Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Jonatan Østgaard
Sondre Rishøi

Unbiased Distributed Key Generation

Master's thesis in Cyber Security and Data Communication
Supervisor: Tjerand Silde
Co-supervisor: Audhild Høgåsen
June 2024

NTNU
Norwegian University of
Science and Technology

Jonatan Østgaard
Sondre Rishøi

Unbiased Distributed Key Generation

Master's thesis in Cyber Security and Data Communication
Supervisor: Tjerand Silde
Co-supervisor: Audhild Høgåsen
June 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title: Unbiased Distributed Key Generation

Students: Jonatan Østgaard and Sondre Rishøi

Problem description:

Switzerland, a pioneer in providing electronic voting, has long provided its citizens with the convenience of voting on national matters using their phones or computers. This innovative approach, replacing the traditional voting booth, is made possible by the Swiss Post electronic voting system. This system, a complex network of cryptographic subprotocols, safeguards voter privacy and election integrity. However, there are areas where the system can be enhanced. Swiss Post is developing a new iteration of its voting system, aiming to bolster its security properties.

In 2020, a collaborative effort between the Swiss Federal Chancellery, academic scholars, and industry experts led to the creation of a comprehensive set of measures to enhance the system. One of these measures, A.5, calls for a reduction in trust assumptions related to the generation and printing of cryptographic parameters and other confidential values. Currently, these values are generated by the setup component and are printed by the printing component during the election's configuration phase. To ensure voter privacy and election integrity, the system relies on the assumption of these components' honesty. For measure A.5, the aim is to weaken these trust assumptions.

The primary objective of this master's project is to conduct a comprehensive analysis of the configuration phase in the Swiss Post electronic voting system. Based on this analysis, the project will propose redesigns for specific components to align with the requirements of measure A.5, partially or in its entirety.

Approved on: 2024-03-06

Main supervisor: Tjerand Silde (NTNU)

Co-supervisor: Audhild Høgåsen (Swiss Post)

Abstract

The Swiss Post electronic voting system is state-of-the-art, allowing citizens to cast their votes digitally from anywhere. However, it is critical to continually enhance the security protocols in response to continuously evolving threats. Swiss Post has proposed measurements to create a more secure system, and we will look closer at measure A.5, which aims to reduce the security assumptions in the existing protocol.

In this study we examine the existing system and propose a new methodology for generating public and secret key pairs that align with the requirements of measure A.5. We present three different approaches: two of the approaches are from existing literature, and rely on a majority of honest participants, while the third is a brand new approach designed to support a scenario where the majority of participants may be dishonest. Finally, we will discuss the trade-offs associated with each approach.

Sammendrag

Det elektroniske stemmesystemet fra Swiss Post er toppmoderne og gjør det mulig for innbyggerne å avgi sine stemmer digitalt fra hvor som helst. Samtidig er det kritisk å kontinuerlig forbedre sikkerhetsprotokollene som respons på stadig utviklende trusler. Swiss Post har foreslått tiltak for å skape et mer sikkert system, og vi vil se nærmere på tiltak A.5, som har som mål å redusere sikkerhetsforutsetningene i den eksisterende protokollen.

I denne studien undersøker vi det eksisterende systemet og foreslår en ny metode for å generere offentlige og hemmelige nøkkelpar som samsvarer med kravene i tiltak A.5. Vi presenterer tre forskjellige tilnærminger: to av tilnærmingene er fra eksisterende litteratur og avhenger av et flertall av ærlige deltakere, mens den tredje er en helt ny tilnærming utformet for å støtte et scenario der flertallet av deltakerne kan være uærlige. Til slutt vil vi diskutere avveiningene knyttet til hver tilnærming.

Preface

The work presented in this was written as a Master's Thesis at Norwegian University of Science and Technology (NTNU) in the Department of Information Security and Communication Technology(IIK).

We extend our sincere gratitude to our supervisors, Tjerand Silde and Audhild Høgåsen for their invaluable guidance, insight and patience throughout the writing process. We would also like to thank Swiss Post for the engaging project and the opportunity to travel to Switzerland to give us good insight.

Contents

List of Figures

List of Tables

List of Symbols

BCK	Ballot Casting Key.
CC	Choice Return Code.
CMtable	Return Code Mapping Table.
SVK	Start Voting Key.
VCC	Vote Cast Return Code.
VCks	Verification Card Keystore.

List of Acronyms

A Auditor.

AB-VRF Attribute-Based Verifiable Random Function.

CCR Return Code Control Component.

CDH Computational Diffie-Hellman Problem.

CHF Cryptographic Hash Function.

DDH Decisional Diffie-Hellman Problem.

DKG Distributed Key Generation.

DL Discrete Logarithm Problem.

DLEQ Discrete Logarithm Equality Proof.

EB Electoral Board.

FVSS Feldman's Verifiable Secret Sharing.

l-SDHE l-Strong DH Exponent Problem.

PC Printing Component.

PoK Proof of Knowledge.

PRF Pseudorandom Function.

PVSS Pedersen Verifiable Secret Sharing.

ROM Random Oracle Model.

SC Setup Component.

SS Secret Sharing.

SSS Shamir's Secret Sharing.

T Tally Component.

TA Techincal Aid.

V Voter.

VC Voting Client.

VRF Verifiable Random Function.

VS Voting Server.

VSS Verifiable Secret Sharing.

ZKP Zero Knowledge Proof.

Chapter 1
Introduction

In today's digital landscape, it is crucial for governments to keep pace with the evolution of digital society. Adapting the voting process to allow citizens to cast their ballots from home has the potential to be more convenient, cost-effective, efficient, and increase voter participation[TKO+16]. Swiss Post has been a pioneer in electronic voting and continues to work on enhancing the security of its e-voting system. They have collaborated with experts to identify measures to achieve this. This paper will focus on Measure A.5, which aims to "Weaken trust assumptions in the printing process and in the software that generates cryptographic parameters"[Swi20].

1.1 UN-Sustainability Goals

The United Nations has established 17 Sustainable Development Goals (SDGs) and 169 subgoals to address global challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice [Nat15]. This paper will specifically relate its findings towards subgoal 16.7: *Ensure responsive, inclusive, participatory, and representative decision-making at all levels.* Electronic voting can significantly enhance citizen participation in the democratic process by providing voting options that are more accessible to a larger demographic.

1.2 Outline

To address the problem description and Measure A.5, we will first present the Swiss Post E-Voting system in 2. This chapter introduces the necessary knowledge required to understand the existing Swiss Post electronic voting system, including its configuration phase, e-voting properties, and threat models. Following, 3 will introduce the cryptographic building blocks needed for this thesis. Distributed Key Generation is the chosen approach for addressing the problem description. 4 will introduce Distributed Key Generation, explain its security properties and present three different constructions. Concluding this chapter, a cryptographic construction is

proposed to address the problem description. A discussion of the proposed approach follows in 5. The trade-offs of the proposed redesigns will be highlighted, and lastly concluding remarks will be presented.

Chapter 2
Swiss Post E-Voting System

In this chapter we look at the relevant parts of the current Swiss Post e-voting system, in particular focusing on the configuration phase of the current system protocol, e-voting properties the system aims to satisfy, and the threat models for these properties. Finally we make some observations about the system.

2.1 System E-Voting Properties

The Swiss Post e-voting system computational proof [Swi24b] describes the e-voting properties and security goals the system aims to sarisfy. In this section we describe the three main e-voting properties: individual verifiability, universal verifiability, and vote secrecy, based on [Smy18].

2.1.1 Individual Verifiability

A voting system satisfies individual verifiability if it allows a voter to check whether their ballot is collected. They capture this property through a game between an adversary and a challenger. The game starts with the adversary providing the challenger with the inputs required to create a valid ballot, which is then returned to the adversary. The adversary now wins, if they manage to provide a second set of necessary inputs to construct another valid ballot that is indistinguishable from the first ballot. In such a scenario it is impossible for the voter to convince themselves that the ballot stored by the system reflects their intentions, meaning individual verifiability is broken.

A system with individual verifiability satisfies the following two properties:

- **Recorded-as-intended**: The system allows a voter to verify that the system has recorded a ballot marked as they intended.

– **Recorded-as-confirmed**: The system allows a voter to verify that the system has recorded the received ballot as confirmed.

In particular, *recorded-as-intended* is achieved since the voter can check whether the stored ballot matches the ballot they expect for the voting options they chose, and *recorded-as-confirmed* is achieved since the voter can check if the system has stored their expected ballot at all. Such a system allows the voter to verify that no vote tampering or vote rejection has occurred.

2.1.2 Universal Verifiability

A voting system satisfies universal verifiability if it allows anyone to check whether the election outcome corresponds to the votes expressed in the collected ballots. To achieve this, the system needs to produce an election proof in addition to the election outcome. Once the outcome is published, auditors may use the election proof along with recorded ballots, to decide whether to accept or reject the election outcome.

This property can be defined through a game between an adversary and a challenger. Here the goal of the adversary is to falsify the election proof such that either an incorrect outcome is accepted, or a correct outcome is rejected. As such, if a voting system satisfies universal verifiability, a correct outcome will always be accepted, while an incorrect outcome will always be rejected.

Another name for this property is *tallied-as-recorded*. As such, a system that satisfies both individual and universal verifiability can be described as completely verifiable, allowing voters to verify that their votes are recorded as intended, and anyone to verify that recorded votes are tallied correctly.

2.1.3 Vote Secrecy

A voting system satisfies vote secrecy if no information about a voter's vote is revealed beyond what is leaked by the tally at the end of the election. We exclude the final election tally from this definition because there are legitimate scenarios where the election outcome could reveal the contents of a voter's ballot.

This property can be captured by an indistinguishability game between an adversary and a challenger. Firstly, the challenger samples a bit $B \leftarrow\!\!\$\, 0, 1$. Afterwards, the adversary may send any number of sets of votes (v_0, v_1) to the challenger, who responds with ballot b corresponding to v_B. When the adversary is ready, they send a set of ballots to the challenger for tallying, which consists of ballots both made by the challenger and the adversary. When the challenger is done tallying, it returns the result to the adversary who attempts to guess the value of bit B. The voting system

satisfies vote secrecy if for all efficient adversaries, their probability of guessing bit B is at most $\frac{1}{2} + \mathsf{negl}(\lambda)$.

2.2 System Description

The Swiss Post e-voting system is a two-round return code based e-voting system, with an architecture similar to what is shown in Figure 2.1. When an election is held, each Voter (V) first receives a voting card by postal mail, containing a SVK, a unique CC per voting option in the election, a BCK, and a VCC. Voting occurs in two rounds. In the first round V uses their Voting Client (VC) to construct a ballot with their SVK, which is sent to the ballot box of the system. In return, they receive a CC per voting option marked on their ballot, which they can compare against those written on their voting card to ensure their ballot has been *recorded-as-intended*. In the second round V uses their VC to construct a code with their BCK, which is also sent to the ballot box. In return they receive a VCC, which they compare against the one written on their voting card, to ensure their ballot has been *recorded-as-confirmed*.

Once the deadline for voting has passed, the system begins tallying the ballots it has received. They are stored in the ballot box, which in actuality is distributed across four components called the Return Code Control Components (CCRs). Each of these components partially decrypt and shuffle their ballots and send the results, along with proofs, to the Tally Component (T). T performs a final round of decryption and shuffling, before publishing election outcome. Finally at the end of the election, a number of chosen Auditors (As) use their Techincal Aids (TAs) to verify the proofs generated by the system which ensures the integrity of the election, and that the ballots are *tallied-as-recorded*.

The above summarizes the voting and tallying phases respectively, of an election held using the Swiss Post e-voting system. Before these phases, the system runs a configuration phase, consisting of the two protocols (SetupVoting) and (SetupTally). Each protocol configures and distributes a number of important values, asymmetric key pairs, and credentials needed for the later two phases. The As of the election also feature here to verify any Zero Knowledge Proofs (ZKPs) produced during the two protocols after they are done to verify their integrity. For the rest of this section we will briefly describe the participants of both protocols, and take a close look at how they are defined, in particular focusing on the generation of asymmetric cryptographic keys.

