Institute of Information Security

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Security Analysis of the Russian Federal Remote E-Voting Scheme

Marvin Buck

Course of Study:	Informatik
Examiner:	Prof. Dr. Ralf Küsters
Supervisor:	Julian Liedtke, M.Sc.
Commenced:	June 1, 2022
Completed:	December 28, 2022

Abstract

When developing and deploying systems for electronic voting (e-voting systems), it is fundamental to guarantee what is referred to as verifiability. Informally, the security property verifiability implies that an e-voting system enables voters and external observers to check whether the votes have actually been counted and thus have not been dropped, whether the votes have been counted correctly and thus have not been altered as well as whether the published election result is correct. Those checks should be possible even in the case that voting devices and servers have programming errors or are malicious.

The work at hand deals with the introduction of one of the two remote e-voting systems that were used in the Russian parliamentary elections of 2021, referred to as the Russian federal remote e-voting system. More precisely, we present a description of the protocol the remote e-voting system is based on and investigate the cryptographic primitives that are used in the protocol. With verifiability being a fundamental security property, we analyze the security of the Russian federal remote e-voting protocol w.r.t. verifiability. This is done under assumptions about the used cryptographic primitives and assumptions about the honesty of the protocol participants. The used notion of verifiability is formally captured by the KTV framework.

Kurzfassung

Bei Entwicklung und Einsatz von Systemen für die elektronische Stimmabgabe (E-Voting Systeme) ist es von grundlegender Bedeutung die Überprüfbarkeit, auch Verifizierbarkeit genannt, des Wahlergebnises zu gewährleisten. Vereinfacht bedeutet die Sicherheitseigenschaft Verifizierbarkeit, dass ein System zur elektronischen Stimmabgabe es den Wählern und externen Beobachtern ermöglicht, zu überprüfen, ob die Stimmen tatsächlich gezählt wurden und somit nicht entfernt wurden, ob die Stimmen korrekt gezählt wurden und somit nicht verfälscht wurden, sowie ob das veröffentlichte Wahlergebnis korrekt ist. Diese Überprüfungen sollten auch dann möglich sein, wenn die Wahlgeräte und Server Programmierfehler aufweisen oder böswillig manipuliert sind.

Die vorliegende Arbeit befasst sich mit der Einführung eines der beiden E-Voting Systeme, die bei den russischen Parlamentswahlen 2021 verwendet wurden, das russische E-Voting System für föderale Wahlen. Konkret wird das Protokoll beschrieben, auf dem dieses E-Voting System basiert und es werden die kryptographischen Primitive untersucht, die in diesem Protokoll verwendet werden. Da die Verifizierbarkeit eine grundlegende Sicherheitseigenschaft ist, analysieren wir das russische E-Voting System für föderale Wahlen hinsichtlich dieser Eigenschaft. Dies erfolgt basierend auf Annahmen über die verwendeten kryptographischen Primitive und Annahmen über die Ehrlichkeit der Protokollteilnehmer. Hierzu wird die im KTV Framework beschriebene Definition der Verifizierbarkeit verwendet.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CPA	chosen-plaintext attack. 22
EC	Elliptic Curve. 11
NIZKP	non-interactive zero-knowledge proof. 29
ZKP	zero-knowledge proof. 29
ZKRP	zero-knowledge range proof. 29

1 Introduction

Motivation

Due to the progressing digitalization of our everyday life, a lot of processes of the physical world have been transformed into the digital world, with shopping or banking transactions being one of the most prominent ones. A process which is not as widespread in the digital world as shopping or banking is voting. However, systems for electronic voting (e-voting systems) are increasingly employed for national, state-wide and municipal elections. Beyond political elections, e-voting systems are used for elections within companies, organizations, and associations [14].

A basic distinction can be made between two types of e-voting systems according to [35]. For the first type, the voter has to visit a polling station where a vote can be cast under the use of a voting machine. The second type does not require going to a polling station, but enables the voter to use her own voting device in order to cast her vote remotely over the Internet. This type of e-voting is called remote e-voting and is the type of e-voting considered in this work.

