoshi Nakamoto solved when he invented Bitcoin [2].

Instead of relying on a single trusted third party, each participant relies on the assumption that honest third parties control more computing power than any single group of dishonest third parties. In e-cash schemes, a trusted third party determines who got paid first and decline subsequent usage of that note. In Bitcoin, all transactions are broadcasted to the network and logged in a distributed ledger containing all transactions in the order they were made, allowing anyone to verify that no bitcoins are double-spent. In short, the Bitcoin network works by having all new transactions sent to all nodes on the network. Some of these nodes are called miners. These collect transactions and bundle them together in blocks which they aim to get accepted into the above mentioned public ledger. This ledger is, in fact, a chain of such blocks. For a miner to get her block accepted and put on the blockchain, she has to find a solution to a computationally heavy task, known as a proof-of-work, before any other miner does. If some other miner beats her to it, then that other miner's block will be published, and she has to begin all over again, since the task changes when a new block is added.

Forks

Let's say the two miners, Michael and Millie, manage to find a solution each at roughly the same time. If 50% of the network receive Michael's block first and the other half gets Millie's first. Then there will exist a temporary "fork" in the blockchain. Miners continue work on the block they received first. If miners later receive a block that is a continuation of the block they previously disregarded, then they switch to work on the longer blockchain, as the longest blockchain is considered to be the valid one [2].

What happens if a group of dishonest third parties controls more than 50% of the computing power?

Majority Attack

Let's for a minute assume that our miner, Eve, is evil. Let's also assume that Eve just bought a tulip in exchange for bitcoin. Since she is evil, she will attempt to fool the tulip vendor. For this to work, Eve must have access to more than 50% of the computing power of all the miners, so let us assume that she does. While her tulip transaction goes into another miner's block, she secretly begins mining blocks that do not contain this transaction. Normally, as the tulip vendor, you prefer to wait until the block containing the tulip transaction has a certain number of blocks succeeding it to avoid the problem illustrated with the help of Michael and Millie. Eve is aware of this and secretly starts building her own continuation of blocks. Since she has more computing power than the rest of the network together, her chain will increase in length faster than the one created by everyone else. When the vendor is happy with the number of succeeding coins, she gives Eve her tulip,

and she proceeds to publish her fork, which the network then will consider valid, as it is longer, in effect removing her payment from the blockchain. Eve has now gotten a tulip for free.

2.8 Anti-Money Laundering Laws in the EU

In 1991, the first anti-money laundering directive (AMLD) was passed. Since then, a few more AMLDs have followed. As of today, in 2021, the first AMLD has been joined by five later ones [16].

The directives have introduced a multitude of new laws. Amongst other things, they have introduced the obligation for the EU member states to create a centralised register of so-called "Ultimate Beneficial Owners" (UBOs), containing detailed information on the legal entities registered in EU member states bundled with information on the entity's owner. More importantly for this thesis, however, are two other concepts related to countering terrorism financing and anti-money laundering in general that will each be given their own section below.

2.8.1 Customer Due Diligence

In the context of finance, "due diligence" is the process of examining the financial records of a potential business partner. To prevent money laundering, the EU has placed a heavy emphasis on customer due diligence.

Let us imagine that we, at our disposal, have a banker, Bob, a customer, Carol, and a company Acme. Let's assume Carol, for the first time, wants to initiate a trade with Acme, with the help of Bob. Bob is then forced to apply customer due diligence measures. Through these measures, Bob must obtain knowledge of Carol's identity, the UBO of Acme, the purpose of the transaction, amongst other things.

The EU member states prohibit their financial institutions from keeping anonymous accounts [8]. Any already existing anonymous accounts must be subject to customer due diligence procedures.

2.8.2 Suspicious Transaction Reporting

There are multiple requirements when it comes to reporting suspicious transactions, presented in the AMLDs, most of which can be found in AMLD IV [8]. Some of the requirements follow below.

Each EU member state must have a Financial Intelligence Unit (FIU) that shall prevent and detect money laundering and terrorist financing.

The FIU shall be able to suspend or withhold a pending transaction in order to analyse it and communicate the result to relevant authorities, if the transaction is suspected to be related to money laundering or terrorist financing.

All EU member states must collect and analyse links about suspicious transactions and criminal activity. All suspicious transactions should be reported, regardless of the amount of the transaction.

There is an exception that allows suspicious transactions to take place if stopping them would make it harder to pursue the suspected money laundering or

terrorist financing operation. Although, if by UN resolution, the funds of terrorists or a terrorist organisation have been requested to be "frozen without delay", that overrides the exception, and the transactions should therefore be stopped even if it makes it harder to pursue the terrorists.

Chapter 3

Publicly Auditable Privacy Revocation

"PAPR Credentials - Conditional Privacy with Publicly Auditable Privacy Revocation" is the title of a paper introducing the notion of Publicly Auditable Privacy Revocation (PAPR). A credential scheme with this property must provide both conditional privacy of its users and public verifiability of correct behaviour of a privacy revocation authority. Two suggested constructions of a credential scheme with this property are provided in the paper, one pseudonymous and one anonymous. The former is covered in this thesis and will be explained in the following paragraphs, wherein we will refer to the scheme simply as PAPR.

3.1 Introduction

Before we delve into the specific procedures of the scheme, we will provide an intuitive overview.

In some cases, one may want users of a system to each have some pseudonym concealing their real identity, but still retain the ability to deanonymize them if they break any laws. For such a system to work, someone would have to know which pseudonym represents which real identity. If the service owner knows this, then it could be argued that each pseudonym is worthless. To solve the problem of who should host the pseudonym-to-identity-lookup table, one could use trusted third parties (TTPs). In a system with a TTP, the user and the service owner agree on a third party, which both trust, to hold the link between the pseudonym and the real identity. The trusted third party would then be trusted to only release this information upon a valid request (for example, proof of illegal activities).

Introducing a TTP might at first glance seem like a solution to the hosting issue, but instead of trusting the service owner, the user must now trust the TTP. The user must trust the TTP not to release the information without a valid reason, but also trust that they will protect the user's data from a data breach.

PAPR was designed to address the issue of needing trusted third parties (TTPs) in conditionally private cryptographic schemes.

Before PAPR, a system providing public accountability of its revocation authority had to rely on at least one TTP. This TTP would have insight into the authority's revocations to ensure that no unmotivated revocations take place. Contrary to what the name suggests, however, a TTP does not possess any inherent trust, as it is run by humans. Humans, as we all know, have self-interests and can

be persuaded. By instead ensuring trust through the use of cryptography, TTPs can be eradicated.

PAPR solves the problem of accountability of privacy revocation authorities, and does so by not giving the authority the means to carry out privacy revocations alone. Instead, the authority has to ask other users for help, as each user has distributed shares, or escrow shares, as they are sometimes called, of its revocation-essential identity to a set of users called *custodians*, rather than directly to the authority itself. These shares need to be decrypted by the custodians, and handed over to the authority, for the authority to restore the real identity.

Each user's custodian set is randomly determined by the user, in cooperation with the authority.

All of the above might seem confusing at first, but rest assured that we cover it all in greater detail below.

3.2 Entities

PAPR consists of two entities: the *issuer* and the *user*.

The issuer, I, corresponds to what has previously been referred to as the authority. This is some type of administrator responsible for the system and capable of issuing privacy revocations.

Every actor using the system is an instance of a user, U. These are the ones potentially exposed to privacy revocations. We denote a user's real identity (for example a name or a social security number) as ID. Further, we denote the user's public identity key pair as $(PrivID, PubID)$, and its credential key pair as $(PrivCred, PubCred)$, where $PrivCred = (x_e, x_s)$, and $PubCred = (y_e, y_s)$. The two parts of $PubCred$, namely y_e and y_s are the user's public encryption key, and public signing key, respectively.

3.3 The Lists

For the issuer to be able to ask for help regarding privacy revocations, the issuer needs a public data structure that can be monitored by each and every one of the users. For that reason, PAPR makes use of four lists. These are not ordinary lists, as they are required to be tamper-proof, i.e., what is on the lists should not be able to be taken off them. If they are not tamper-proof, then the accountability of the revocation authority would not hold. One way to make them *append-only* is by use of a blockchain. If I publishes the hash $hash(hash(newEntry)|previousHash)$ for each list to a blockchain at regular intervals, then every user can check the validity of these themselves by hashing the lists and comparing the results.

The lists that PAPR makes use of are:

- L_{sys} (sys. params.): Houses all of the system parameters, which are written to the list during the setup procedure. More specifically, it holds two elements, namely, The Common Reference String, CRS, and the public keys of the issuer Pk_I. The CRS contains all parameters needed for a user to set up an instance of the system on its own.