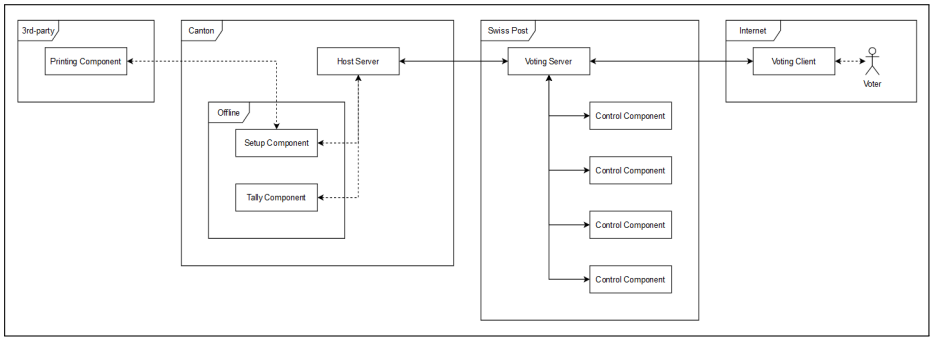


Figure 2.1: A simplified overview of the system's components and how they are connected according to [Swi24a, p. 31]. Normal arrows define online communication channels while dotted arrows define offline communication.

2.2.1 Participants

The configuration phase's (SetupVoting) protocol includes the following participants: one Setup Component (SC), one Printing Component (PC), one Voting Server (VS), four CCRs, one T, the Electoral Board (EB), and the Vs.

The SC is only used during the configuration phase, and is an offline component located on the premises of the canton running the election. This component is responsible for generating a number of confidential values including voter credentials, cryptographic keys, and return codes required for individual verifiability.

The PC is only used during the configuration phase, and is responsible for printing the physical voting cards, which include credentials and return codes belonging to each V participating in the election. The values used to fill in the voting cards are received from the SC. When PC is done printing, the voting cards are sent by postal mail to each voter.

The VS, located at Swiss Post, is used both during the configuration phase and the voting phase. Its main job is to act as an intermediary between the SC and all CCRs during the configuration phase, and V and all CCRs during the voting phase. Additionally it also initializes and maintains some public parameters during both phases.

CCRs, also located at Swiss Post, are used during all three phases and act like the ballot box of a physical election. During the configuration phase they assist the SC in generating and signing certain parameters, while during the voting phase they are the components that receive user ballots and confirmation codes, verify the eligibility of the ballots and send return codes back to the user. Finally during the

tally phase they participate in mixing and decrypting the stored votes, such that each ballot can be tallied and no link can be made to the voter who cast them.

Lastly is the offline T located at the canton hosting the election, and the EB, which is comprised of four human members. Both of these participants only feature during the configuration phase and the tally phase. During the configuration phase, each member of the EB inputs their own secret password into the SC, in order to construct a commitment the secret key the T will use to decrypt ballots during the tally phase. When the election reaches the tally phase, the members of the EB are finally asked to give their passwords to the T such that it may perform the final decryption of the ballots and produce the election result.

2.2.2 Asymmetric Key Cryptography

The current configuration phase of the Swiss Post e-voting system produces six sets of asymmetric key pairs. Three of these sets contain elgamal key pairs [Elg85] used for elgamal encryption and decryption, while the remaining three contain exponentiation key pairs, where the secret key is used for exponentiating certain values. All of these sets are generated over the same group $\mathbb{G}$ for which both 3.2 and 3.3 hold. To ensure this is the case, $\mathbb{G}$ is defined as the group of all quadratic-residues modulo $p \in \mathbb{P}_{3072}$ of order $q \in \mathbb{P}_{3071}$, where $p = 2q + 1$ is satisfied. The generator g for the group is chosen as the smallest value satisfying $g \in \mathbb{G}, g \neq 1$. Sets of asymmetric key pairs are denoted as follows $(pk \in \mathbb{G}^x, sk \in \mathbb{Z}_q^x)$, where pk is the list of public keys and sk is the list of of secret keys. Key pairs correspond by their index in each list, and both lists are of size x.

Additionally, four of the sets are generated in a distributed fashion, meaning multiple participants work together to create a list of combined public keys. To achieve this, each participant creates their own local secret key share, and publish the corresponding public key share. Each public key share is shared with each participant, to create the list of combined public keys, for which no participant knows the full list of corresponding combined secret keys.

Finally, we note that system uses a variation of the elgamal encryption scheme [Elg85] called multi-recipient elgamal [BBS03], where the main difference is that a single randomness can be used when encrypting multiple different plaintexts with multiple different elgamal key pairs.

2.2.3 SetupVoting

In the (SetupVoting) protocol, participants generate and distribute: the election public parameters, voter credentials, return codes, and asymmetric key pairs required for message encryption and exponentiation during the voting phase. Table 2.1

summarizes all algorithms executed in this protocol as defined by [Swi24c]. In this section we will mostly focus on how asymmetric key pairs are generated, and how they are used. As such, Figure 2.2 and Figure 2.3 show a simplified view of the (SetupVoting) protocol, which for clarity omitts: zero-knowledge proofs generated by the CCRs to prove the integrity of the parameters they have produced to the As; the function GetVoterAuthenticationData, as it is only used to create the authentication challenge used by the VS to authenticate the voter during the voting phase; and the functions CombineEncLongCodeShares and GenVerCardSetKeys, as they are only used to combine the outputs of the CCRs. Below we describe the functions of Figure 2.2 in more detail.

Function	Component
GenSetupData	SC
GenVerDat	SC
GetVoterAuthenticationData	SC
GenKeysCCR	CCR
GenEncLongCodeShares	CCR
CombineEncLongCodeShares	SC
GenCMTable	SC
GenVerCardSetKeys	SC
GenCredDat	SC

Table 2.1: All functions in the order they are executed during the configuration phase (SetupVoting) protocol, along with the components that run them.

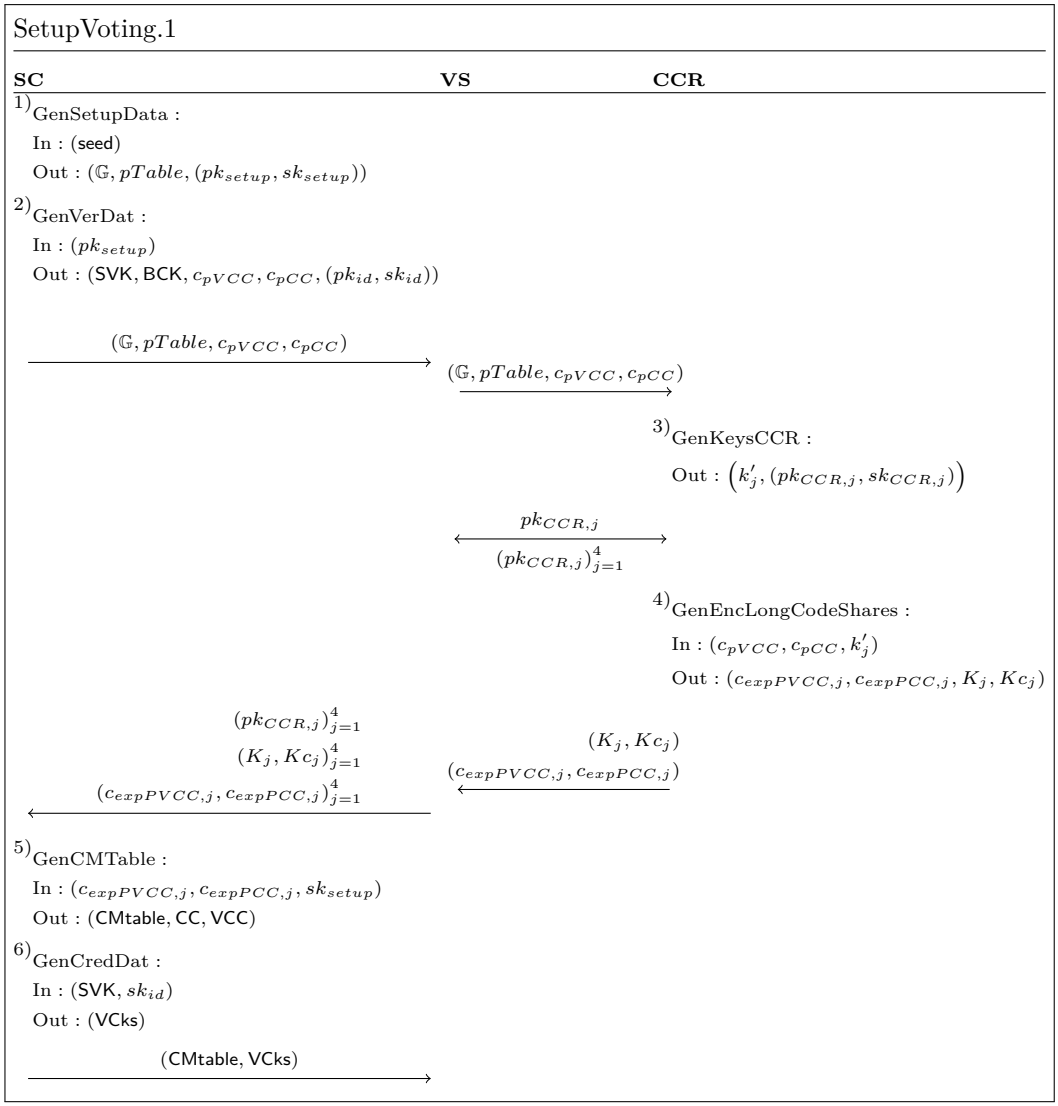


Figure 2.2: Simplified sequence diagram for the (SetupVoting) protocol part 1 [Swi24c].

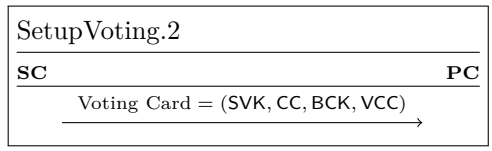


Figure 2.3: Simplified sequence diagram for the (SetupVoting) protocol part 2 [Swi24c].

1. The **GenSetupData** function takes seed as input, which is the unique identifier of the election in progress. The seed is used to sample the parameters (p, q) for group $\mathbb{G}$, after which an appropriate generator g is chosen. Once $\mathbb{G}$ has been chosen, the list $pTable$ is generated, mapping unique small primes $\mathrm{p} \in \mathbb{G}$ to unique voting options in the election. Finally, the elgamal key pair $(pk_{setup} \in \mathbb{G}^n, sk_{setup} \in \mathbb{Z}_q^n)$ is generated, where n is the total number of voting options in the election.

2. The **GenVerDat** function takes (pk_{setup}) as input, and firstly samples a random SVK, BCK, and exponentiation key pair $(pk_{id} \in \mathbb{G}^{n_v}, sk_{id} \in \mathbb{Z}_q^{n_v})$, where n_v is the number of Vs. Next, the function produces the two-dimensional list of pre-CCs $pCC = \{\{p^{k_{id}}\}_{p \in pTable}\}_{k_{id} \in sk_{id}} \in \mathbb{G}^{n \times n_v}$, and the list of pre-VCCs $pVCC = \{H(\mathsf{BCK})^{k_{id}}\}_{k_{id} \in sk_{id}} \in \mathbb{G}^{n_v}$. We call these values the pre-CC and pre-VCC respectively, as they are the values used by the CCRs during the voting phase to derive the correct CCs and VCCs. Finally, SC hashes each value in both pCC and $pVCC$, before encrypting them using pk_{setup} into ciphertexts $c_{pCC} \in \mathbb{G}^{n \times n_v}$, and $c_{pCC} \in \mathbb{G}^{2 \times n_v}$ respectively.