E-voting systems are complex software and hardware systems, implying that programming errors occur and are hardly avoidable and identifiable. Furthermore, deliberate manipulation of such systems is often hard or virtually impossible to detect [35]. This means that there exists a risk that neither the votes have been counted correctly nor that the votes have actually been counted. In case one of the two described scenarios occurs, the published election result does not correspond to the ideal election result, i. e., the one that is reflecting how the voters have actually voted. In order to address this concern, e-voting systems are desired to provide verifiability. Informally, the security property verifiability implies that an e-voting system enables voters and possibly external auditors to check whether the votes have actually been counted and thus have not been dropped, whether the votes have been counted correctly and thus have not been altered as well as whether the published election result is correct, even in the case that voting devices and servers have programming errors or are malicious [14]. One of the most prominent verifiable remote e-voting systems is Helios [1].

As one of the first countries, Estonia has introduced remote e-voting in 2005. Other countries as for example Norway [27] have also tried to implement remote e-voting for political elections. An overview of the state of e-voting worldwide (from 2016) can be obtained from [51], a review of the development of e-voting from [26] respectively. A newcomer in remote e-voting is Russia, allowing to cast votes via the Internet during the Moscow local elections of 2019 [50]. As the used system exhibited serious cryptographic issues, two new remote e-voting systems have been deployed in the parliamentary elections of 2021. One of them was developed to conduct e-voting in Moscow, whereas the other one was developed for e-voting in six federal districts of Russia.

This work considers the latter one, referred to as the Russian federal remote e-voting system and provides a security analysis of this system w.r.t. verifiability as defined in the KTV framework.

Contribution

In this work, we introduce the Russian federal remote e-voting scheme including a description of the protocol and protocol participants as well as an investigation of the cryptographic primitives used in the Russian e-voting system. Before the investigation, we provide formal definitions of the used cryptographic primitives and consider most of them in terms of their security requirements. Besides, we describe the KTV framework that allows to formally capture the notion of verifiability.

Finally, we perform the security analysis of the Russian federal remote e-voting protocol w.r.t. verifiability as defined in the KTV framework under assumptions about the used cryptographic primitives as well as assumptions about the honesty of the protocol participants.

Structure

This work starts with introducing the fundamentals in Chapter 3, necessary for the consideration of the Russian federal remote e-voting scheme as well as for the investigation of the used cryptographic primitives. In Chapter 4, we present a high level view on the process of e-voting. Next, we consider the KTV framework in Chapter 5 laying the foundation for the security analysis of the Russian e-voting scheme w.r.t. verifiability. This is followed by Chapter 6 which is concerned with the Russian e-voting scheme and investigating the cryptographic primitives that have been used in the e-voting scheme. Chapter 7 deals with performing the security analysis w.r.t verifiability. Finally, Chapter 8 provides a summary as well as an outlook on future work regarding the Russian federal remote e-voting system.

2 Related Work

The Russian federal remote e-voting scheme to be analyzed was deployed for the first time in the Russian parliamentary elections of 2021. The official full version of this scheme has never been published by the election organisers. Since all the available information has been in Russian, the scheme was not accessible for the international community. In [50], Vakarjuk et al. have been the first to put information from different public sources together and made this information available in English. This enabled the international community to access the scheme and thus allowing for further studies. Until now, no further work about the description of the Russian federal remote e-voting scheme has been published. Additionally, Vakarjuk et al. provided an initial high-level analysis. Other than that, there has not been any sort of analysis yet. Consequently, this work is the first to formally analyze this scheme w.r.t verifiability as defined by Küsters et al. in [14].

3 Fundamentals

In this chapter, we first present some mathematical basics that will be necessary throughout this work. This is followed by an introduction of the fundamentals of the cryptographic primitives used in the Russian federal remote e-voting scheme. Most of the primitives are formally defined as well as considered in terms of their security requirements whereas others are viewed on a high level. This lays the basis for understanding the used primitives and allows us to take a closer look at them in Section 6.4.

3.1 Mathematical Fundamentals

This section is concerned with introducing the mathematical fundamentals, more precisely with some characteristics of functions as well as elliptic curves, that will be necessary throughout this work. The introduced characteristics of functions will be elementary for defining the security of most of the cryptographic primitives used in the Russian federal remote e-voting scheme, but also essential for formally capturing the notion of verifiability in Section 5.2. On the other hand, elliptic curves will play an important role in the utilized public-key encryption scheme.