- L_u (users): In the final step of the enrollment procedure, if all has gone well, the issuer publishes a tuple of the user's *PubID* and *ID*.
- L_c (credentials): Analogously to L_u, this list contains every user's *PubCred*, if they have been issued one.
- L_{rev} (revocations): Lists all the published revocation requests. Each entry contains the following. The *PubCred* of the user whose anonymity is requested to be revoked, the escrow shares, S_e, and the set of the user's custodians' public credentials (encryption keys), D.

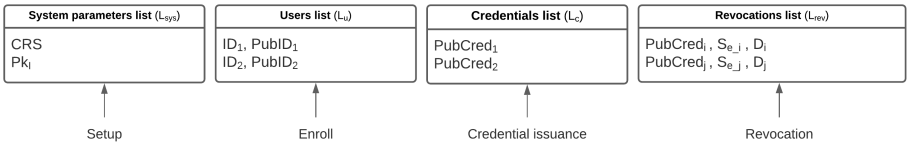

Figure 3.1: The four lists that are required in a PAPR Credential scheme.

3.4 Procedures

PAPR can be divided into five parts, *Setup*, *Enrollment*, *Credential Issuance*, *Authentication* and *Privacy Revocation*. A short description of each follows.

- *Setup*: The issuer executes this procedure to generate all the necessary system parameters. Amongst other things, the elliptic curve group is determined, which generators to use, and so on.
- *Enrollment*: A soon-to-be user of the system generates its public identity key pair, *PrivID* and *PubID*, and discloses its real identity, *ID*, along with the *PubID* to the issuer. If the user is not already enrolled, the issuer returns a token to the user. This token, $T(ID)$, cannot be linked to the user's *PubID* or *ID*. The possession of it only verifies that the holder is some enrolled user.
- *Credential Issuance*: A user that has been enrolled uses its token to get a valid pseudonymous credential.
- *Authentication*: A user can sign a message, sent from the issuer, with its private signing key (part of the *PrivCred*) when authenticating. The signature is then sent to I, which verifies it with the help of the user's public signing key that is part of *PubCred*.
- *Privacy Revocation*: If a user misbehaves, the authority can publish a privacy revocation request, to revoke the user's privacy. Such a request consists of an entry in L_{rev}. Each entry in L_{rev} is on the form $(PubCred, S_e, D)$, where S_e are the escrow shares, and D is the set of data custodians' public

encryption keys, y_e. All users that find their own y_e as an element in D are expected to answer, as they are custodians for a credential that has been requested to be revoked. Once a certain number of custodians have answered, the authority can recreate the *PubID* of the misbehaving user.

3.4.1 Setup

Setup is run to determine the parameters of the system. This includes choosing a large prime p to determine the group G and which generators, g_0, g_1 to use. Apart from that, the procedure is also responsible for generating the issuer's keys, along with a set of issuer parameters. The procedure takes two values, k and n, as input. These relate to Schoenmakers' PVSS scheme [3], which is explained below, in Section 3.4.3.

3.4.2 Enrollment

A user, U, can initiate an enrollment request and does so by generating its public identity key pair, $(PrivID, PubID)$. She then proceeds by sending the ID and $PubID$ to the issuer, I, after which U and I initiate the CMZ issuance procedure (as described in Section 2.5.1), with *PrivID* as the user's secret attribute. If I accepts the zero-knowledge proofs involved in the CMZ issuance procedure, then I goes on to construct a multiuse CMZ credential token $T(ID)$. In short, the construction of the $T(ID)$ begins when U initiates the CMZ issuance by encrypting its *PrivID* with ElGamal. This encrypted value is then modified homomorphically by I and sent back to the user. So to extract the $T(ID)$, U decrypts the value. Hence the $T(ID)$ is unknown to I. The issuer then signs *PubID* to $\sigma_I(PubID)$ and sends that to U, before finally publishing the tuple $(ID, PubID)$ on the list L_u.

3.4.3 Credential Issuance

Using $T(ID)$, which U received during the enrollment procedure, U can request a signature on a PAPR credential, i.e. a *PubCred*. This will be the user's pseudonymous credential with which U later can authenticate herself.

Credential issuance consists of four steps: *Anonymous Authentication*, *Data Distribution*, *Equal Identity Proof*, and *Credential Signing*.

Anonymous Authentication

A user uses $T(ID)$ to get authenticated as a holder of a registered *PubID*. She does this by running *Show* and *ShowVerify*, defined in the underlying CMZ scheme, which, amongst other things, proves, in zero-knowledge, her possession of *PrivID*.

Data Distribution

Once U is authenticated as an enrolled user, she then has to select a random set of users which we will subsequently refer to as her *data custodians*. A user and the issuer select the custodians cooperatively by committing to a value each, and

then using those values as input to a pseudorandom function. The output of that function is then used to determine the set of custodians D.

Having agreed upon which custodian set to proceed with, U then splits her *PubID* into n shares, where n is equal to the cardinality of the set of custodians. For this step, PAPR makes use of Schoenmakers' PVSS scheme [3], as described in Section 2.4. To split her identity, U executes the PVSS method *distribution of the shares*. This method will return the escrow shares S_e. These are the shares that have been encrypted using the custodians' keys.

Equal Identity Proof

So far, I has not been given any proof that it was in fact *PubID* that was distributed during *Data Distribution*. To remedy this, U makes use of a zero-knowledge proof scheme invented by Chaum [13], essentially proving that the secret that U proved possession of during *Anonymous Authentication*, i.e., *PrivID*, is the same value as the one that was distributed in *Data Distribution*. This is true, since it is actually $PubID = g_0^l$ that is distributed, where $PrivID = l$.

Credential Signing

Lastly, if all of the above subprocedures were executed correctly according to both sides, it is time for I to sign the user's PAPR credential. A PAPR credential consists simply of two public keys.

A user, U, generates her $PrivCred = (x_e, x_s)$, where $x_e, x_s \xleftarrow{\$} Z_p^*$ are the private encryption key and the private signing key, respectively. With her *PrivCred*, U forms $PubCred = (y_e, y_s)$ where $y_e = g_0^{x_e}, y_s = g_0^{x_s}$. The public encryption key, y_e, is used for encrypting identity shares by users that has U as one of their custodians. The private signing key, y_s, is used during authentication.

For each PAPR credential issued, I stores the corresponding sets of escrow shares and public encryption keys of its data custodians.

3.4.4 Authentication

A user of PAPR can use the *Show Credential* procedure to show that she is a valid user of the system. The entity that she authenticates to can therefore trust that the user's privacy is revocable. Let us assume that U wants to authenticate herself to I. The procedure begins by I sending a message, m, to be signed by the user's private signing key, x_s. Upon receiving the signature from U, I verifies it by using y_s. If it is valid, U is authenticated. More on digital signatures can be found in Section 2.2.

3.4.5 Revocation

The authority can request the privacy of a credential holder to be revoked. The data custodians of the user being revoked are now obliged to answer. If at least k out of n custodians answer, the authority is able to restore *PubID* from the shares, since a threshold scheme is used. During enrollment, a link is made between the

user's *PubID* and its real-world identity, *ID*. Hence, this procedure restores the real identity of the user.

If the authority misbehaves, for example, if she revokes an unreasonably large amount of users, the custodians of those users may refuse to answer.

If an authority revokes a very large number of users, she will, with increasing likelihood, already have revoked the custodians of the user that she is currently revoking, thus rendering the revocation impossible. This is an issue that can arise sooner or later, depending on the choice of system parameters. We will explore this in Section 5.2.

Chapter 4
Implementation

During our time working on this thesis, we have implemented PAPR and PAPR Money in Python. We have also implemented PVSS [3] and CMZ, [9] upon which PAPR depends. Our code has been tested on and works on Arch Linux and Ubuntu 20.04 (both natively and inside Windows Subsystem for Linux version 1 [17]).

4.1 Dependencies

Petlib [18] provides the elliptic curve cryptography operations and the big number arithmetic in general. The latest version is officially tested on Python 3.6 and 2.7 [18], although it still works on Python 3.9.2 in our experience. Petlib has OpenSSL as a dependency, so it is also an indirect dependency of our code. We initially attempted to use *Charm Crypto* [19] instead of Petlib, but it would not compile with more recent Python versions.

Pytest [20] is needed to run the tests.

Bit [21] is the library that enables PAPR Money to talk to the blockchain (see Section 4.3). A multitude of different Bitcoin libraries exist for Python, but Bit seemed to be the most popular and most recently updated.