3. The **GenKeysCCR** function, run by each CCR j, takes no input and produces distributed elgamal key pair $(pk_{CCR,j} \in \mathbb{G}^{n_s}, sk_{CCR,j} \in \mathbb{Z}_q^{n_s})$, where n_s is the total number of vote selections, and secret key $k'_j \in \mathbb{Z}_q$ which is used to derive further keys later. Here vote selection is understood to define a number of mutually exclusive voting options the voter has to choose between in their ballot. Afterwards, each CCR shares $(pk_{CCR,j}, sk_{CCR,j})$ with each other through the VS, such that each $pk_{CCR,j}$ can be combined, through multiplication, into a common $pk_{CCR} \in \mathbb{G}^{n_s}$.

4. The **GenEncLongCodeShares** function, also run by each CCR j, takes $(c_{pVCC}, c_{pCC}, k'_j)$ as input. First j uses k'_j to derive the two distributed exponentiation key pairs $(K_j \in \mathbb{G}^{n_v}, k_j \in \mathbb{Z}_q^{n_v})$ and $(Kc_j \in \mathbb{G}^{n_v}, kc_j \in \mathbb{Z}_q^{n_v})$. Finally, the function produces outputs $(c_{expPCC,j}, c_{expPVCC,j})$ by exponentiating c_{pCC} and c_{pVCC} respectively with the keys in k_j and kc_j belonging to the same voter.

5. The **GenCMTable** function takes the ciphertexts $(c_{expPCC,j}, c_{expPVCC,j})$ from all CCRs, and sk_{setup} as input. First, the lists of ciphertexts from each CCR are combined into $(c_{expPCC}, c_{expPVCC})$, which are then decrypted using sk_{setup} into codes $(s_{expPCC}, s_{expPVCC})$. Next, $(n \times n_v)$ CCs are sampled, before they are encrypted by a symmetric key derived from the code in s_{expPCC} corresponding to the same voter and voting option. Similarly a random VCC is sampled per voter and encrypted by a symmetric key derived from the corresponding code in $s_{expPVCC}$. Finally, these encrypted CCs and VCCs are stored in CMtable, which is ordered lexiographically.

6. The **GenCredDat** function, on input SVK and sk_{id}, generates VCks similarly to how CMtable. For each voter, the function derives a symmetric key from their SVK, which it uses to encrypt their sk_{id} before it is stored in VCks. Once finished, SC sends CMtable and VCks to the VS, so that they may be used during the voting phase.

At the end of the (SetupVoting) protocol, SC sends the voting cards of each V to PC to be printed as shown in Figure 2.3.

2.2.4 SetupTally

The (SetupTally) protocol is much shorter than (SetupVoting), and has the more succinct end goal of configuring the election elgamal key pair used to encrypt the ballots cast by the Vs during the voting phase. To achieve this it uses the functions defined in Table 2.2. A simplified view of the full protocol is shown in Figure 2.4, which much like Figure 2.2 omitts zero-knowledge proofs produced by the CCRs, and assumes (SetupVoting) has already happened, meaning certain participants already has required context $(\mathbb{G}, pTable)$. Below we describe the functions of Figure 2.4 in more detail.

Function	Component
SetupTallyCCM	CCR
SetupTallyEB	SC

Table 2.2: All functions in the order they are executed during the configuration phase (SetupTally) protocol, along with the components that run them.

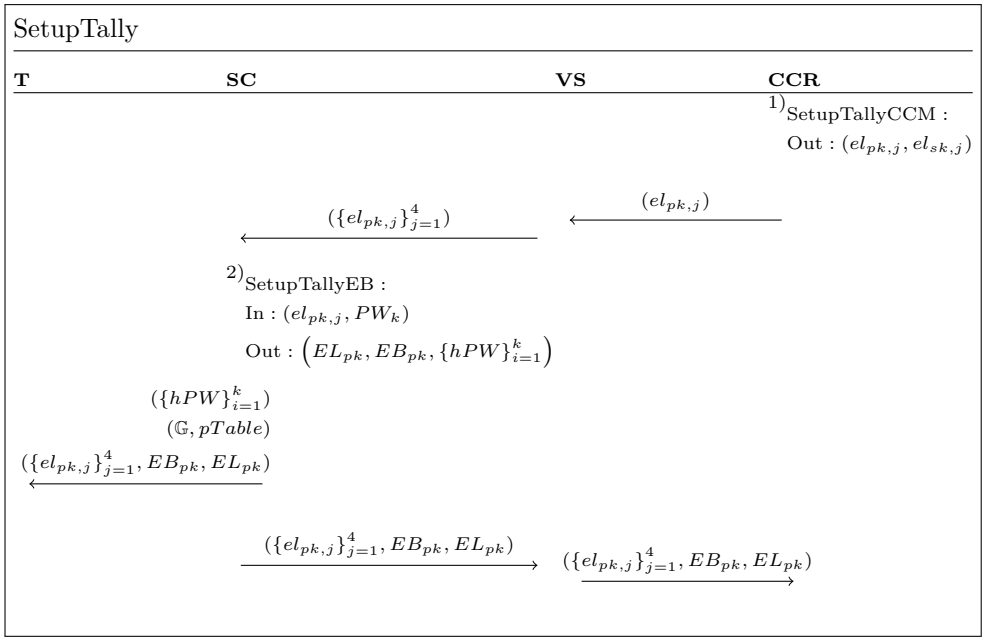


Figure 2.4: Simplified sequence diagram for the (SetupTally) protocol [Swi24c].

1. The **SetupTallyCCM** function runs in each CCR j, takes no input, and produces distributed elgamal key pair $(el_{pk,j} \in \mathbb{G}^{w+1}, el_{sk,j} \in \mathbb{Z}_q^{w+1})$, where w is the number of write-in options in the election.

2. The **SetupTallyEB** function takes $el_{pk,j}$, from each CCR j, and password PW_k of each EB member k as input. Each password is written directly into the SC, before a hash hPW_k of each password, and the electoral board elgamal key pair $(EB_{pk} \in \mathbb{G}^{w+1}, EB_{sk} \in \mathbb{Z}_q^{w+1})$, are produced. The electoral board elgamal key pair and the elgamal key pairs of each of the CCRs j are finally combined into a single election public key $EL_{pk} \in \mathbb{G}^{w+1}$. All the keys are distributed to each relevant component such that their correctness may be verified, while T also receives the hashed passwords, that act as commitments for when the members of the EB will once again give their passwords for the final decryption.

2.3 Threat Models

The Swiss Post computational proof [Swi24b] also describes threat models for each of the e-voting properties the system aims to satisfy. In particular, each threat model has

an associated table of trust assumptions, describing which system participants must be trustworthy in order to achieve the desired property. A participant is considered trustworthy if they keep private data secret and only perform operations specified by the system protocol. In contrast, Untrustworthy participants are considered to all have the ability to share any information between each other and may perform whatever operations they like. Table 2.3 summarizes the trust assumptions for each of the e-voting properties defined in Section 2.1.

Participant\Property	Individual Verifiability	Universal Verifiability	Vote Secrecy
Voters*	✓	✓	✓
Voting Client	-	-	✓
Voting Server	-	-	-
Control Component	1/4 ✓	1/4 ✓	**1/5 ✓
Tally Component	-	-	
Electoral Board	-	-	***1/4 ✓
Setup Component	✓	-	✓
Printing Component	✓	-	✓
Auditors*	-	✓	-
Technical Aid*	-	✓	-

Table 2.3: Table of trust assumptions on each system participant for individual verifiability, universal verifiability, and vote secrecy. * At least some participants are trustworthy. ** Either one Control Component or the Tally Component must be honest. *** If Tally Component is honest, at least 1 Electoral Board member must be honest.

2.4 Observations

In this section we make some observations regarding the sets of distributed asymmetric key pairs described in Section 2.2.3 and Section 2.2.4, in particular focusing on whether the key pairs may be biased by an adversary $\mathcal{A}$ under the current trust assumptions for the different threat models of the system, and what effect this could have on security. Usually, asymmetric key generation in the single-party setting first samples a secret key sk uniformly, and computes a public key pk from this, resulting in a uniform public key as well. In the distributed setting however, where multiple participants produce a combined public key pk_c for an unknown combined secret key sk_c, it is sometimes possible for an adversary to bias the keys, meaning they are somehow distinguishable from a uniformly generated key pair. This negatively impacts the secrecy of the secret key, and could invalidate computational proofs if they rely on the key pairs being uniformly sampled.

First we look at the following distributed keys generated during the (SetupVoting) protocol ($pk_{ccr} = \prod_{j=1}^{4}(pk_{ccr,j}) \in \mathbb{G}^{n_s}, K = \prod_{j=1}^{4}(K_j) \in \mathbb{G}^{n_v}, Kc = \prod_{j=1}^{4}(Kc_j) \in \mathbb{G}^{n_v}$). In the case of pk_{ccr}, Figure 2.2 shows that each CCR generates their share of the public key before forwarding it to the other CCRs. An adversary $\mathcal{A}$, able to corrupt one or more CCRs, would here be able to bias pk_{ccr} by first waiting for the honest CCRs to broadcast their shares of the public key, before repeatedly generating their own shares until a pk_{ccr} with a particular bias is created. The case of (K, Kc) is slightly different in that the public key shares are only ever sent to the VS, and not to the other CCRs. Though it still stands that an adversary $\mathcal{A}$, able to corrupt VS and one or more CCRs, is able to bias both (K, Kc).

By the trust assumptions for each of the properties presented in Table 2.3, we see that each threat model considers both one or more CCRs and the VS untrustworthy, meaning these public keys cannot be considered unbiased for any property. The computational proof for vote secrecy, presented in [Swi24b, Sections 18.2-18.3], does however not rely on any of these public keys being unbiased, or on the secrecy of the corresponding secret keys, leaving vote secrecy is unaffected. The computational proof for individual verifiability is similar in that it does not requires the public keys to be unbiased. We note however that pk_{ccr} is only ever used to encrypt certain values sent from the VC to the CCRs during the voting phase. As Table 2.3 shows that the VC is considered untrustworthy in the individual verifiability threat model, an adversary looking to exploit the weaker secrecy of sk_{ccr} could rather avoid the encryption altogether by corrupting the VC. On the other hand, the secrecy of keys (k, kc) is particularly important in order to avoid an adversary breaking *sent-as-intended*, and achieving *vote rejection* respectively.

The second distributed key generation that occurs, during the (SetupTally) protocol, is different in that it may or may not be considered unbiased depending on which components are trustworthy. In this scheme, $(EL_{pk} = \prod_{j=1}^{4}(el_{pk,j}) \cdot EB_{pk} \in \mathbb{G}^{w+1})$ is generated with share contributions from the CCRs and EB members, that are gathered and combined in the SC. This set of keys is only used for encrypting the actual voting options selected by the voter during the voting phase. To that extent, in order to guarantee the vote secrecy of the system, the threat model specifies that either a CCR must be trustworthy, or both an EB member and T must be trustworthy. Interestingly, since the SC is considered trustworthy, and does not reveal EB_{pk} before it reveals EL_{pk}, the only way an adversary can pre-compute, and therefore bias, EL_{pk}, is if they control the entire EB. As such, only the first scenario allows an adversary to bias EL_{pk}, though again the computational proof for vote secrecy in the Swiss Post e-voting system does not require EL_{pk}, and therefore also not EL_{sk}, to be guaranteed unbiased. It is trivial to see however that the secrecy of EL_{sk} is vital to the vote secrecy of the system, and as such it would be ideal if it was properly uniformly sampled.

Chapter 3
Background

3.1 Notation

Security notation. Let λ be the security parameter, and let $\mathsf{negl}(\lambda)$ describe a negligible function in the security parameter.

Group and field notation. Let $\mathbb{Z}_x$ denote the set of integers $\{0, \ldots, x-1\}$, and let $\mathbb{P}_x$ denote the set of x-bit prime integers. Furthermore, let $\mathbb{G}$ denote the multiplicative group $\langle g \rangle_{q,p} = \{g^i \mod p : \forall i \in \mathbb{Z}_q\}$ for some generator g of order q modulo p.