Characteristics of Functions

In [14], Küsters et al. provide the subsequent definitions characterizing functions.

Negligible A function $f\colon \mathbb{N} \to [0,1]$ is negligible if the following condition holds:

$$\forall c > 0\ \exists l_0\ \forall l > l_0\colon f(l) \leq \frac{1}{l^c}$$

Overwhelming A function $f\colon \mathbb{N} \to [0,1]$ is overwhelming if $1 - f$ is negligible.

δ-bounded A function $f\colon \mathbb{N} \to [0,1]$ is δ-bounded if the following condition holds:

$$\forall c > 0\ \exists l_0\ \forall l > l_0\colon f(l) \leq \delta + \frac{1}{l^c}$$

Elliptic Curves

Essentially, an elliptic curve can be defined over arbitrary fields. According to [43] the elliptic curves used in cryptography are however mostly defined over finite fields. In [37], elliptic curves are defined as follows. Let $p > 3$ be prime. An elliptic curve over F_p[1] is the set of points (x, y) with $x, y \in F_p$ which satisfy the congruence

$$y^2 \equiv x^3 + ax + b \bmod p$$

together with a single element O called the point at infinity. The constants $a, b \in F_p$ have to fulfill $4a^3 + 27b^2 \neq 0 \pmod{p}$. The elliptic curve over the finite field F_p could be written as $E_p(a, b)$. The points on $E_p(a, b)$ form an (additive) abelian group where O serves as the identity element. Further information on elliptic curves can be found in [32] or [33].

3.2 Cryptographic Hash Functions

Cryptographic hash functions can be used in order to achieve integrity. That is to make sure that a message was received exactly as it was sent by the sender. Furthermore, they are utilized as crucial building block for message authentication codes as well as for digital signatures among other things. Informally, a hash function maps inputs of arbitrary length to outputs of fixed length. Subsequently, we formally define a hash function and look at different security properties characterizing hash functions [30, 47].

Definition

Formally, a function $H\colon D \longrightarrow R$, where $D = \{0, 1\}^*$ is the domain and $R = \{0, 1\}^l$ is the range (for $l \geq 1$), is called a hash function.

Security Properties

The main requirement to hash functions is the avoidance of collisions. A collision occurs if H maps two different inputs to the same output. We are exclusively interested in hash functions with $|D| > |R|$. Thus, collisions definitely exist. However, it should be as hard as possible to find them.

There are different types of hash functions meeting different requirements in terms of security.

- Collision resistance: Collision resistant hash functions guarantee that it is computationally infeasible to find a pair x, x' such that $x \neq x'$ and $H(x) = H(x')$.
- Second-preimage resistance[2]: Second-preimage resistant hash functions guarantee that given x, it is computationally infeasible to find x' such that $x \neq x'$ and $H(x) = H(x')$.

[1] $F_p = \{0, \ldots, p - 1\}$

[2] Second-preimage resistance is also referred to as target-collision resistance.

- Preimage resistance: Preimage resistant hash functions guarantee that given a hash v, it is computationally infeasible to find x such that $H(x) = v$.

Those security properties of hash functions H as defined above yield the following implications:

$$\forall H\colon H \text{ collision resistant } \implies H \text{ second preimage resistant} \implies H \text{ preimage resistant}$$

3.3 Public-Key Encryption Schemes

We imagine a scenario in which party A wants to send some message m in encrypted form to party B who wants to decrypt this message in order to know what A wants to send. Encryption as well as decryption of messages is enabled by encryption schemes. A distinction is made between public-key encryption and private-key encryption. For private-key encryption, there is only a single key which is used for both, encryption and decryption. In contrast, public-key encryption is based on two keys, where one is used for encryption and the other one is used for decryption. Thereby, public-key encryption allows for private communication between parties without the necessity to agree on any secret information in advance as stated in [30]. In the following, based on [30], we formalize public-key encryption and introduce the circumstances for a public-key encryption scheme to be considered secure.