Now is a good time to point out that, by using elliptic curve point groups, the cryptographic operations become different from what is described in the thesis. Thus, for example, even though we might write g^x in the thesis, in our code, the corresponding expression would be $x * g$, which is the equivalent elliptic curve group operation. So as to preserve group agnosticity, we keep the old notation, however. To read more on how to convert conventional finite field operations (with modular reduction) into the realm of elliptic curves, please see Appendix A.

4.2 PAPR

The implementation of PAPR is based on two entities, the *issuer* and the *user*. These entities strictly follow the logic and procedures described in the original paper on PAPR. To simplify matters, in our implementation we have no network

interface between a *user* and the *issuer*, as there would be in a real-world application. As a proof of concept, it was instead deemed sufficient to let each user hold a reference to the *issuer*, from which the user calls *issuer* code. Structuring the system this way enabled us to skip implementing a network interface, which, due to the nature of our research questions, would not have changed our results significantly.

A user class without any reference to the *issuer* is also provided. That class would be recommended for anyone who wishes to develop real-world applications on top of PAPR. Note, however, that this class has methods with names such as *req_enroll_1()* and *req_enroll_2()*. This naming convention arises from the fact that PAPR is inherently interactive. To give an idea of what this means we show an example. In this particular case we concern ourselves with the enrollment. This is initiated by a user wishing to register, which she does by calling *req_enroll_1()*. The output could then be sent over the wire to the issuer as input for its *iss_enroll()*. This, in turn, sends back its output to be used by *req_enroll_2()*.

As a matter of fact, most of the logic was tested using the user without a direct reference to the issuer, with the help of Pytest. In Pytest an issuer was set up along with a number of users. The information exchange was sent to and from the issuer using method arguments and return values. Once all procedures presented in PAPR [1] passed the tests, we moved on to our version of the user class that directly references the issuer. Hereafter the version with an issuer reference will be referred to when writing simply *user*.

4.2.1 CMZ

Alberto Sonnino's work [22] heavily influenced our implementation of CMZ. However, our code differs from his since ours only support one hidden parameter, as is all that is necessary for PAPR.

We initially set out to build the CMZ credential system that is secure under decisional Diffie-Hellman (DDH), but this idea was made obsolete when realizing that the other credential scheme presented in the paper, secure in the generic group model (GGM), was sufficient. As we had already come a long way when this dawned on us, we decided to continue working on our implementation, instead of switching to Sonnino's, which also is an implementation of the GGM-version. Moreover, by building it ourselves, we also believed that that would help us get a deeper understanding of the credential scheme. Nevertheless, we believe our implementation could be swapped for Sonnino's without too much hassle.

4.2.2 Schoenmakers' PVSS

We also implemented Schoenmakers' Publicly Verifiable Secret Sharing scheme [3]. This is the first implementation in Python that we are aware of. In our implementation we have five main methods: *distribute_secret*, *verify_encrypted_shares*, *reconstruct*, *participant_decrypt_and_prove* and *verify_decryption_proof*. The remaining methods are helper methods.

- *distribute_secret*: Generates encrypted shares, commitments and a proof that the commitments and encrypted shares represent the same secret. The shares are encrypted to the public keys that the method takes as an argument.
- *verify_encrypted_shares*: Verifies the proof generated in *distribute_secret*.
- *participant_decrypt_and_prove*: Decrypts an encrypted share given the private key and the encrypted share. It also generates a proof that the share was correctly decrypted.
- *verify_decryption_proof*: Verifies the proof from *participant_decrypt_and_prove*
- *reconstruct*: Reconstructs the shares that have been decrypted in *participant_decrypt_and_prove* to the g_0^s where s is the secret and g_0 is the generator.

Our PVSS implementation takes, as input, an elliptic curve group and two generators. It is vital that both CMZ and PVSS uses the same curve and generators as PAPR.

The answer from the decryption and reconstruction is the g_0^s where s is the secret and g_0 is the generator.

4.2.3 Bootstrapping

An obstacle to overcome regarding PAPR is the bootstrapping. In order for a user to distribute the shares of her *PubID* to n other users, the n other users must exist and have valid credentials. But what about the first n users? To solve this, we constructed a bootstrap procedure, which we explain further down.

The bootstrapping problem could be solved in different ways. One way would be for the issuer to create the first $n+1$ users herself and then add real users as usual. However, this would make it so that the issuer is in control of the first $n+1$ users and thus make the issuer in control of all custodians of the first real user, i.e., user $n+2$. Hence, enabling the issuer to revoke the first real user's privacy without having to publish that they do so.

Another option would be to use random points as the public keys. Doing so would make it impossible to revoke the privacy of the first users who only have the random points as custodians.

A third option is to batch the first $n+1$ users who want to sign-up and let them be each other's custodians. However, this option is more complex and will have to be re-done if any of the users fail their sign-up procedure.

We implemented the third option.

During normal credential issuance, when we have at least $n+1$ users already, a new user distributes the shares of its *PubID* amongst the users that are already fully registered, with valid PAPR credentials. As for our users registering during bootstrap, however, there are no users that have finished the credential issuance procedure. Therefore the custodians of the first users must be other users registering at the same time. Every user participating in the bootstrap procedure must share their *PubCred* before any user can begin their *Data Distribution* step.

Therefore the credential (*PrivCred*, *PubCred*) that are generated as part of *Credential Signing* must be created and sent to the issuer at an earlier point than usual. This means that *Credential Signing* must be split into two parts, credential generation and credential signing.

Our method creates $n+1$ users and enrolls them ($n+1$ is needed since a user cannot be its own custodian). Once enrolled, these users then create credentials and send them to the issuer. The credentials are assumed valid by all users right from the start, hence allowing the users to distribute their shares amongst each other. Once the escrow shares have been created, the users go through the same process as usual. Once all users participating in the bootstrap procedure have completed their registration, the credential issuance procedure can be carried out as usual.

Our bootstrap implementation introduces a temporary credential list, to which every user writes their unverified public encryption key, y_e. When every user has done that, we yet again loop through the users. This time, with the exception of credential generation, carrying out the normal procedure of the credential issuance.

We found no disadvantage of always letting the user generate its credential as the first step in the credential issuance procedure. The main benefit gained from running the sub-processes in this order was that the credential could be used as an identifier for the specific user throughout the four steps of the credential issuance procedure.

It could be argued that our bootstrapping procedure is volatile since if one of the credentials fails during *Credential Signing*, then everything fails. However, the research questions posed in this thesis do not address issues regarding malicious users. Thus this is not a problem we will concern ourselves with.

Another thing worth mentioning is that users' enrollment does not need to be part of a proper bootstrap procedure. We structured it this way simply for convenience.

4.3 PAPR Money

We have implemented an addition to PAPR, which we call *PAPR Money*, which uses PAPR to add revocable privacy to Bitcoin. Although still simply a proof of concept of a real-life usage scenario of PAPR, it facilitates real Bitcoin transactions within a network of PAPR Money customers. This could be used by banks wanting to provide cryptocurrency trading without having to worry about regulations concerning anonymous payments.

To carry out a transaction, the user must first get it accepted by the bank, which checks if the payee's address corresponds to one of the users in PAPR Money. If the bank accepts the transaction, a standard Bitcoin transaction is created.

For PAPR Money to work with Bitcoin, we changed the group used by PAPR to be the same elliptic curve group as the one used in Bitcoin. This allows for a Bitcoin address to be derived from the *PubCred*, thus allowing for the *PubCred* to be directly linked to the Bitcoin address used by PAPR money. Our implementation is a proof-of-concept and does only support the usage of one Bitcoin address per customer. (See Chapter 6.2.3 for a proposal on how multiple Bitcoin addresses

could be implemented.)

As in our PAPR implementation, we do not provide a network interface. This implementation is sufficient for a proof-of-concept, but in a real-world application this would have to be changed to use a messaging interface, where most of the issuer's messages would be put on and read from the lists.

4.3.1 Bank

We implemented a class, *bank*, that extends the issuer's functionality. In addition to the functionality provided by our *issuer* class, *bank* contains a method to check if a Bitcoin address is an address from a registered user, as well as methods for sending money and for getting the address of the bank.

A change was made to override the credential signing step in the issuer, to also add the Bitcoin address of the customer (user) to a list, since it's not possible to go from a Bitcoin address to a public key (the *PubCred*). This is due to the address being a hash of the public key.

4.3.2 Customer

As part of our proof-of-concept *PAPR Money* implementation, we created *customer*, which inherits from *user*.