3.2 Computationally Hard Problems

In cryptography, when a protocol cannot be proven unconditionally secure, cryptographers instead attempt to prove computational security. This is done by reducing the problem of breaking the protocol into breaking a more well-studied problem. Whether a well-studied problem is hard depends on if an efficient, probabilistic, or deterministic, algorithm solving it has been found. Here "efficient" means the algorithm runs in polynomial time, and if no such algorithm is known we say the protocol is secure against efficient adversaries. In this section, we list a few such problems relevant to protocols and primitives to be discussed later.

Let $\mathbb{G}$, for some generator g of order q with modulo p, be public knowledge for all problems described below.

Definition 3.1. Discrete Logarithm Problem (DL) [DH76] Given $y = g^x \in \mathbb{G}$, where $x \leftarrow\!\!\$\ \mathbb{Z}_q$ is secret, it is difficult to compute $x = \log_g(y)$.

Definition 3.2. Computational Diffie-Hellman Problem (CDH) [DH76] Given $(g^{x_1}, g^{x_2}) \in \mathbb{G}$, where $x_1, x_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ is secret, it is difficult to compute $y = g^{x_1 \cdot x_2}$.

Definition 3.3. Decisional Diffie-Hellman Problem (DDH) [DH76] Given $(g^{x_1}, g^{x_2}, g^{x_3}) \in \mathbb{G}$, where $x_1, x_2 \leftarrow\$ \mathbb{Z}_q$ is secret, it is difficult to determine whether $x_3 = x_1 \cdot x_2$ or $x_3 \leftarrow\$ \mathbb{Z}_q$.

Definition 3.4. *l*-**Strong DH Exponent Problem (l-SDHE) [ZSS04; Che06]** Given $\left(g^{x^1}, \ldots, g^{x^l}\right) \in \mathbb{G}$, where $x \leftarrow\$ \mathbb{Z}_q$ is secret, it is difficult to compute $g^{x^{l+1}}$.

3.3 Cryptograhpic Hash Functions

Cryptographic Hash Functions (CHFs) are deterministic functions that efficiently, given a variable size input, produce a fixed size output. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a hash function, mapping a variable size domain to an λ-bit codomain. Such a function can be considered a CHF if it satisfies the following three properties:

Definition 3.5. Pre-Image Resistance Given $y \in \{0,1\}^{2\lambda}$, it is difficult for an adversary $\mathcal{A}$ to find any $x \in \{0,1\}^*$ such that $y = \mathsf{H}(x)$. This is to say:

$$\Pr\left[\, y = \mathsf{H}(x) \;\middle|\; \begin{array}{l} y \in \{0,1\}^\lambda \\ x \leftarrow \mathcal{A}(\mathsf{H}, y) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

Definition 3.6. Second Pre-Image Resistance Given $x \in \{0,1\}^*$, it is difficult for an adversary $\mathcal{A}$ to find any $x' \in \{0,1\}^*$ such that $x \neq x'$, and $\mathsf{H}(x) = \mathsf{H}(x')$. This is to say:

$$\Pr\left[\begin{array}{l} x \neq x' \\ \mathsf{H}(x) = \mathsf{H}(x') \end{array} \;\middle|\; \begin{array}{l} x \in \{0,1\}^* \\ x' \leftarrow \mathcal{A}(\mathsf{H}, x) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

Definition 3.7. Collision Resistance It is difficult for an adversary $\mathcal{A}$ to find any $(x \in \{0,1\}^*, x' \in \{0,1\}^*)$ such that $x \neq x'$, and $\mathsf{H}(x) = \mathsf{H}(x')$. This is to say:

$$\Pr[\mathsf{H}(x) = \mathsf{H}(x') \mid (x, x') \leftarrow \mathcal{A}(\mathsf{H})] \leq \mathsf{negl}(\lambda)$$

3.4 Random Oracle Model

In cryptography, protocols are proven secure under a given model which allows cryptographers to make assumptions about the environment a protocol is running in. Ideally we want a protocol to be secure in the *standard model*, which only assumes that adversaries are somehow bounded by time and computational power. Proofs in this model rely on hard problems such as those described in the previous section, but it is not always easy to produce proofs in this model. This is why other models have been defined, that use other assumptions that seem probable but which have not been proved yet.

In the Random Oracle Model (ROM) [BR93], we assume that random oracles exist, which, as a cryptographic primitive, act as idealized CHFs. When random oracles receive an input for the first time, they return an output uniformly at random from their codomain. They also commit this output to memory, such that it is returned every time they receive the same input later. Besides satisfying the same security properties as CHFs, their output can also be considered truly random. Therefore, when a protocol using a CHF requires that the output is randomly generated to be proven secure, it is common to model it in ROM, and to replace the CHF with a random oracle.

3.5 Zero Knowledge Proof

A ZKP [GMR89], is an interactive proof protocol, where an honest prover P tries to convince an honest verifier V that some public input x is true, without revealing any additional information beyond that fact. On initiating the protocol, both P and V are given a public input x, for which P will know the witness w_x. After exchanging a number of messages, V ends the protocol by either accepting or rejecting x, depending on whether they are convinced x is true. Additionally, the protocol must also prevent: dishonest provers P^*, that don't know w_x, from convincing V that x is true, and dishonest verifiers V^* from discovering any additional information besides the fact that x is true. These concepts are summarised in three important properties of ZKPs: *Completeness*, *Soundness*, and *Zero-Knowledge*.

Let R be a relation on language L, where $x \in L$ only if $(x, w_x) \in R$. Furthermore, let Setup be a setup algorithm that produces public setup parameters sp on input of the security parameter λ. Finally, let $(\mathsf{P}(sp, x, w_x), \mathsf{V}(sp, x))$ be the output of V on public input x after interaction with P knowing witness w_x. An interactive proof protocol is considered a secure ZKP protocol Π, if it satisfies the following three properties:

Definition 3.8. Completeness A ZKP is *complete*, if an honest P will always convince an honest V that x is true. This is to say, for any efficient sampling algorithm P_0:

$$\Pr\left[(\mathsf{P}(sp, x, w_x), \mathsf{V}(sp, x)) = 1 \;\middle|\; \begin{array}{l} sp \leftarrow \mathsf{Setup}(\lambda) \\ (x, w_x) \leftarrow \mathsf{P}_0(sp) \\ (x, w_x) \in R \end{array} \right] = 1$$

Definition 3.9. Soundness A ZKP is considered *sound*, if it is difficult for P^* to convince V that x is true. This is to say:

$$\Pr\left[(\mathsf{P}^*(sp,x,\cdot),\mathsf{V}(sp,x))=1 \;\middle|\; \begin{array}{l} sp \leftarrow \mathsf{Setup}(\lambda) \\ \forall x \notin L \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 3.10. Honest-Verifier Zero-Knowledge A ZKP is considered *honest-verifier zero-knowledge*, if it is difficult for an honest but curious verifier V' to learn anything beyond the fact that $x \in L$. This is to say, given transcript $T_{(\mathsf{P}(sp,x,w_x),\mathsf{V}(sp,x))}$ from a real accepting conversation between P and V, and transcript $S_{(\mathsf{P}(sp,x,\cdot),\mathsf{V}(sp,x))}$ generated by simulator S on input x, the following holds:

$$2 * \left| \Pr\left[b = b' \;\middle|\; \begin{array}{l} sp \leftarrow \mathsf{Setup}(\lambda) \\ T_0 = T_{(\mathsf{P}(sp,x,w_x),\mathsf{V}(sp,x))} \leftarrow \Pi(sp,x,w_x) \\ T_1 = S_{(\mathsf{P}(sp,x,\cdot),\mathsf{V}(sp,x))} \leftarrow \mathsf{S}(sp,x) \\ b \leftarrow\!\!\$\ \{0,1\} \\ b' \leftarrow \mathsf{V}'(T_b,sp,x) \end{array}\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Here we choose to define the weaker property of *honest-verifier zero-knowledge* as it is simpler than full *zero-knowledge*, and sufficient for the purposes of this thesis.

3.5.1 Proof of Knowledge

An interactive proof protocol is a Proof of Knowledge (PoK) when the goal of the protocol is for prover P to convince verifier V that they know witness w_x corresponding to public input x. In loose terms, "P knowing w_x" means that it is possible to extract the information from them somehow. To capture this idea, Bellare and Goldreich [BG93] introduce a machine called a knowledge extractor E, which is used to define a new property *knowledge soundness*. An interactive proof protocol can be considered a PoK if it fulfills the properties of *completeness*, as described in the previous section, and *knowledge soundness*, which is a stronger variant of *soundness*.

Let R be a relation on language L, where $x \in L$ only if $(x, w_x) \in R$.

Definition 3.11. Knowledge Soundness A PoK is considered *knowledge sound* if, whenever arbitrary P manages to convince an honest V that x is true, there also exists a polynomially bounded E with black-box access to P, who can output w_x

such that $(x, w_x) \in R$. This is to say:

$$\Pr\left[\begin{array}{l|l} & sp \leftarrow \mathsf{Setup}(\lambda) \\ (x, w_x) \in R & (\mathsf{P}(sp, x, \dots), \mathsf{V}(sp, x)) = 1 \\ & w_x \leftarrow \mathsf{E}^{\mathsf{P}}(sp, x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

3.5.2 Σ-protocols

Σ-protocols [Cra96] define a class of interactive proof protocols that are both honest-verifier zero-knowledge, a weaker notion of zero-knowledge, and proofs of knowledge. In these protocols, P interacts with V in the following sequence of 3 steps:

1. **Commitment step**: In the first step P generates a public commitment R, from some secret randomness r, which is sent to V.
2. **Challenge step**: On receiving R, V generates a random challenge c which is sent back to P.
3. **Response step**: In the last step, P computes some response s that is sent to V who, holding the protocol transcript (R, c, s), decides whether to accept or reject.

Besides having this 3-step form, Σ-protocols must also satisfy the properties of *completeness*, *honest-verifier zero-knowledge* and *knowledge soundness* as described in earlier sections. Instead of proving *knowledge soundness*, the stronger property *special soundness* is usually proven instead, as it is easier to show and is sufficient to prove a σ-protocol is a PoK.

3.5.3 Discrete Logarithm Equality Proof

A particularly relevant Σ-protocol, is the Discrete Logarithm Equality Proof (DLEQ) based on the Chaum-Pedersen protocol [CP93]. In this protocol P aims to prove to V that "they know two discrete logarithms and that they are equal" without revealing the discrete logarithm.

Interactive Protocol

Let $x = (\mathbb{G}, q, g, h, y, z)$ be public input known by both P and V, where $\mathbb{G}$ is the multiplicative group for some generator g of prime order q with prime modulo p, and $h \in \mathbb{G}$. Furthermore, let $w_x \leftarrow \mathbb{Z}_q$ be private input known only by P such that $y = g^{w_x}$ and $z = h^{w_x}$. In this case, P wants to prove to V that they know

$w_x = \log_g(y) = \log_h(z)$, without revealing w_x. We define the 3 steps of the Σ-protocol as follows:

1. **Commitment step**: P sends to V public commitment $R = (R_1, R_2) = (g^r, h^r)$, where $r \leftarrow \mathbb{Z}_q$.
2. **Challenge step**: V samples challenge $c \leftarrow \mathbb{Z}_q$ and sends it to P.
3. **Response step**: P computes response $s = r + c \cdot w_x$ which is sent back to V.

V uses the transcript (R, c, s) to verify the two equations $g^s = R_1 \cdot y^c$, $h^s = R_2 \cdot z^c$ and accepts the proof if both are true.

The protocol fulfills the property *completeness*. It trivially has the usual 3-step form required. Furthermore, an honest P will always choose an x for which they know w_x. As such, given an honest V, they will always be able to produce a full accepting transcript (R, c, s), which will always be accepted.