Definition

A public-key encryption scheme is defined as a triple of probabilistic polynomial-time algorithms (Gen, Enc, Dec) such that:

1. The key-generation algorithm Gen takes the security parameter[3] 1^n as input and outputs a pair of keys (pk, sk). The first key pk is referred to as the public key and the second one sk to as the private key. We assume that $|pk_2|, |sk_2| \geq n$, and that n can be determined from pk_2 or sk_2.
2. The encryption algorithm Enc takes a public key pk and a message m from some message space $\mathcal{M}$[4]as input. It outputs a ciphertext c, written as $c \leftarrow \mathsf{Enc}_{pk}(m)$. (In order to achieve meaningful security, Enc needs to be probabilistic.)
3. The deterministic decryption algorithm Dec takes a private key sk and a ciphertext c as input, and outputs a message m or a special symbol $\perp$ denoting failure. This is written as $m := \mathsf{Dec}_{sk}(c)$.

[3]The security parameter is a means to specify the complexity for an adversary to break the cryptographic scheme. In case of public-key encryption schemes, n denotes the minimum length of pk_2 and sk_2. Thus, there are at least 2^n possible private keys that the adversary has to try in a brute force approach in order to get the actual sk and by this break the public-key encryption scheme.

[4]$\mathcal{M}$ may depend on pk.

In addition, the definition includes the requirement that, except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^n)$, it holds that

$$\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$$

for any (legal) message m.

Security Definition

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme and $\mathcal{A}$ an adversary, consider the eavesdropping indistinguishability experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).
2. $\mathcal{A}$ is given the public key pk[5], and outputs a pair of messages $m_0, m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$.
3. A uniform bit $b \in \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ is computed and given to $\mathcal{A}$. The ciphertext c is called challenge ciphertext.
4. $\mathcal{A}$ outputs a bit b'. The output of the experiment is 1 if $b' = b$, and 0 otherwise. In case $b' = b$, $\mathcal{A}$ succeeds.

A public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has indistinguishable encryptions under a chosen-plaintext attack (CPA) (or is CPA-secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$Pr\left[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n).$$

3.4 Homomorphic Encryption

Homomorphic encryption is a property of an encryption scheme. Intuitively, homomorphic encryption schemes allow making (particular) computations performed on the encrypted ciphertexts, yielding a single ciphertext that contains the encrypted result. Subsequently, we formalize the homomorphic property based on [30] and present a use case of homomorphic encryption which is vote tallying as described below.

Definition

A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is homomorphic if for all n and all pairs of keys (pk, sk) output by $\mathsf{Gen}(1^n)$, it is possible to define groups $(\mathcal{M}, \gamma)$, $(\mathcal{C}, \tau)$ (depending on pk only) such that[6]:

- The message space is $\mathcal{M}$, and all ciphertexts output by Enc_{pk} are elements of $\mathcal{C}$.

[5]The adversary can use pk to obtain ciphertexts for arbitrary plaintexts.

[6]Note that γ and τ are mappings $\gamma\colon \mathcal{M} \times \mathcal{M} \to \mathcal{M}, (x, y) \mapsto \gamma(x, y)$ and $\tau\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}, (x, y) \mapsto \tau(x, y)$ such that associativity holds, a neutral element exists and every $x \in \mathcal{M}, \mathcal{C}$ has an inverse element.

- For any $m_1, m_2 \in \mathcal{M}$ and any $c_1, c_2 \in C$ with $c_1 \leftarrow \mathsf{Enc}_{pk}(m_1)$ and $c_2 \leftarrow \mathsf{Enc}_{pk}(m_2)$, it holds that

  $$\mathsf{Dec}_{sk}(\tau(c_1, c_2)) = \gamma(m_1, m_2).$$

  Moreover, the distribution on ciphertexts obtained by encrypting m_1, encrypting m_2, and then computing $\tau(c_1, c_2)$ is identical to the distribution on ciphertexts obtained by encrypting $\gamma(m_1, m_2)$.

The last part of the definition ensures that if ciphertexts c_1 output by $\mathsf{Enc}_{pk}(m_1)$ and c_2 output by $\mathsf{Enc}_{pk}(m_2)$ are generated and the result $c_3 := \tau(c_1, c_2)$ is computed, then the resulting ciphertext c_3 does not contain more information about m_1 or m_2 than $m_3 := \gamma(m_1, m_2)$.