The customer has, in addition to the methods it inherits from *user*, methods for sending money (*send*), getting its Bitcoin address (*get_ address*) and a method for checking its balance (*get_ balance*).

4.3.3 Limitations

We had to limit the scope of our PAPR-money implementation due to time constraints.

In our implementation, we create one hash-chain per list, with a hash for each new item put on the list. The intent was to enable the issuer to batch-update the lists and only publish one hash per batch, to reduce the transaction fees. We do not publish the hashes to a blockchain, which would be needed in a real-world application to have the immutability requirement met.

Some of our simulations run multiple threads. Therefore, unfortunately, we had to disable the hash-checks as that part of our implementation was not made thread-safe. However, it should be noted that the verification of the hashes works when running a single thread.

During the simulations, mock transactions were made, instead of real ones, since it would be unfeasibly costly otherwise, even on the Bitcoin testnet.

During the transactions, the payer verifies that the payee is a registered participant of the system. However, the payer does not verify if the other party had been subject to a privacy revocation.

4.4 Simulations

Our simulations were created to gather enough data to answer our three main questions. To learn more about these questions, please see Chapter 5.

To investigate our questions, we created three simulations. To address Question 1, we created *Load Simulation*. To address Question 2, we created *Revocation Failure Simulation*, which intends to determine when revocations start to fail based on how many previous revocations have been made. Lastly, to address Question 3, we created *Revocation Delay Simulation*.

4.4.1 Load Simulation

A list of parameter combinations (k, n) is supplied to a Pool object from multiprocessing, a built-in multicore Python library. The Pool starts as many processes as there are CPU cores, distributes the parameter combinations among the cores, and lets the processes pick one combination at a time. For each combination, the program runs one bootstrap procedure, while measuring the time it takes for each user. Once it has completed and all $n+1$ users have been created, the result time (the minimal entry out of all the times) is written to a file along with k and n. All of the processes are running on separate cores in parallel.

4.4.2 Revocation Failure Simulation

We start with every user having valid PAPR credentials. Then we let the issuer, I, request a revocation for a random user. Then we let all the users poll the L_{rev}. If a user finds their y_e in at least one of the custodian sets on L_{rev}, then it will help out by decrypting those escrow shares. But, on the other hand, if a user finds their *PubCred* in the list of public credentials that I wants to revoke, then they stop answering forever.

Once every user has checked L_{rev}, and responded if necessary, it is time for I yet again. This time, I attempts to restore the previously published request. If it was successfully restored, then a counter keeping track of successful revocations ticks up. Otherwise, the counter for unsuccessful revocations is incremented. At the end of the loop, we print the total number of requests, successful requests, unsuccessful requests. This loop continues until I has published requests for the revocation of every user in the system.

4.4.3 Revocation Delay Simulation

In this simulation, we use our PAPR Money classes to simulate a running system. With it, we intend to find out how long the delay of revocations are, i.e., how long it takes from publishing a revocation request until that revocation request has been successfully restored. To test this we use a system with 250 customers, and run the simulation for different k, n and user patterns. We make use of two user patterns. The first user pattern defines the login interval, measured in days, as a value taken from a Gaussian distribution with $(\mu, \sigma^2) = (2, 0.5)$. The second user pattern uses a Gaussian distribution with $(\mu, \sigma^2) = (7, 2)$.

The simulation begins by running the bootstrap procedure, adding the first $n+1$ customers. Then the $250-(n+1)$ customers are added, enrolled and issued credentials.

To our aid we have two threads, one for the bank and one for the customers. The customers all have fixed login intervals determined at the start by the Gaussian distribution. They are placed in a priority queue, sorted so that the customer who has the earliest next login time is first.

The customer thread polls the priority queue to get the customer that is supposed to log on next, and will then do one of three things. First, if the customer is not enrolled, she requests to become so. Second, if the customer is enrolled but does not have a credential, she requests a credential. Lastly, if the customer has a credential, it will first check L_{rev} and respond to any relevant revocation requests and then perform a mock transaction to another customer in the system. If the customer finds her credential in the list of revocation requests, she will refuse to do anything, effectively simulating a user logging off forever. If the user is not on L_{rev}, the thread puts the customer back into the queue, only to repeat the processes again and again.

Meanwhile, the bank thread is issuing a random revocation request once every simulated day and logs whenever this happens. Ten times a day, the bank checks if any of the published revocation requests can be restored, and if so, the request is restored, and the time is logged. The simulation stops once the bank has issued a revocation request for every customer.

4.4.4 UML

Figure 4.1 shows an UML diagram covering the essential parts of our codebase.

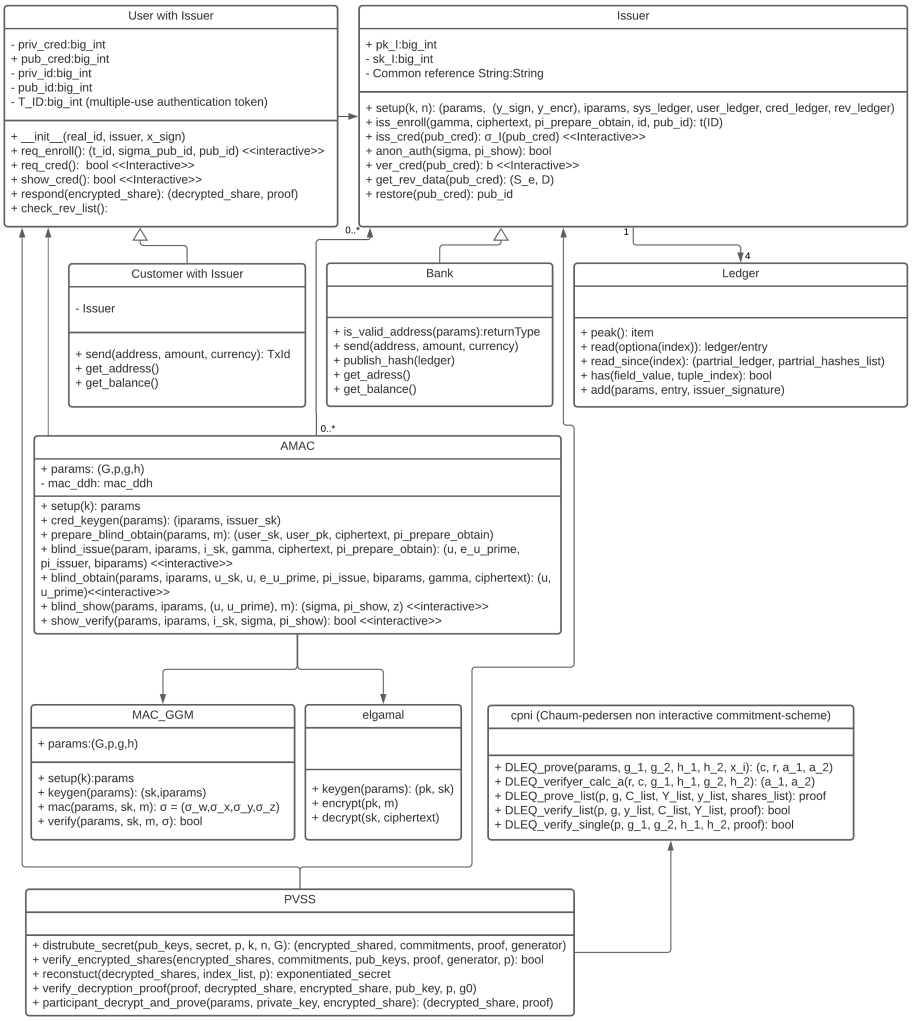

Figure 4.1: Simplified UML

Chapter 5
Simulation Results & Analysis

Once PAPR and PAPR Money were implemented and set up, we ran simulations of a PAPR Money network. What we intended to find out was the following:

- *Question 1*: How does the committee size, n, and threshold number, k, affect the computational load of the user's device during credential issuance?
- *Question 2*: At what point in time does a PAPR network begin to experience unsuccessful revocations, due to the number of previously revoked users, for different committee sizes and threshold numbers?
- *Question 3*: How do the size of the committee, the threshold number and the time between logins for users affect the time it takes for a revocation request to be fully processed all the way to a successful deanonymization?

5.1 Question 1: Computational Costs

During the credential issuance procedure, and more specifically the distribution of the encrypted shares of *PubID* the user is required to construct a polynomial of order $k-1$, and then evaluate this polynomial at n places. This is, of course, increasingly taxing as threshold number k and committee size n increase. How much slower does credential issuance become when increasing k versus increasing n? How do these two parameters relate to each other? In this section we investigate how the parameters k and n affect the time it takes to run the credential issuance subprocedure for an arbitrary user, by running the bootstrap procedure. For more information on how the bootstrap procedure works, see Section 4.2.3.