The protocol also fulfills *special soundness*. To show this we define two accepting transcripts for the same (x, w_x) pair as (R, c, s) and (R, c', s'), where $c \neq c'$. Anyone receiving these transcripts can trivially compute:

$$\frac{s - s'}{c - c'} = \frac{(r + c \cdot w_x) - (r + c' \cdot w_x)}{c - c'} = \frac{(r - r) + (c - c') \cdot w_x}{c - c'} = w_x$$

Finally, the protocol fulfills *honest-verifier zero-knowledge*. To illustrate we define a simulator that takes x as input. They start by randomly sampling s and c, where the latter value is chosen randomly to emulate an honest verifier following protocol. Afterwards they compute $R_1 = g^s * y^{-c}$ and $R_2 = h^s * z^{-c}$. The result is that the simulator produces accepting transcripts that are indistinguishable from those produced by real executions of the protocol with an honest verifier. For the purposes of this paper, it is sufficient that DLEQ is *honest-verifier zero-knowledge*, as we will only use the non-interactive variant of it anyways, which in ROM is equivalent to having an honest verifier.

Non-interactive Protocol

As DLEQ is a Σ-protocol, we can use the Fiat-Shamir heuristic [FS87] to turn it into a non-interactive protocol. In this version a CHF H replaces V in generating c, resulting in the following protocol:

1. P shares public input $x = (\mathbb{G}, q, g, h, y, z)$ with V.

2. P computes $\pi_{DLEQ} = (c, s = r - c \cdot w_x) = \text{ProveDLEQ}(w_x, g, y, h, z)$, and sends π_{DLEQ} to V.

3. V accepts if $\text{VerifyDLEQ}(\pi_{DLEQ}, g, y, h, z) = 1$, and rejects otherwise.

for the following two algorithms:

ProveDLEQ(w_x, g, y, h, z)

1 : $r \leftarrow\$ \mathbb{Z}_q$
2 : $c = H(g, y, h, z, g^r, h^r)$
3 : $\pi_{DLEQ} = (c, s = r - c \cdot w_x)$
4 : return π_{DLEQ}

VerifyDLEQ(π_{DLEQ}, g, y, h, z)

1 : $c' = H(g, y, h, z, g^s \cdot y^c, h^s \cdot z^s)$
2 : if $c \neq c'$:
3 : return 0
4 : return 1

The non-interactive DLEQ protocol is considered secure in ROM. In the interactive DLEQ protocol, zero-knowledge is only guaranteed when the verifier is honest, as only then can the protocol guarantee that c is not maliciously generated. In the non-interactive protocol, the verifier is replaced by H, which in the standard model is a deterministic CHF. To generate c with the same randomness as an honest verifier, we use ROM to model H as a random oracle. The result is that the protocol becomes zero-knowledge, independent of the verifier.

3.6 Pseudorandom functions

Informally PRFs, first introduced by Goldreich et al. [GGM86], are functions that produce pseudorandom output when given a random key and some deterministic input. More formally we define them as follows.

Definition 3.12. Pseudorandom Function Let F be a collection of efficiently computable functions on the form:

$$F : K \times X \to Y$$

with index-space K, domain X, and range Y. We say that F is a secure PRF if, given a random $k \in K$ and some $x \in X$, it produces pseudorandom output $y \in Y$. In other terms, F is a secure PRF if, given random $k \in K$, it produces output indistinguishable from that of a random oracle with the same domain X and range Y.

3.6.1 Naor-Reingold PRF

One particularly relevant construction of a PRF was defined by Naor and Reingold in [NR04]. Its security relies on the DDH-assumption, and is defined as follows.

Definition 3.13. Naor-Reingold PRF Let $\mathbb{G}$ be defined for some generator g with prime modulo p of prime order q that divides $(p-1)$. Furthermore, let $a = (a_0, \ldots, a_n)$ be a sequence of $n+1$ elements of $\mathbb{Z}_q$, and $x = (x_1, \ldots, x_n)$ be a sequence of n elements of $\{0,1\}$. Then $F_{a,g}$ is a Naor-Reingold PRF defined by the following equation:

$$F_{a,g}(x) = g^{a_0 \cdot \prod_{i=1}^{n} (a_i^{x_i})}$$

with uniformly random chosen a, and input x.

3.7 Verifiable Random Function

Verifiable Random Functions (VRFs), first defined by Micali et al. [MRV99], are an extension of PRFs, and are defined as follows.

Definition 3.14. Verifiable Random Functions A VRF scheme consists of the following three algorithms:

- KeyGen(λ): Outputs secret key $sk \in \{0,1\}^\lambda$, verification key pk, and PRF $f : \{0,1\}^\lambda \times X \to Y$.
- Eval(sk, x): Outputs evaluation $y = f(sk, x)$, and proof π.
- Verify(pk, x, y, π): Verifies that $y = f(sk, x)$ according to π, and outputs 1 if true, otherwise 0.

Such a VRF scheme is considered secure if f is a secure PRF, and it fulfills the following two requirements:

1. **Correctness**:

$$\Pr\left[\text{Verify}(pk, x, y, \pi) = 1 \;\middle|\; \begin{array}{l}(sk, pk, f) \leftarrow \text{KeyGen}(\lambda) \\ (y, \pi) \leftarrow \text{Eval}(sk, x)\end{array}\right] = 1$$

2. **Uniqueness**:

$$\Pr\left[\begin{array}{l|l} y_1 \neq y_2 & (sk, pk, f) \leftarrow \text{KeyGen}(\lambda) \\ \text{Verify}(pk, x, y_1, \pi_1) = 1 & (y_1, y_2, \pi_1, \pi_2) \leftarrow \mathcal{A}(sk, pk) \\ \text{Verify}(pk, x, y_2, \pi_2) = 1 & \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3.7.1 Attribute-Based Verifiable Pseudorandom Function

In their whitepaper, Huang et al. [HIJ+21] describe how they achieve anonymous collection of data from authenticated clients through what they call an Attribute-Based Verifiable Oblivious Pseudorandom Function. Their construction is based on a similar one defined by Everspraugh et al. [ECS+15], and consists of seven algorithms. The first three of these algorithms can be used to create a VRF scheme for generating many asymmetric keys from a single master key. To achieve this the scheme uses the Naor-Reingold PRF, along with the non-interactive variant of the DLEQ ZKP. We define this VRF scheme we call Attribute-Based Verifiable Random Function (AB-VRF) below.

Definition 3.15. Attribute-Based Verifiable Random Function Let AB-VRF be a VRF scheme, defined by three algorithms: (MasterKeyGen, PKGen, PKVerify). Let $\mathbb{G}$ be defined for some generator g, with prime modulo p, of prime order q that divides $(p-1)$. Furthermore, let $h \leftarrow\!\!\$\ \mathbb{G}$. The algorithms are defined as follows:

MasterKeyGen(n)	
1 :	$\text{msk} = (a_0, \ldots, a_n) \leftarrow\!\!\$\ \mathbb{Z}_q$
2 :	$\text{mpk} = (P_0 = g^{a_0}, h_1 = h^{a_1}, \ldots, h_n = h^{a_n})$
3 :	return (msk, mpk)

PKGen($t \in \{0,1\}^n, msk$)

1 : $\mathrm{sk}_t = a_0 \cdot \prod_{i=1}^{n} (a_i^{t[i]})$
2 : $\mathrm{pk}_t = g^{\mathrm{sk}_t}$
3 : $\pi_t = \varnothing$
4 : $P_{prev} = P_0$
5 : for $i = (1, \ldots, n)$ where $t[i] == 1$ do:
6 : $\quad P_i = P_{prev}^{a_i}$
7 : $\quad \pi_i = \mathrm{ProveDLEQ}(a_i, h, h_i, P_{prev}, P_i)$
8 : $\quad \pi_t \leftarrow (P_i, \pi_i)$
9 : $\quad P_{prev} = P_i$
10 : return $(\mathrm{sk}_t, \mathrm{pk}_t, \pi_t)$

PKVerify(pk_t, π_t)

1 : $P_{prev} = P_0$
2 : for all $(P_i, \pi_i) \in \pi_t$ do:
3 : $\quad$ if $\mathrm{VerifyDLEQ}(\pi_i, h, h_i, P_{prev}, P_i) \neq 1$:
4 : $\quad\quad$ return 0
5 : $\quad P_{prev} = P_i$
6 : if $\mathsf{pk}_t \neq P_{prev}$:
7 : $\quad$ return 0
8 : return 1

First the MasterKeyGen algorithm takes a publicly known input n, which decides the number of asymmetric keys that can be generated, and outputs a master secret key and master public key pair. Next, the PKGen algorithm uses the master key pair to derive an asymmetric key pair for the public input t, along with a chain of proofs that prove the secret key is derived from the master secret key without revealing either. Finally, the PKVerify algorithm allows anyone to verify the correspondence between the master public key and the public key using the corresponding list of ZKPs.

3.8 Commitment scheme

A commitment scheme is a cryptographic building block that allows a party to commit to a chosen value while keeping it a secret. The party can then, at a later stage

reveal(open) the value, allowing others to verify that the commitment corresponds to the value.

Theorem 3.16. *A commitment schemes consists of the following algorithms:*

- ***KeyGen(*λ*):*** *Outputs the public commitment key ck*
- ***Commit(*ck, m*):*** *Outputs a commitment cmt, and an opening op for m.*
- ***Open(*ck, cmt, op*):*** *Outputs* 1 *if the commitment is valid,* 0 *else.*

We say that a commitment scheme is hiding if it is computationally infeasible for an adversary to distinguish between a commitment and a uniformly random sampled value in the commitment space. More formally we say that

$$\Pr\Big[A(commit(ck,m)) = 1 \;\Big|\; (ck,m)\Big] - \Pr\Big[A(cmt \leftarrow \mathcal{C}) = 1 \;\Big|\; (ck,m) \leftarrow KeyGen(\lambda)\Big] \leq \mathsf{negl}(\lambda)$$

We say that a commitment scheme is binding if it is computationally infeasible for any adversary to find two different values that produce the same commitment. More formally we say that

$$\Pr\left[\begin{array}{l} ck = ck' \\ m \neq m' \\ commit(ck,m) = commit(ck',m') \end{array} \;\middle|\; \begin{array}{l} (ck,m) \leftarrow KeyGen(\lambda) \\ \big(ck',m'\big) \leftarrow KeyGen(\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3.9 Secret Sharing

Secret sharing is a cryptographic technique that enables the distribution of sensitive information among participants so that no individual possesses the secret. Instead, each participant holds a share of the secret, and collaboration among a subset of the group is required to reconstruct the original secret. We refer to this type of scheme as a (t, n) threshold scheme. With n participants involved, the threshold indicates that a minimum of $t + 1$ participants must collaborate to reconstruct the secret.

Definition 3.17. Shamir's Secret Sharing Let Shamir Secret sharing (Shamir's Secret Sharing (SSS))[Sha79] be defined by the algorithms: $(\text{Share}, \text{Reconstruct})$. Let $x \in \mathbb{Z}_q$ be a secret distributed as shares $(\sigma_1, \ldots, \sigma_n)$ securely among n parties, s.t. any set of at least $t+1$ can reconstruct x using Lagrange Interpolation as defined in [Kre18], denoted by $Interpolate$.

Share(x, n, t, q)

1 : $(f_1, \ldots, f_t) \leftarrow\$ \mathbb{Z}_q$

2 : $f(X) = x + \sum_{i=1}^{t} f(i) \cdot X^i$

3 : $(\sigma_1, \ldots, \sigma_n) = (f(1), \ldots, f(n))$

4 : return $(\sigma_1, \ldots, \sigma_n)$

Reconstruct$((\sigma_1, \ldots, \sigma_{t+1}), q)$

1 : $x = \text{Interpolate}(0, \sigma_1, \ldots, \sigma_{t+1})$

2 : return x

3.9.1 Verifiable Secret Sharing

Verifiable Secret Sharing is an extension of Secret Sharing by allowing the participants to verify the correctness of the received shares.

Definition 3.18. Feldman's Verifiable Secret Sharing Let Feldman's Verifiable Secret Sharing [Fel87] (Feldman's Verifiable Secret Sharing (FVSS)) be an extension of SSS, defined by the algorithms: (Share, Verify, Reconstruct). Let $x \in \mathbb{Z}_q$ be a secret distributed securely among n parties, as with SSS. For FVSS, we also generate $(y_0, \ldots, y_n)$, which are broadcasted publicly, and any participant can verify the correctness.