Homomorphic Vote Tallying

For homomorphic vote tallying, we need a public-key encryption scheme offering (usually) an additive or multiplicative homomorphic operation [38]. Additive homomorphic vote tallying is visualized in Figure 3.1. There is a set of voters $v_1, \ldots, v_n$ where every voter v_i casts a ballot C_{v_i}, which is an encryption of the plaintext vote M_i. After having received all ballots, the polling station aggregates them by their summation, yielding a single ciphertext T. The decryption of T give us the sum of all votes $M_1 + \cdots + M_n$.

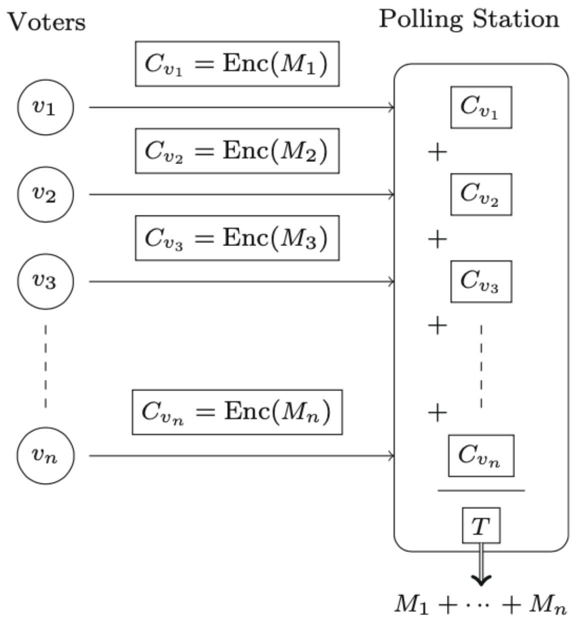
Voters
Polling Station
$C_{v_1} = \mathrm{Enc}(M_1)$
$C_{v_2} = \mathrm{Enc}(M_2)$
$C_{v_3} = \mathrm{Enc}(M_3)$
$C_{v_n} = \mathrm{Enc}(M_n)$
v_1
v_2
v_3
v_n
C_{v_1}
C_{v_2}
C_{v_3}
C_{v_n}
T
$M_1 + \cdots + M_n$

Figure 3.1: Homomorphic Vote Tallying [38]

3.5 Digital Signature Schemes

A digital signature scheme is a cryptographic primitive that allows a signer with public key pk and private key sk to sign a message m using sk such that every party with access to pk is able to verify that m originated from the signer and has not been altered en route. Note that the party also needs to know that the public key pk was actually established by the signer. Thus, digital signatures provide integrity in the public-key setting. In the following as outlined in [30], we formalize digital signatures and introduce the circumstances for a digital signature scheme to be considered secure.

Definition

A digital signature scheme is defined as a triple of probabilistic polynomial-time algorithms (Gen, Sign, Vrfy) such that:

1. The key-generation algorithm Gen takes the security parameter 1^n as input and outputs a pair of keys (pk, sk). The first key pk is referred to as public key and the second one sk to as private key. We assume that $|pk_2|, |sk_2| \geq n$, and that n can be determined from pk_2 or sk_2.
2. The signing algorithm Sign takes a private key sk and a message m from some message space $\mathcal{M}$[7]as input. It outputs a signature σ, written as $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$.
3. The deterministic verification algorithm Vrfy takes a public key pk, a message m, and a signature σ as input. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. This is written as $b := \mathsf{Vrfy}_{pk}(m, \sigma)$.

In addition, the definition includes the requirement that, except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^n)$, it holds that

$$\mathsf{Vrfy}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$$

for every (legal) message m. A signature σ is called valid on a message m if $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

Security Definition

Given a digital signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ and an adversary $\mathcal{A}$, consider the signature experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).
2. $\mathcal{A}$ is given the public key pk as well as access to an oracle $\mathsf{Sign}_{sk}(\cdot)$. The adversary then outputs (m, σ). Let Q denote the set of all queries that $\mathcal{A}$ has asked its oracle.
3. The output of the experiment is 1 if and only if (1) $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ and (2) $m \notin Q$. In this case, we say that $\mathcal{A}$ succeeds.

A digital signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is existentially unforgeable under an adaptive chosen-message attack (or is EUF-CMA-secure), if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function negl such that

$$Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

[7]$\mathcal{M}$ may depend on pk.