As the bootstrap procedure runs $n+1$ rounds of credential issuance, we believe it makes for a good candidate for extracting our needed data points. Albeit we do not calculate the average of the data points, but instead use the time of the user that managed to execute the procedure the fastest, amongst the $n+1$. This is done to avoid outliers, arising from swaying CPU load and such, from affecting the result.

We decided on measuring all $n \in \{10, 20, ..., 200\}$. For each n we test all $k \in \{5x : x \in \mathbb{Z}^+, 5x < n\}$. The result can be seen in Figure 5.1.

The duration of a credential issuance run clearly increases as k grows. That behaviour was expected as the data distribution subprocedure requires the user to

construct a polynomial of degree $k-1$. Clearly, that would be more computationally expensive if k increases.

It is clear that the larger n becomes, the steeper the slope will be when letting n be fixed while varying k. This is obvious as a large n requires more evaluations on the polynomial generated in *data distribution*. With a big n, there are also more data custodians to select and to encrypt shares for.

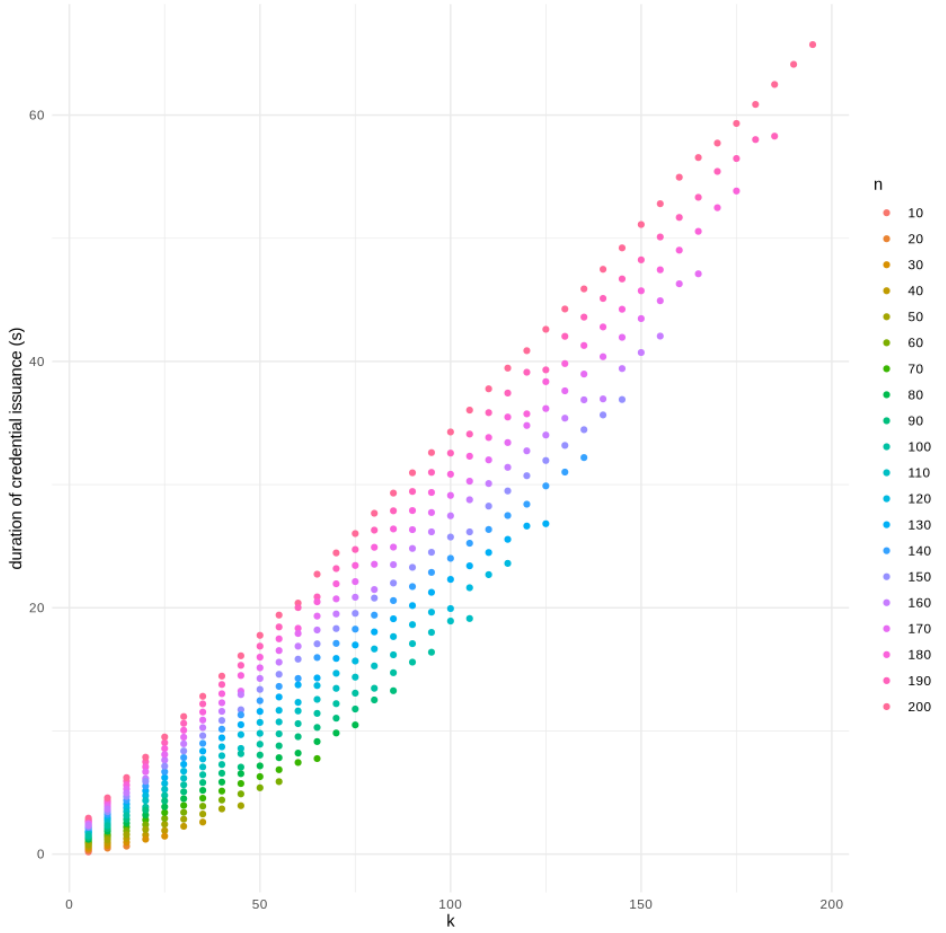

Figure 5.1: Duration of the credential issuance procedure for different combinations of k and n.

Please note that our results measure both the user processing time and the issuer processing time combined. The whole credential issuance was run sequentially, where user and issuer take turns processing. The main computationally heavy task of the issuer is her selection of custodians. In the real world, this could be done in tandem with the user selecting custodians. This would reduce the time slightly. Hence, Figure 5.1 could be regarded as an upper bound. Speed is also implementation-dependent. Given optimization or hardware support, our code

would be faster. How much faster our code would be is, of course, unclear.

5.2 Question 2: Critical Number of Revocations

This inquiry is both tricky to answer and to imagine. To explain the problem, we must consult probability theory and, more specifically, the hypergeometric distribution. The hypergeometric distribution is a discrete probability distribution that simulates the following particular case: Imagine a hat with N number of marbles in it, out of which K are white and $N - K$ black. If n marbles were to be removed from the hat, the probability that there were exactly k white marbles among the removed marbles is given by the hypergeometric distribution. The *cumulative* hypergeometric distribution function instead describes the probability of getting k or less white marbles, for a specific number of removals. This distribution can easily be tweaked to instead show the probability that more than k white marbles were removed.

Suppose we instead picture a PAPR Credentials system, with a (k, n)-threshold-scheme, where the white marbles correspond to the n custodians of an arbitrary user, and the black marbles symbolise the rest of the users, and each marble removal corresponds to a revocation. In that case, the cumulative hypergeometric distribution tells us how big the chances are that among those earlier revocations, there were more than $n - k$ custodians for an arbitrary user.

If more than $n - k$ of a user's custodians have already been revoked, then that user is suddenly irrevocable, as there are less than k custodians left. This is clearly a problem. Take PAPR-money, for instance. If a user cannot be revoked, the anti-money laundering directives [8, 16] might deem the digital currency software illegal. To read more on the anti-money laundering directives, see Section 2.8.

So how do we select the parameters for the threshold scheme such that the risk of unsuccessful revocations is low? Figure 5.2 shows some suggestions. If we step back to the original question, we can see that the answer is completely dependent on what values we give to k and n. From Figure 5.2 we can derive that the probability of revocation failure increases faster for threshold schemes with bigger n. We also see that the quotient of k/n determines where, on the x-axis, the probability of unsuccessful revocations starts to rise. The closer the quotient is to 1—and the further it is from 0—the earlier the probability begins to ascend.

Increasing n, while letting k be fixed, is advantageous as it postpones the unsuccessful revocations in regard to the number of earlier revocations, but worse since a selection of users colluding with the issuer have a higher chance of each getting a share. We will, however, not concern ourselves with problems connected to collusion or malicious users.

What happens when k is set to a value close to n? Revocation delay will increase as more users are needed for reconstruction, the computational load will be higher, and, of course, the number of unsuccessful revocations will go up.

Does the number of users affect the probabilities in Figure 5.2? Taken to the extreme, it does. As an example of this, please see Figure 5.3, with an emphasis on the curve for $(k, n) = (50, 100)$. In such a system, every user is a data custodian of every other user.

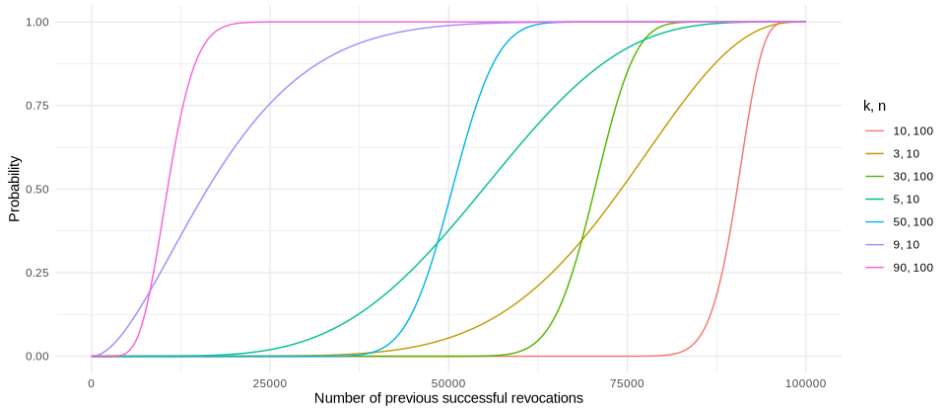

Figure 5.2: Probability that a revocation will not be possible based on the number of previous user privacy revocations, for different k and n, for $100,000$ users.