Share(x, n, t, q)

1 : $(f_1, \ldots, f_t) \leftarrow\$ \mathbb{Z}_q$

2 : $f(X) = x + \sum_{i=1}^{t} f(i) \cdot X^i$

3 : $(\sigma_1, \ldots, \sigma_n) = (f(1), \ldots, f(n))$

4 : $(y_1, \ldots, y_n) = (g^{\sigma_1}, \ldots, g^{\sigma_n})$

5 : return $(\sigma_1, \ldots, \sigma_n, y_1, \ldots, y_n)$

Verify$(i, y_i, (\sigma_0, \ldots, \sigma_t))$

1 : if $y_i \neq Interpolate(i, \sigma_0, \ldots, \sigma_t)$:

2 : return 0

3 : return 1

Reconstruct$((\sigma_1, \ldots, \sigma_{t+1}), q)$	
1 :	$x = \text{Interpolate}(0, \sigma_1, \ldots, \sigma_{t+1})$
2 :	return x

Definition 3.19. Pedersen Verifiable Secret Sharing Let Pedersen Verifiable Secret Sharing Pedersen Verifiable Secret Sharing (PVSS) [Ped92] be a secret sharing scheme, defined by the algorithms: (Share, Verify, Reconstruct). Let $x \in \mathbb{Z}_q$ be a secret distributed securely among n parties, PVSS generates two distinct polynomials $f(X), \hat{f}(X)$ and their respective shares $\sigma_i, \hat{\sigma_i}$. The dealer also broadcasts the values $y_i, \hat{y_i}$ to all participants. Participants can validate the correctness of the scheme, by verifying the commitment using the following equation.

$$g^{\sigma_i} h^{\hat{\sigma}_i} = \prod_{k=0}^{t} (y_k)^{i^k} \tag{3.1}$$

where g and h are generators of a cyclic group, t is the threshold for reconstruction, and i is the index of the participant. If the verification fails, the participant complains against the dealer. The dealer then has to reveal the shares $(\sigma_i, \hat{\sigma}_i)$. If the revealed equations also fails this, the dealer is disqualified.

Share(x, n, t, q)	
1 :	$(f_1, \ldots, f_t) \leftarrow\!\!\$\ \mathbb{Z}_q$
2 :	$\left(\hat{f}_1, \ldots, \hat{f}_t\right) \leftarrow\!\!\$\ \mathbb{Z}_q$
3 :	for i from 1 to n :
4 :	$f_i(X) = x + \sum_{j=1}^{t} f_j \cdot X^j$
5 :	$\hat{f}_i(X) = x + \sum_{j=1}^{t} \hat{f}_j \cdot X^j$
6 :	$(\sigma_{i1}, \ldots, \sigma_{it}) = (f_i(1), \ldots, f_i(t))$
7 :	$(\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{it}) = \left(\hat{f}_i(1), \ldots, \hat{f}_i(t)\right)$
8 :	$(y_{i1}, \ldots, y_{it}) = (g^{\sigma_{i1}}, \ldots, g^{\sigma_{it}})$
9 :	$(\hat{y}_{i1}, \ldots, \hat{y}_{it}) = \left(h^{\hat{\sigma}_{i1}}, \ldots, h^{\hat{\sigma}_{it}}\right)$
10 :	return $((\sigma_{i1}, \ldots, \sigma_{it}, y_{i1}, \ldots, y_{it}), (\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{it}, \hat{y}_{i1}, \ldots, \hat{y}_{it}))$

Verify$(g, h, (C_{i0}, \ldots, C_{it}), (s_{ij}, \hat{s}_{ij}))$

```
1 :  for j from 1 to n :
2 :      for k from 0 to t :
3 :          if g^{s_ij} · h^{ŝ_ij} ≠ (C_ik)_k^j   mod p :
4 :              return 0
5 :  return 1
```

Reconstruct$((\sigma_1, \ldots, \sigma_{t+1}), q)$

```
1 :  x = Interpolate(0, σ_1, ..., σ_{t+1})
2 :  return x
```

Chapter 4
Distributed Key Generation

Distributed Key Generation (DKG) is a cryptographic primitive that allows multiple parties to collaboratively generate a shared public key, without any single party learning the corresponding secret key. In a DKG protocol, n participants each generate a secret key share, and public key share. Constructing the shared public key requires combining all the public key shares, while constructing the secret key requires $t+1$ participants to cooperate. As with Secret Sharing (SS) schemes, multiple parties cooperate to reconstruct a secret. However, in the setting of DKG, we do not rely on a trusted dealer, removing the single point of failure. Rather, DKG allows the parties themselves to generate shares of the secret key, where $t+1$ participants are able reconstruct the secret key, and any t or fewer participants cannot learn anything about the secret key.

4.1 Fully Secure DKG

In this section we note the security properties of a fully secure DKG. Definitions presented are mainly based on those presented by Genarro et al. [GJKR99], while borrowing the wording for the definition of unbiased from Katz [Kat24].

Definition 4.1. **Correctness** We say that a DKG fulfills correctness if the following properties are fulfilled

1. All sets of $t+1$ honest participants define the same unique secret key x.

2. All sets of $t+1$ honest participants define the same unique public key $y = g^x \mod p$.

3. The secret key x is uniformly distributed in $\mathbb{Z}_q$, which implies that y also is uniformly distributed in $\mathbb{Z}_q$.

Definition 4.2. Secrecy We say that a DKG fulfills secrecy, if no information can be learned of the secret key, x except for what is given by the public key $y = g^x \mod p$.

Definition 4.3. Robustness In the presence of up to t malicious or faulty shares, it is possible to reconstruct the secret key x, given the public information and n shares.

4.2 Round complexity

The round complexity of a DKG is the number of communications required among participants, where each round typically consists of sending, broadcasting, or receiving messages, along with processing the information gathered. In this setting the round complexity of the protocols is static. Still, for other protocols, the number might depend on detecting malicious behavior resulting in a distinction between optimistic round complexity (if all the participants are honest) and worst case (for adversarial behavior).

4.3 Fully-Secure DKG in ROM by Katz

In this section we will present a DKG in which is fully secure under an honest majority by Katz [Kat24]. An honest majority means that more than $n/2$ participants in the protocol are honest and follow the protocol correctly. Let $\mathbb{S}_{n-t,n}$ denote the power set of all participants, $\mathbb{P}(P_i)$. We note that a set S must exist where all participants are honest. The lowest-index participant in each set S initiates the protocol, by generating the key k_S, which it sends to all other participants in the set. If any of the participants fails to receive k_S, the key is set to 0. In the next phase, each participant computes the value $\hat{y}_{i,S} = g^{F_{k_{i,S}}(N)}$ for each set they are a part of, along with broadcasting a *commitment* $h_{i,S} = H(\hat{y}_{i,S})$ using a hash function. In the next phase, we create a new set $\mathcal{I}$, where all sets with honest participants are added. Each participant $P_i \in S \in \mathcal{I}$ broadcasts $\hat{y}_{i,S}$. Each participant set the value $\hat{y}_S = \hat{y}_{i,S}$, for valid commitments. According to Pseudorandom Secret Sharing, similar to SSS, each participant computes it's share as $\sigma_i = \sum_{S\in\mathcal{I}:i\in S} F_{k_{i,S}}(N) \cdot Z_S(i)$, where Z_S is a t-degree polynomial. We compute the public key as $y = \prod_{S\in\mathcal{I}} \hat{y}_S$, and the commitments $y_j = \prod_{S\in I} \hat{y}_S^{Z_S(j)}$.

4.4 Fully-Secure DLog-Based DKG by Genarro

In this section we will present a DKG in the honest majority by Genarro et al. [GJKR99]. The protocol begins with each participant acting as a dealer in PVSS. Following this sharing phase, all parties proceed to verify the shares they receive from one another. If the verification fails they are forced to reveal their shares $\sigma_{ij}, \hat{\sigma}_{ij}$. We

DKG^n_t

Pre-Protocol :

Let $p = 3072$-bit prime

Let $q = \dfrac{p-1}{2}$

Let (g, h) be generators for group G_p

Round 1 For each $S \in \mathbb{S}_{n-t,n}$:

1 : the lowest index party in S chooses:

2 : $k_S \leftarrow \{0,1\}^\lambda$ and sends the value to each participant $P_j, j \in S$.

3 : $k_{j,S}$ is set to the value received by P_j

4 : if the participant did not receive any value$k_{j,S} = 0$.

Round 2 For each $S \in \mathbb{S}_{n-t,n}$, with $i \in S$:

5 : $\hat{y}_{i,S} = g^{F_{k_{i,S}}(N)}$

6 : $h_{i,S} = H(\hat{y}_{i,S})$

7 : Broadcast the value $h_{i,S}$.

8 : If some participant $P_j, j \in S$ fails to broadcast $h_{j,S}$, set $h_{j,S} = \varnothing$

Round 3 For each $S \in \mathbb{S}_{n-t,n}$, with $i \in S$:

9 : Initialize the set $\mathcal{I}$

10 : If there exists h_s s.t $h_{j,S} = h_S$ For all $P_j, j \in S$ Add S to $\mathcal{I}$

11 : Broadcast $\hat{y}_{i,S}$

Output

12 : For each $S \in \mathcal{I}$, if some participant broadcasted $\hat{y}_{i,S}$, where $H(\hat{y}_{i,S}) = h_S : \hat{y}_S = \hat{y}_{i,S}$.

13 : $\sigma_i = \sum_{S \in \mathcal{I} : i \in S} F_{K_{i,S}}(N) \cdot Z_S(i)$

14 : $y_j = \prod_{S \in \mathcal{I} : j \in S} \hat{y}_S^{Z_S(j)}$, For $j \in n$

15 : $y = \prod_{S \in \mathcal{I}} \hat{y}_S$

16 : Output $(y, \sigma_i, (y_1, \ldots, y_n))$

Figure 4.1: 3-Round Fully-Secure DKG scheme by Katz

build a set for honest participants $\mathcal{I}$. For said honest participants we generate their share of the secret as $x_i = \sum \sigma_{ij}$. Next, the public key y is extracted using FVSS. Each participant broadcast the value $g^{a_{ik}}$. If the verification of the FVSS share fails, despite passing the PVSS, we can recover their share using lagrange interpolation. Finally, we can compute the key as $y = \prod_{i \in \mathcal{I}} y_i$, where $y_i = g^{x_i}$.