If you are thinking that 101 is the minimum number of users for a PAPR Credential system where $n = 100$, then you are correct. This plot does not consider that a user cannot be its own custodian, but this is beside the point. The point is that in such a system, once 51 users have been revoked, the chance of being able to revoke the next user's privacy is zero.

If we increase the number of users to $1,000$, we obtain the probabilities of Figure 5.5. Finally, Figure 5.6 shows the probabilities of a system with $10,000$ users. As can be seen, when comparatively looking at the above-mentioned figures, the differences between the graphs become smaller and smaller as the number of users increases.

Hence we can deduce, and conclude, that, as the number of users tends toward infinity, our probabilities asymptotically reach a steady state.

To verify the behaviour displayed in the above plots, we ran a simulation on the system with $1,000$ users (Figure 5.5). To read more about how the simulation works, see Section 4.4.2. Figure 5.4 shows the results of the simulation. As can be seen, the hypergeometric distribution, when used correctly, can be a great aid in predicting when revocations begin to fail in a PAPR Credentials system with a (k, n)-threshold scheme.

5.3 Question 3: Revocation Delay

To answer how the size of the committee, the threshold number and the time between logins for users affect how long it takes for a revocation request to be fully processed all the way to a successful deanonymization, we must first establish some assumptions, namely,

- how often a user uses the service and
- how big the committee and threshold number should be.

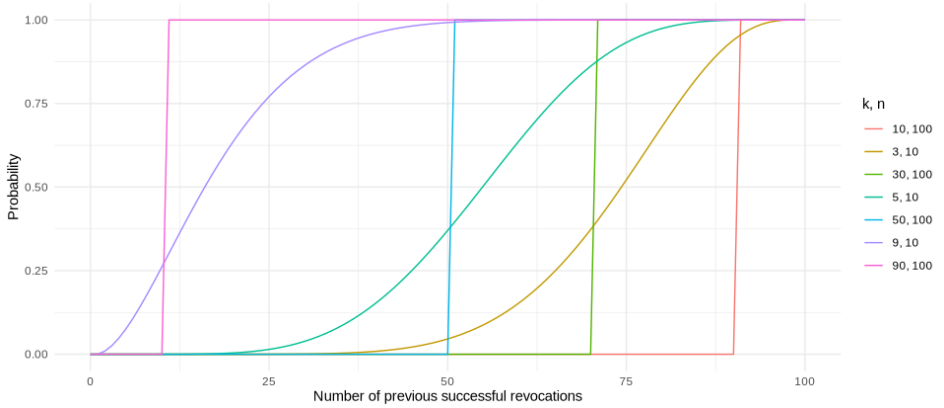

Figure 5.3: Probability that a revocation will not succeed based on the number of previous user privacy revocations, for different k and n, for 100 users.

We decided to investigate two user patterns. For each user pattern, we looked at two k's, and for each of the k's we examined four different values of n. On top of that, we have decided to look at $(k, n) = (5, 10)$ for the selected user patterns. In total 18 combinations.

Each user is given a login frequency (measured in days) from normal distribution values, which determines what we call the user pattern. To clarify: each user will be given their own login frequency at the start, and each user's login frequency will be fixed throughout the simulation.

The first user pattern we decided to work with is defined by $\mathcal{N}(2,\ 0.5)$. In this distribution, almost all login frequencies fall in between zero and four days. The few users who end up with negative frequencies are taken care of by using the absolute value of the frequency.

The other user pattern we went for was $\mathcal{N}(7{,}2)$. With this distribution, most of the values end up in the range of 0 to 14.

The reason for choosing these user patterns, in particular, was simply because they allow us to observe what happens when the variance of login times changes. One can quite simply deduce that, for two user patterns that only differ in mean-values, the delay would change while the variance of the delay would remain the same. The Gaussian distributions used are visualised in Figure 5.7.

We chose to inspect $k = 5$ and $k = 20$. For n, we looked at 50, 100, 150 and 200. As mentioned earlier, we also added $n = 10$ for $k = 5$. It was specially added to allow us to see what happens when k and n both are small, while k is comparatively large.

Using these values, we were able to produce Figure 5.8 and Figure 5.9, the only difference between the plots being that the latter has an added polynomial regression fitting [23].

As can be seen in Figure 5.9, the regression fitting of each parameter combination, where k is small in relation to n, is almost straight, until a certain point, when some revocations have been made. After that point, the average revocation

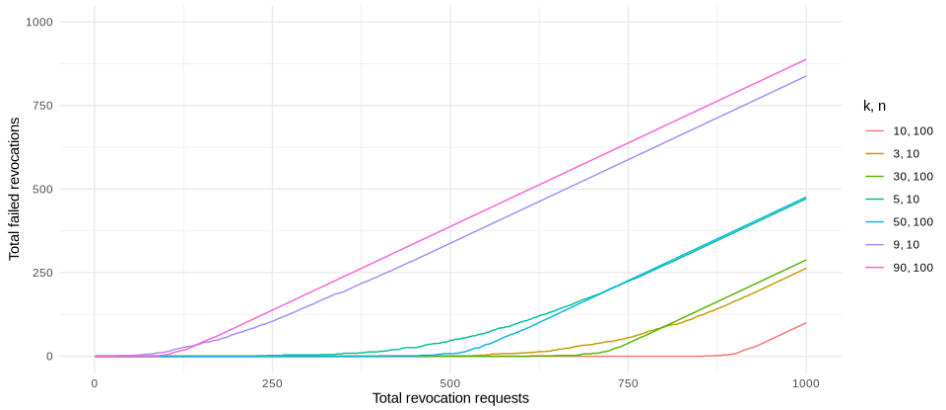

Figure 5.4: Total number of failed revocations in relation to number of revocation requests, for different k and n, for $1,000$ users.

delay starts growing exponentially until no more revocations are possible. Combinations where k is big in relation to n, e.g., $(k, n = 20, 50)$, exhibit exponential growth already from the start.

5.3.1 Fixed k

In Figures 5.8 and 5.9 we can see that by increasing n relative to k, the expected duration from the time of requesting a revocation to the time of the revocation succeeding is kept short for a longer period of time. This could be explained by the following. The likelihood of at least k custodians answering quickly becomes higher the more custodians there are. The larger the committee size is, in comparison to the number of custodians that are needed for a successful revocation, the quicker the revocation can complete. This can be compared to what we previously have seen in Figure 5.2, where we can see that an increase in the size of the custodian set—in comparison to the threshold number—affects the likelihood of successful revocations in a similar way.

5.3.2 Fixed n

What happens when we increase k while having a fixed n? The change can be observed by looking at the difference between the rightmost plots and the leftmost plots of Figure 5.9. Clearly, this increases delay no matter how many previous revocations have been made. The revocation delay also seems to begin growing exponentially from the start, as mentioned above.

5.3.3 Variance

We can see that different values of parameters, apart from changing the average delay of revocations, give rise to more or less variance of the data points. For example, in the case of $(k, n) = (5, 10)$, we can see that the variance of the time

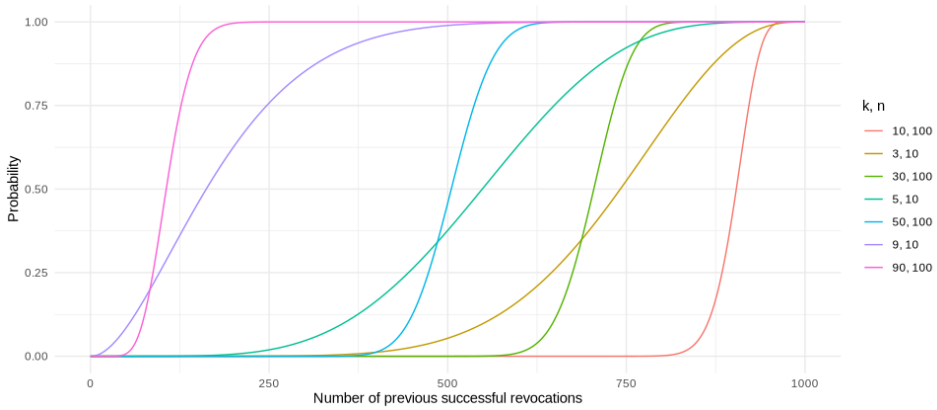

Figure 5.5: Probability that a revocation will not succeed based on the number of previous user privacy revocations, for different k and n, for $1,000$ users.