4.5 Unbiased Dishonest-Majority DKG^n_{n-1} Construction

In this section we present a DKG construction $\prod^{n-1,n}_{disDKG}$ which is strongly inspired by the two fully secure DKG constructions [Kat24, Figure 4] and [Kat24, Figure 5] by Katz. As such we are hesitant to call it a new construction, though it does stand that the construction has no formal proof of security yet. Unlike the DKGs

DKG_t^n

Pre-Protocol :

Let $p = 3072$-bit prime

Let $q = \dfrac{p-1}{2}$

Let (g, h) be generators for group G_p

Round 1(for all P_i, with $i \in n$) :

1 : Generate two polynomials according to PVSS , $f_i(x), \hat{f}_i(x)$ of degree t, on the form

2 : $f_i(x) = a_{i0} + a_{i1}x + \cdots + a_{it}x^t, \hat{f}_i(x) = \hat{a}_{i0} + \hat{a}_{i1}x + \cdots + \hat{a}_{it}x^t$

3 : Broadcast the commitments $C_{ik} = g^{a_{ik}} h^{\hat{a}_{ik}} \mod p$

4 : Compute and send the shares to each respective participant P_j for j $\in n$

5 : $\sigma_{ij} = f_i(j), \hat{\sigma}_{ij} = \hat{f}_i(j)$

Round 2(for all P_j, with $j \in n$) :

6 : Verify the share received by each Participant P_i

7 : If $g^{\sigma_{ij}} h^{\hat{\sigma}_{ij}} \neq \Pi_{k=0}^{t} (C_{ik})^{j^k} \mod p$

8 : Broadcast (Complaint, i)

Round 3(for all P_i, with $i \in n$) :

9 : If (Complaint, i)

10 : Broadcast $(\sigma_{ij}, \hat{\sigma}_{ij})$

Round 4(for all P_i, with $i \in n$) :

11 : If P_i received fewer than t complaints or broadcasted valid shares

12 : $\mathcal{I}.insert(P_i)$

13 : $x_i = \Sigma_{j \in \mathcal{I}} \sigma_{ij} \mod q$

14 : $\hat{x}_i = \Sigma_{j \in \mathcal{I}} \hat{\sigma}_{ij} \mod q$

Round 5(for all P_i, with $i \in \mathcal{I}$) :

15 : Broadcast the value $g^{a_{ik}}$ for $k = 0, \ldots, t$

Round 6(for all P_j, with $j \in n$) :

16 : Verify the values broadcasted by the other participants

17 : If $g^{\sigma_{ij}} \neq \Pi_{k=0}^{t} (g^{a_{ik}})^{j^k} \mod p$

18 : Broadcast (Complaint, i)

19 : If P_i received complaint:

20 : Reconstruct $f_i(x)$ using Lagrange interpolation on the shares σ_{ij}

21 : $z_i = f_i(0)$

22 : Recompute $A_{ik} = g^{a_{ik}} \mod p$

Output(for all P_j, with $j \in n$) :

23 : For each $P_i \in \mathcal{I}$

24 : $y_i = g^{z_i} \mod p$

25 : $y = \prod_{i \in \mathcal{I}} y_i \mod p$

Figure 4.2: 6-round Fully-Secure DKG scheme by Genarro

by Katz this construction is set in the dishonest-majority setting, which means foregoing robustness, and therefore also full security, in favor of needing to trust fewer participants. Additionally the construction does not include a Verifiable Secret Sharing (VSS) scheme either, meaning the threshold is currently locked at $t = n - 1$. This makes the construction simpler both to implement and to prove secure, but it also means there is no redundancy in scheme for the case where a participant suddenly becomes unavailable.

4.5.1 Requirements of a Secure Dishonest-Majority Setting DKG

Because definitions 4.1, 4.2, and 4.3 provided earlier are mostly incompatible with the dishonest-majority setting, we provide a new set of definitions here.

Definition 4.4. Secure Dishonest-Majority Setting DKG We say that a $DKG^{t,n}$ is secure in the dishonest-majority setting if it exhibits the following properties as long as $t < n$.

1. **Correctness**: All honest participants always define the same combined secret key x along with the same combined public key $y = g^x$.

2. **Unbiased**: Combined secret key x is uniformly distributed in $\mathbb{Z}_q$, meaning y is uniformly distributed in $\mathbb{G}$.

3. **Security with abort**: If any inconsistencies arise during protocol execution, all honest participants immediately instruct other participants to abort before aborting themselves.

4.5.2 Construction

To build this construction we make use of the VRF scheme AB-VRF defined in 3.15, which furthermore relies on both the Naor-Reingold PRF defined in 3.13 and the non-interactive variant of the DLEQ ZKP from 3.5.3. As a result, we note that the construction requires a multiplicative group $\mathbb{G}$ for which the DDH-assumption, defined in 3.3, holds, and must be modeled in the ROM.

When describing the DKG we divide it into three phases. The first phase is called *Pre-Protocol*, and is where variables required to run the construction are initialized. Here group $\mathbb{G}$, of order q, with modulo p, and generators (g, h) are defined, along with an appropriate CHF H is chosen, and the list of participants P. We leave the setup function used to sample these variables ambiguous, as all that matters is that the function is run by a trusted third party or process, and the output is shared securely with all protocol participants.

The second phase is called the *Preprocessing* phase, and marks the beginning of the actual construction which is only run once. Here each participant first runs MasterKeyGen(n_k), as defined in 3.15, to produce a master secret key msk and a corresponding commitment mpk. The commitment is hashed and broadcast to all other participants in round one, before the proper commitment mpk is broadcast in round two. Once everyone has received an mpk from each of the other participants, they verify that mpk is a valid pre-image to the hash received from the same participant in round one. Finally, to verify that no malicious participant has sent different mpks to different honest participants, all participants broadcast all received mpk. As a result, all participants can now be sure that they received the same mpk from each other participant. As a side note, since H is modeled as a random oracle it can be assumed to have collision-resistance. As such we say that the hash of mpk is *statistically binding*, as compared to mpk proper which i *perfectly binding*, meaning we could decide to broadcast the received hashes of mpk instead of mpk to lessen data transmission.

The third phase, *Key Generation*, is run once per combined asymmetric key pair the protocol participants want to generate, and is where each participant generates their asymmetric key pair share. Using PKGen(msk, t) 3.15, each participant derives an asymmetric key pair share from their master secret key msk, based on a publicly known nonce t. Along with the key pair the function also creates a DLEQ proof 3.5.3, which proves the public key corresponds to commitment mpk, and therefore that sk corresponds to msk. After receiving a public key and corresponding proof from all other participants, each participant verifies the proof and computes the combined public key y.

4.5.3 Security Outline

To outline the security of $\prod_{disDKG}^{n-1,n}$, we define a simulator $\mathcal{S}$, with black-box access to adversary $\mathcal{A}$, as follows. Through a sequence of hybrid experiments, we show that the view of $\mathcal{A}$ interacting with simulation $\mathcal{S}_{disDKG}^{n-1,n}$ is indistinguishable from the view of $\mathcal{A}$ interacting with the real protocol $\prod_{disDKG}^{n-1,n}$. Let $expt_0$ refer to $\prod_{disDKG}^{n-1,n}$.

In $expt_1$, we modify $expt_0$ in the following way. For each party $P_i \in l_H$:

– In round 1 of the preprocessing phase, instead of running MasterKeyGen to generate (msk_i, mpk_i), instead choose $mpk_i \leftarrow\!\!\$\ \mathbb{G}^{n_k+1}$.

Because sampling a random msk_i, and from that generating mpk_i, is indistinguishable from simply sampling mpk_i randomly, $expt_0$ and $expt_1$ are indistinguishable.

In $expt_2$, we modify $expt_1$ in the following way. For each party $P_i \in l_H$:

$\prod^{n-1,n}_{disDKG}$

Pre-Protocol :

1 : $(\mathbb{G}, q, p, (g, h), H, P = \{P_i\}_{i=1}^n) \leftarrow\$\ setup(\lambda, n)$

Preprocessing - $(\forall P_i \in P)$:

Round 1:

2 : $(msk_i, mpk_i) \leftarrow \text{MasterKeyGen}(n_k)$

3 : $h_i = \text{H}(mpk_i)$

4 : Broadcast h_i

Round 2:

5 : Broadcast mpk_i

6 : For each $P_j \in P \backslash P_i$:

7 : verify $h_j == \text{H}(mpk_j)$ otherwise abort

8 : Send $mpk_{j\to i} = mpk_j$ to all $P_m \in P \backslash \{P_j, P_i\}$

9 : For each $P_j \in P \backslash P_i$:

10 : verify $mpk_j == mpk_{j\to m}, \forall P_m \in P \backslash \{P_i, P_j\}$ otherwise abort

Key Generation - $(\forall P_i \in P,$ and shared nonce $t \in \{0,1\}^{n_k})$:

Round 1:

11 : $(sk_{i,t}, pk_{i,t}, \pi_{i,t}) \leftarrow \text{PKGen}(msk_i, t)$

12 : broadcast $(pk_{i,t}, \pi_{i,t})$

Round 2:

13 : verify $\text{PKVerify}(pk_{j,t}, \pi_{j,t}) == 1, (\forall P_j \in P \backslash P_i)$ otherwise abort

14 : $y_t = \prod_{j=0}^{n} (pk_{j,t})$

15 : output $(sk_{i,t}, y_t)$

Figure 4.3: Dishonest majority secure DKG scheme

– We move the sampling of mpk_i from preprocessing round 1 to preprocessing round 2.

– In preprocessing round 1 we now only sample, and broadcast, hash h_i randomly from the codomain of H.

– In preprocessing round 2, after sampling mpk_i, we program $H(mpk_i) = h_i$.

The only way $\mathcal{A}$ can distinguish $expt_1$ and $expt_2$ is if they query $H(mpk_i)$ before it is reprogrammed in round 2 of preprocessing. Since mpk_i is sampled randomly from $\mathbb{G}^{n_k+1}$, the chances of this happening are negligible, meaning $expt_1$ is indistiguishable from $expt_2$.

In $expt_3$, we modify $expt_2$ in the following way. For each party $P_i \in l_H$:

$\mathcal{S}^{n-1,n}_{disDKG}$

Pre-Protocol :

$\mathcal{S}$ receives $(\mathbb{G}, q, p, (g, h), H, P = \{P_i\}_{i=1}^{n})$ from trusted third party.
$\mathcal{S}$ runs $\mathcal{A}$ to receive $l_C \subset P$, where $0 \leq |l_C| < n$
Let $l_H = P \setminus l_C$

Preprocessing :

Round 1:

For each $P_i \in l_H$:
  $h_i \leftarrow\$ \{0,1\}^{2\lambda}$
  Broadcast h_i
For each $P_j \in l_C$:
  Receive h_j

Round 2:

For each $P_i \in l_H$:
  $mpk_i \leftarrow\$ \mathbb{G}^{n_k+1}$
  Program $H(mpk_i) = h_i$
  Broadcast mpk_i
For each $P_j \in l_C$:
  Receive $mpk_j = \{mpk_{j\rightarrow i}\}_{\forall P_i \in l_H}$
  Verify that all $mpk_{j\rightarrow i}$ are equal, otherwise abort
  Verify $h_j == H(mpk_{j\rightarrow m}), \forall P_m \in l_H$, otherwise abort

Key Generation - (for shared nonce t) :

For each $P_i \in l_H$:
  $pk_{(i,t)} \leftarrow\$ \mathbb{G}$
  $\pi_{(i,t)} = (\pi_0, \ldots, \pi_m = (c_m \leftarrow\$ \{0,1\}^{2\lambda}, s_m \leftarrow\$ \mathbb{Z}_q))$
  Program H such that the hash of the input to π_m is equal to c_m
  Broadcast $(pk_{(i,t)}, \pi_{(i,t)})$
For each $P_j \in l_C$:
  Receive $(pk_{(j,t)}, \pi_{(j,t)})$
  Verify $\pi_{(j,t)}$, otherwise abort
Compute $y_t = \prod_{i=1}^{n} (pk_{i,t})$

Figure 4.4: Dishonest Majority secure DKG simulation

- In round 1 of the key generation phase, instead of using PKGen to generate asymmetric key pair $(sk_{i,t}, pk_{i,t})$, along with DLEQ ZKP $\pi_{i,t}$, we instead choose $pk_{i,t} \leftarrow\$ \mathbb{G}$.
- Furthermore, for each $\pi = (c, s) \in \pi_{i,t}$, we instead choose $\pi = (c \leftarrow\$ \{0,1\}^{2\lambda}, s \leftarrow\$ \mathbb{Z}_q)$, and program the hash of the inputs to π to be equal to c.

Firstly, since $pk_{i,t}$ in $expt_2$ is generated by the Naor-Reingold PRF 3.13, as long as the DDH-assumption holds in $\mathbb{G}$ it is indistinguishable from $pk_{i,t}$ generated by $expt_3$. Furthermore, $\pi_{i,t}$ generated by $expt_2$ is a list of proofs $\pi = (c, s)$, where c is the hash of the proof inputs, and s is the difference between two uniformly sampled values. As such, since H is modeled as a random oracle, we not that both (c, s) are uniformly distributed, and are as a result indistinguishable from proof π generated by $expt_3$. Finally, since the parameters (c, s) of $\pi \in \pi_{i,t}$ are sampled randomly, the chances of $\mathcal{A}$ querying H with the correct inputs before H is reprogrammed are negligible. As a result, $expt_3$ is indistinguishable from $expt_2$. It is trivial to see that $expt_3$ is indistinguishable from $\mathcal{S}^{n-1,n}_{disDKG}$.

In $\mathcal{S}^{n-1,n}_{disDKG}$, $\mathcal{S}$ first of all ensures that the commitments mpk from honest parties are all generated independently of those from the dishonest parties. Next, it ensures that all honest parties abort if $\mathcal{A}$ tries to cheat by: sending different commitments mpk to different honest parties, sending an mpk which is not the pre-image of the hash from round one of *preprocessing*, or by sending a public key pk with invalid ZKP. Finally, while dishonest participants are forced to derive their public keys from their commitment mpk, $\mathcal{S}$ samples uniform public keys, resulting in unbiased combined public keys.

Chapter 5
Discussion

In this section, we discuss the implications of introducing different DKG protocols into the current system, comparing it with the existing one-round method.

5.1 Observations on current system

As detailed in Section 2.4, the current system utilizes a simple one-round protocol for generating its distributed asymmetric key pairs, which allows an adversary to bias both the secret and the public key. Although this is not incompatible with the systems computational proofs for individual verifiability, universal verifiability, and vote secrecy, it does weaken the secrecy of the secret key, which could pose a potential attack vector that could be exploited. It would be ideal if this distributed key generation, like single-party key generation, guaranteed that the secret key was sampled from the uniform distribution.

To address this potential weakness we propose replacing the current one-round DKG schemes with an alternative scheme that guarantees generation of unbiased asymmetric key pairs. To this end we explore three different DKG constructions introduced in Chapter 4: one scheme by Genarro, one scheme by Katz, and finally a scheme we built on VRFs.

5.2 Comparative analysis

The three-round protocol by Katz introduces a fully secure DKG scheme for an honest majority. One significant advantage of such a scheme is that the keys generated would be impossible to bias by malicious adversaries. However as the protocol is based upon SSS, it requires an honest majority to function correctly. Using the four CCR components, along with the SC, we have $n = 5$ parties. For an honest majority, the protocol can support $t = \lfloor 5/2 \rfloor = 2$ malicious parties. In the context of the current trust assumptions as described in 2.3, the system needs to trust that $\frac{1}{4}$

of the CCR components and the SC component are honest. Although these trust assumptions don't line up with the requirements of the protocol. Nevertheless, if Swiss Post were to adjust its trust assumptions, a system based on SSS could offer notable advantages. The threshold approach in SSS could provide redundancy to the system. For instance, if the system did not require all components to be online, key generation would still be possible using only t components. In contrast, the current system would not be operational when generating the keys if any of the CCR components were to be offline. Katz's protocol stands out for its efficiency, offering a fully secure DKG in just three round while not introducing many new cryptographic primitives. It is based on PRSS and utilizes hash functions as a commitment scheme.

Similarily, the DKG created proposed by Genarro et al. also requires an honest majority. However, Genarro's DKG is based on different cryptographic primitives, specifically PVSS and FVSS, providing secrecy under the DL assumption. This protocol, however is slower and requires six rounds of communication to generate the keys. However, as the keys are generated during the setup phase, it can be argued how important the runtime of the protocol is.

Finally, the DKG construction using VRFs offers a alternative approach. It employs additive secret sharing and VRFs to collectively generate the keys, supporting a dishonest majority. This method requires collaboration among $t = n-1$ components generate the key pairs, which means that no redundancy is possible in the system. Nevertheless, this solution aligns better with Swiss Post's requirements as to ensure secrecy, all components must collaborate in order to reconstruct the key and for the system, and only that one of the CCR or the SC component must be honest. Implementing such a protocol would be more straightforward compared to the other protocols, as it relies on additive security. The AB-VRF, would also be possible to implement, as they already have knowledge with NIZK proofs, and the simplicity of the Naor-Reingold PRF.

5.3 Concluding Remarks

We have presented multiple approaches to improving the distributed generation of asymmetric key pairs in the Swiss Post e-voting system. Each approach comes with its share of benefits and drawbacks, and entails additional costs in order to implement the changes. In conclusion, the improvements will resolve an aspect of the challenges presented by Measure A.5 and improve the security of the Swiss Post E-Voting System, and should thus be implemented.

References

[BBS03] M. Bellare, A. Boldyreva, and J. Staddon, "Randomness re-use in multi-recipient encryption schemeas", in *Public Key Cryptography*, 2003, pp. 85–99.

[BG93] M. Bellare and O. Goldreich, "On defining proofs of knowledge", in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 390–420.

[BR93] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols", in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, New York, NY, USA: Association for Computing Machinery, 1993, pp. 62–73. [Online]. Available: https://doi.org/10.1145/168588.168596.

[Che06] J. H. Cheon, "Security analysis of the strong diffie-hellman problem", in *EUROCRYPT*, 2006, pp. 1–11.

[CP93] D. Chaum and T. P. Pedersen, "Wallet databases with observers", in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105.

[Cra96] R. Cramer, *Modular Design of Secure yet Practical Cryptographic Protocols*. University of Amsterdam, 1996.

[DH76] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. [Online]. Available: https://doi.org/10.1109/TIT.1976.1055638.

[ECS+15] A. Everspaugh, R. Chatterjee, *et al.*, *The pythia prf service*, Cryptology ePrint Archive, Paper 2015/644, 2015. [Online]. Available: https://eprint.iacr.org/2015/644.

[Elg85] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[Fel87] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing", in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, 1987, pp. 427–438.

[FS87] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions", *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986. [Online]. Available: https://doi.org/10.1145/6490.6503.

[GJKR99] R. Gennaro, S. Jarecki, *et al.*, "Secure distributed key generation for discrete-log based cryptosystems", in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 295–310.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems", *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989. [Online]. Available: https://doi.org/10.1137/0218012.

[HIJ+21] S. Huang, S. Iyengar, *et al.*, "Dit: De-identified authenticated telemetry at scale", 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233480867.

[Kat24] J. Katz, "Round-optimal fully secure distributed key generation", IACR Crypto, Springer-Verlag, 2024.

[Kre18] E. Kreyszig, *Advanced Engineering Mathematics*, 10th. 2018, ch. 19.

[MRV99] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions", in *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, 1999, pp. 120–130.

[Nat15] U. Nations, *THE 17 GOALS | Sustainable Development*, 2015. [Online]. Available: https://sdgs.un.org/goals.

[NR04] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions", *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004. [Online]. Available: https://doi.org/10.1145/972639.972643.

[Ped92] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing", in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[Sha79] A. Shamir, "How to share a secret", 1979. [Online]. Available: https://doi.org/10.1145/359168.359176.

[Smy18] B. Smyth, *A foundation for secret, verifiable elections*, Cryptology ePrint Archive, Paper 2018/225, 2018. [Online]. Available: https://eprint.iacr.org/2018/225.

[Swi20] Swiss Post. "Redesign and relaunch of trials". (2020), [Online]. Available: ttps://www.bk.admin.ch/dam/bk/en/dokumente/pore/Final%20report%20SC%20VE_November%202020.pdf.download.pdf/Final%20report%20SC%20VE_November%202020.pdf.

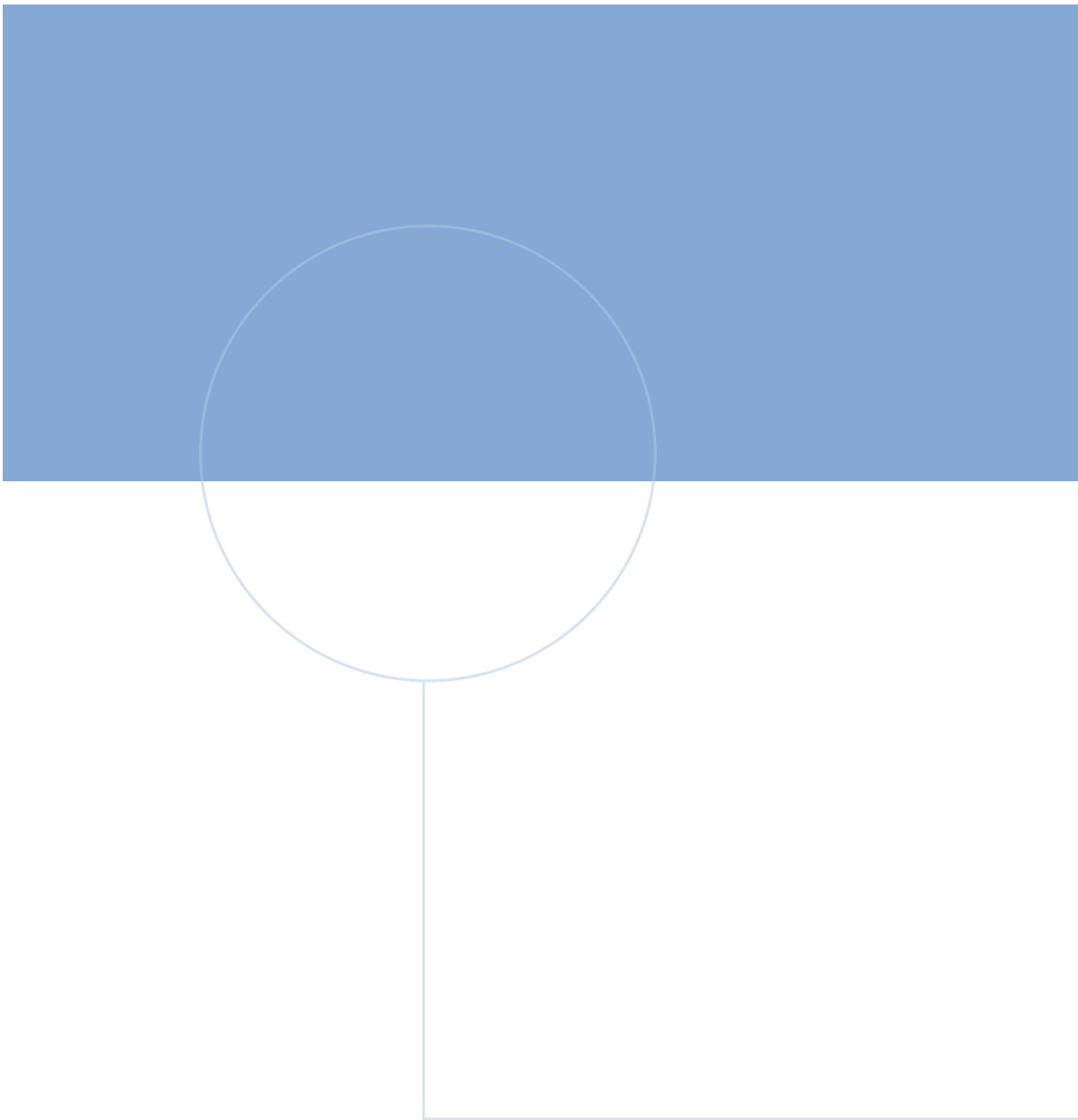
[Swi24a] Swiss Post. “E-voting architecture document. version 1.4.0”. (2024), [Online]. Available: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/System/SwissPost_Voting_System_architecture_document.pdf.

[Swi24b] Swiss Post. “Protocol of the swiss post voting system. computational proof of complete verifiability and privacy. version 1.3.0”. (2024), [Online]. Available: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/Protocol/Swiss_Post_Voting_Protocol_Computational_proof.pdf.

[Swi24c] Swiss Post. “Swiss post voting system. system specification. version 1.4.0”. (2024), [Online]. Available: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/System/System_Specification.pdf.

[TKO+16] A. H. Trechsel, F. S. Kucherenko, *et al.*, *Potential and challenges of e-voting in the european union*, 2016. [Online]. Available: https://www.europarl.europa.eu/RegData/etudes/STUD/2016/556948/IPOL_STU(2016)556948_EN.pdf.

[ZSS04] F. Zhang, R. Safavi-Naini, and W. Susilo, “An efficient signature scheme from bilinear pairings and its applications”, in *Public Key Cryptography*, 2004, pp. 277–290.



NTNU

Norwegian University of
Science and Technology