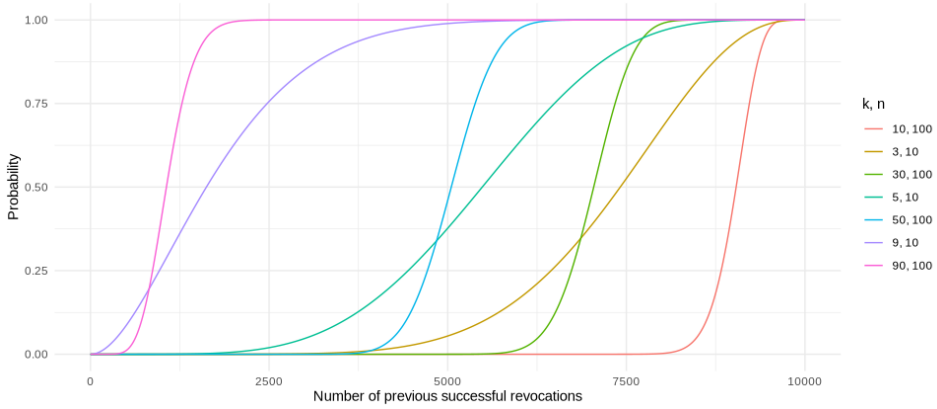

Figure 5.6: Probability that a revocation will not succeed based on the number of previous user privacy revocations, for different k and n, for $10,000$ users.

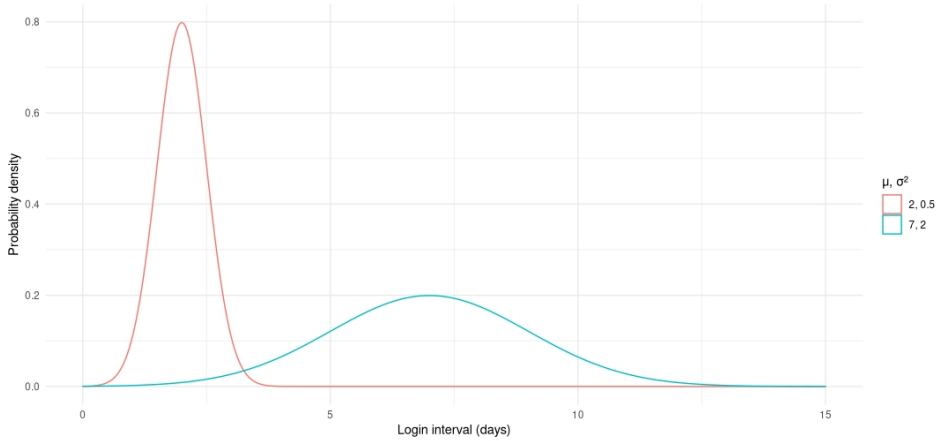

Figure 5.7: The Gaussian distributions used for generating login intervals

until a revocation request is sufficiently answered is much larger than in any other plot. Since a large fraction of the custodians is needed, the likelihood that at least 5 out of 10 of the custodians are fast is much lower than the likelihood that, for $(k, n) = (5, 200)$, 5 out of 200 are fast. For the former, the 5^{th} slowest user determines the delay of the revocation request, while in the latter, the 195^{th} slowest sets the bar.

In the case of $(k, n) = (5, 10)$, we quite early have a noticeable amount of users that will never be able to be revoked (see Question 2 and Figure 5.4). Nevertheless, any user that the issuer has issued a privacy revocation request against will log out forever as soon as it finds itself on L_{rev}. That could play a part in why the average time to revocation is seemingly more dependent on outliers, for $(k, n) = (5, 10)$ than for the other parameter combinations.

5.3.4 Quotient of k and n

We happen to have two curves for each user pattern, where the quotient of k and n is the same. These are $(k, n) = (5, 50)$ and $(k, n) = (20, 200)$. The latter seems to have a lower variance. Why that is is reasoned about above, in Section 5.3.3.

The slope of $(k, n) = (20, 200)$ seems slightly more steep at the end than that of $(k, n) = (5, 50)$; however, the exponential increase in delay times begins later in regard to the number of previous revocations.

Figure 5.8: Delay from revocation request to successful revocation, for different k, n, μ and σ^2

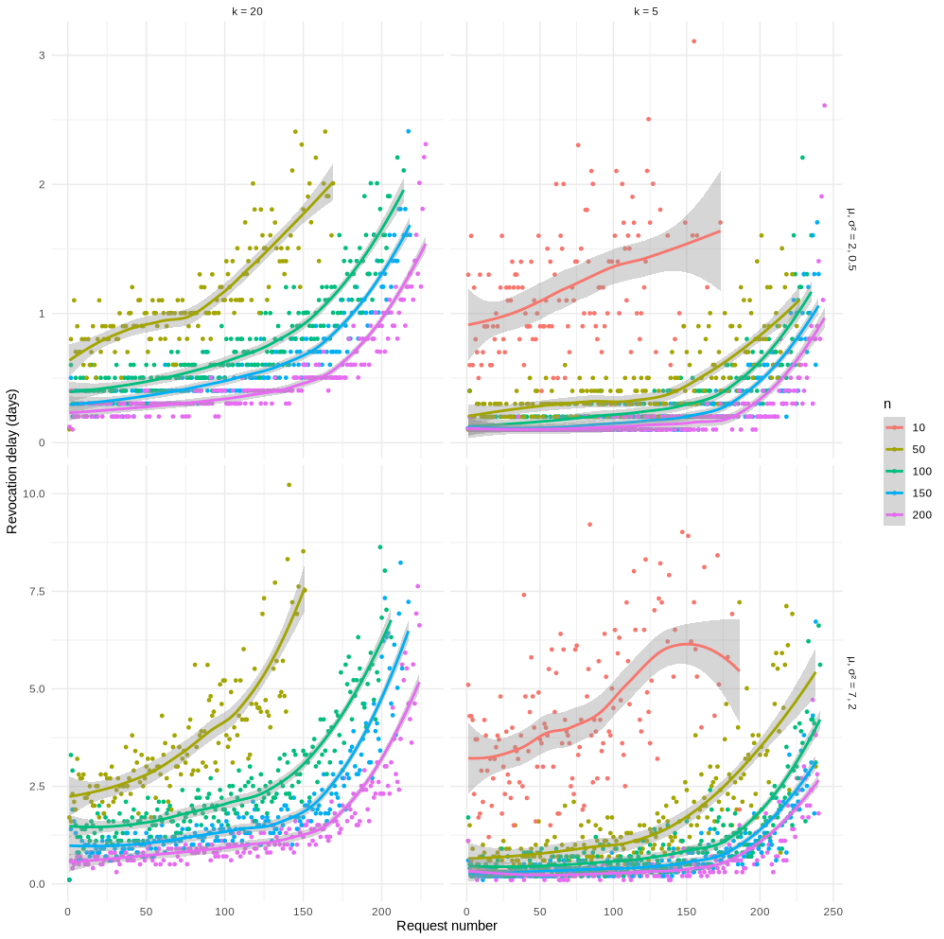

Figure 5.9: Delay from revocation request to successful revocation, for different k, n, μ and σ^2, with a local polynomial regression fitting

Chapter 6
Conclusions & Future Work

6.1 Conclusion

In this thesis, we have presented the first-ever implementation of a PAPR credential scheme. The implementation is available on Github [24]. As part of the implementation, we have built Schoenmakers' PVSS in Python. Our implementation of Schoenmakers' PVSS is the first implementation that we are aware of in Python.

A proof-of-concept real-world application was created using our PAPR credentials scheme, called PAPR Money, capable of transferring bitcoins inside a network of PAPR Credentials users while still maintaining conditional privacy. This could prove to be a big step forward in the field of anonymous transactions.

We presented three research questions concerning the properties of PAPR credential schemes, and we answered them. With the help of our data, we believe that authors of potential future implementations of PAPR credential schemes will construct their schemes with a better understanding of how the threshold number and committee size affect their implementations.

6.1.1 Analysis Summary

To keep the variance of delays low while at the same time keeping revocation delay short, we suggest a big n in comparison to k. One can favourably set k as low as possible.

By lowering k, the revocation delay becomes less affected by users with more sporadic login intervals.

By increasing the committee size, a larger subset of users can help out with revocations. Such an increase of n delays the point in time when revocations start to fail, due to the number of previous revocations, while at the same time reducing the delay of revocations.

6.2 Future Work

In the following section, we will touch on the key missing features of our implementation. We will begin, however, by mentioning an aspect which we have not taken into account before.

6.2.1 Issuer Colluding with Users

Users colluding with the issuer is a real-world problem. Given a threshold scheme with a k much smaller than n, say $(k, n) = (5, 1,000)$, then, if the issuer bribes 0.5% (k/n) of the users, she has a good chance of secretly revoking users, since the issuer could then contact the users necessary for a revocation under the counter. The above example only holds if there are exactly $1,000$ users in the system. As the number of users increases, knowing who to bribe becomes increasingly difficult for the issuer.

6.2.2 Our Implementation

During our time working on the implementation of PAPR Credentials, some corners were cut. These shortcuts did not affect the result of our simulations but should be addressed before anyone decides to use our software in a real-world application. These shortcuts are addressed below.

Proper Randomly Selected Custodians

During data distribution, when the issuer and user cooperatively select which users should become data custodians, we use a pseudorandom number generator of our own design. Any of the two parties could test a huge set of input values, and potentially find input values that somehow affect the selection of data custodians to their advantage. A selection process where this cannot occur would be preferable. One way of achieving this would be to use a multiparty randomness generation protocol like [25].

Anonymous Custodians

When distributing shares and deciding on custodians, the selected custodians must be users that have not had their privacy revoked. This is not a problem for us since, in all of our simulations, we issue credentials to all of our users before any revocations take place.

Binding of $T(ID)$

To allow for trans-credential pseudonymity, the issuer is required to log $T(ID)$ and bind it to $PubCred_i$, so that if a user requests a new $PubCred_j$ this can be linked to the old $PubCred_i$.

Refusal of Credential Usage

Two conditions why the issuer should refuse to authenticate a user with a certain credential exist.

1. If the issuer has issued a privacy revocation on a user.
2. If more than $n-k$ of a user's data custodians has had their privacy revoked.

The former condition requires a simple fix in the code. The latter condition is trickier, as the user has not broken any laws. The latter could be solved by creating a special procedure that runs specifically when $n - k$ data custodians have been revoked of a user attempting to authenticate herself. In such a procedure, the issuer could force the user to generate a new credential, link the two credentials and then allow the user to transfer all potential funds connected to the old credential to the new credential.

6.2.3 Multiple Bitcoin Addresses per User

A way to allow for multiple Bitcoin addresses per user could be by reusing $T(ID)$ to generate multiple credentials. If the issuer keeps track of the $T(ID)$ used to create the different credentials, pseudonymity still holds, and all transactions made by a user are pseudonymously linkable from the view of the authority. By logging $T(ID)$ this way, we would allow for users to create new addresses whenever they want to receive funds, in accordance with Bitcoin best practices, such that the user can determine who paid them. The credential issuance procedure would need to run once for every new Bitcoin address; thus, the time it takes for credential issuance would no longer only be a consideration during sign-up, but for *every* payment. Therefore, significantly lower k and n would likely have to be used in order to decrease the times shown in Figure 5.1. Since every *PubCred* could be used as a custodian, the customer would need to keep all old keys and need to be able to respond to revocation requests on all of her credentials. A system where every user holds several credentials would also introduce a problem where the same user (with different *PubCred*s) could be custodian multiple times to one user.

References

[1] J. Brorsson and P. Stankovski Wagner, Private communication on manuscript with title "PAPR Credentials - Conditional Privacy with Publicly Accountable Privacy Revocation", 2021.

[2] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," p. 9, 2009.

[3] B. Schoenmakers, "A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting," in *Advances in Cryptology — CRYPTO' 99* (M. Wiener, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 148–164, Springer, 1999.

[4] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "BLAC: Revoking Repeatedly Misbehaving Anonymous Users without Relying on TTPs," *ACM Transactions on Information and System Security*, vol. 13, no. 4, p. 33.

[5] J. Camenisch and A. Lysyanskaya, "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation," Tech. Rep. 019, 2001.

[6] D. Chaum, "Blind Signatures for Untraceable Payments," in *Advances in Cryptology* (D. Chaum, R. L. Rivest, and A. T. Sherman, eds.), (Boston, MA), pp. 199–203, Springer US, 1983.

[7] S. Noether, "Ring Signature Confidential Transactions for Monero," Tech. Rep. 1098, 2015.

[8] "Directive (EU) 2015/849 of the European Parliament and of the Council of 20 May 2015 on the prevention of the use of the financial system for the purposes of money laundering or terrorist financing, amending Regulation (EU) No 648/2012 of the European Parliament and of the Council, and repealing Directive 2005/60/EC of the European Parliament and of the Council and Commission Directive 2006/70/EC (Text with EEA relevance)," June 2015. Code Number: 141.

[9] M. Chase, S. Meiklejohn, and G. M. Zaverucha, "Algebraic MACs and Keyed-Verification Anonymous Credentials," Tech. Rep. 516, 2013.

[10] "torusresearch/pvss." `https://github.com/torusresearch/pvss`, accessed 2021-02-14.

[11] N. P. Smart, *Cryptography Made Simple*. Information Security and Cryptography, Springer International Publishing, 2016.

[12] C. P. Schnorr, "Efficient Identification and Signatures for Smart Cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings* (G. Brassard, ed.), Lecture Notes in Computer Science, (New York, NY), pp. 239–252, Springer, 1990.

[13] D. Chaum, J.-H. Evertse, and J. van de Graaf, "An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations," in *Advances in Cryptology — EUROCRYPT' 87* (D. Chaum and W. L. Price, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 127–141, Springer, 1988.

[14] D. Chaum and T. P. Pedersen, "Transferred Cash Grows in Size," in *Advances in Cryptology — EUROCRYPT' 92* (R. A. Rueppel, ed.), (Berlin, Heidelberg), pp. 390–407, Springer Berlin Heidelberg, 1993.

[15] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact E-Cash," in *Advances in Cryptology – EUROCRYPT 2005* (R. Cramer, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 302–321, Springer, 2005.

[16] "Directive (EU) 2019/1153 of the European Parliament and of the Council of 20 June 2019 laying down rules facilitating the use of financial and other information for the prevention, detection, investigation or prosecution of certain criminal offences, and repealing Council Decision 2000/642/JHA," July 2019. Code Number: 186.

[17] C. Loewen et al., "What is Windows Subsystem for Linux." `https://docs.microsoft.com/en-us/windows/wsl/about`, accessed 2021-06-03.

[18] G. Danezis, "petlib 0.0.45 documentation," 2014. `https://petlib.readthedocs.io/en/latest/`, accessed 2021-03-04.

[19] J. A. Akinyele, M. D. Green, and A. D. Rubin, "Charm: A framework for Rapidly Prototyping Cryptosystems," Tech. Rep. 617, 2011.

[20] "pytest: helps you write better programs — pytest documentation." `https://docs.pytest.org/en/6.2.x/`, accessed 2021-05-24.

[21] O. Lev, "Bit: Bitcoin made easy. — Bit 0.7.2 documentation." `https://ofek.dev/bit/`, accessed 2021-04-29.

[22] A. Sonnino, "asonnino/amac." `https://github.com/asonnino/amac`, accessed 2021-03-23.

[23] "loess: Local Polynomial Regression Fitting." `https://rdrr.io/r/stats/loess.html`, accessed 2021-05-25.

[24] W. Nilsson and P. Kron, "wlmr/papr." `https://github.com/wlmr/papr`, accessed 2021-05-31.

[25] I. Cascudo and B. David, "ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing," 2020.

[26] D. McGrew, K. Igoe, and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms," RFC 6090, RFC Editor, Feb. 2011.

Appendix A

Elliptic Curve Operations

Our implementation of PAPR and PAPR Money is constructed using elliptic curve point groups. Elliptic curve operations work somewhat differently than operations in the more classical finite fields constructed using modular reduction. In a classical finite field, only a few of the numbers qualify as generators. In elliptic curve cryptography, we make use of points, i.e., coordinates, on a curve. All points on an elliptic curve are generators.

Modular exponentiation, in a classical finite field, for example, corresponds to multiplication of a number and a point in a cryptographic scheme constructed using elliptic curves.

We list a few examples of how to convert the conventional multiplicative notation—as used in fields constructed using modular reduction—into the realm of elliptic curve groups.

1. The expression x^y, written in multiplicative notation, is equivalent to the elliptic curve operation $y \cdot x$, where x is a point, and y is a number.
2. The expression $x^{a/b}$, written in multiplicative notation, is equivalent to the elliptic curve operation $(a \cdot b^{-1}) \cdot x$, where x is a point on the curve, and a and b are numbers.
3. The expression $x \cdot y$, written in multiplicative notation, is equivalent to the elliptic curve operation $x + y$, where x and y are both points.

More examples can be found in RFC 6090 [26].