3.6 Blind Digital Signature Schemes

Blind digital signatures, proposed by Chaum in [9] are as the name suggests quite similar to digital signatures in the sense that blind digital signatures allow signing of messages as well as verifying the corresponding signatures. However, there is a small but significant difference. In case of blind digital signatures, the receiver of the signature chooses the message m to be signed. In order to obtain a signature for the selected message m, she blinds m resulting in another message m' such that the information of the original message m does not leak to the signer. The signer with public key pk then signs m' and provides the receiver with the appropriate signature σ'. Afterwards, the receiver unblinds the signature σ' such that the resulting signature σ (called blind signature) is valid for m with respect to pk. Thus, the receiver neither reveals any information about the message m nor about the corresponding signature σ. Consequently, blind digital signatures allow to ensure the anonymity of protocol participants [48]. In the following, as presented by Fischlin and Schröder in [25], we formalize blind digital signatures and introduce the circumstances for a blind digital signature scheme to be considered secure.

Notation

In order to formally define blind digital signatures according to Fischlin and Schröder, we need to introduce the subsequent notation for interactive executions between algorithms $\mathcal{X}$ and $\mathcal{Y}$ first. Let $\mathcal{X}$ and $\mathcal{Y}$ be two algorithms with private input x and y correspondingly. Let a and b be the private output of $\mathcal{X}$ and $\mathcal{Y}$ correspondingly. Then, the joint execution of the algorithms $\mathcal{X}$ and $\mathcal{Y}$ is denoted by $(a, b) \leftarrow \langle \mathcal{X}(x), \mathcal{Y}(y) \rangle$. Moreover, we need additional notation for defining security. $\mathcal{Y}^{\langle \mathcal{X}(x), \cdot \rangle^{\infty}}(y)$ denotes that $\mathcal{Y}$ can invoke an infinite number of executions of the interactive protocol with $\mathcal{X}$ where the order can be arbitrarily interleaved. $\mathcal{X}^{\langle \cdot, \mathcal{Y}(y_0) \rangle^1, \langle \cdot, \mathcal{Y}(y_1) \rangle^1}(x)$ denotes that $\mathcal{X}$ can invoke arbitrarily ordered executions with $\mathcal{Y}(y_0)$ and $\mathcal{Y}(y_1)$. However, the interaction with each algorithm is limited to 1, i. e., one interaction with $\mathcal{Y}(y_0)$ as well as one interaction with $\mathcal{Y}(y_1)$.

Definition

A blind digital signature scheme consists of three probabilistic polynomial-time algorithms (Gen, $\langle \mathcal{S}, \mathcal{U} \rangle$, Vrfy) such that:

1. The key-generation algorithm Gen takes the security parameter 1^n as input and outputs a pair of keys (pk, sk). The first key pk is referred to as the public key and the second one sk to as the private key. We assume that $|pk_2|, |sk_2| \geq n$, and that n can be determined from pk_2 or sk_2.
2. The signing algorithm $\langle \mathcal{S}, \mathcal{U} \rangle$ is the joint execution of the algorithm $\mathcal{S}(sk)$ and the algorithm $\mathcal{U}(pk, m)$ for a message $m \in \{0, 1\}^n$. It outputs σ of the user (and some possibly empty output λ for the signer). This is written as $(\lambda, \sigma) \leftarrow \langle \mathcal{S}(sk), \mathcal{U}(pk, m) \rangle$.
3. The deterministic verification algorithm Vrfy takes a public key pk, a message m, and a (blind) signature σ as input. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this as $b := \mathsf{Vrfy}_{pk}(m, \sigma)$.

In addition, the definition includes the requirement that, except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^n)$, it holds that

$$\mathsf{Vrfy}_{pk}(m, \sigma) = 1$$

for every message $m \in \{0, 1\}^n$ and every σ output by $\mathcal{U}$ in the joint execution of the algorithms $\mathcal{S}(sk)$ and $\mathcal{U}(pk, m)$. A (blind) signature σ is called valid on a message m if $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

Security Definition

The security of blind digital signature schemes is defined by two properties called unforgeability and blindness. An adversary $\mathcal{U}^*$, trying to break the unforgeability of the scheme, attempts to produce $k + 1$ valid message-signature pairs after a maximum of k completed interactions with the honest signer. The number of interactions is not fixed up front, but is adaptively determined by the adversary $\mathcal{U}^*$ during the attack. In order to identify completed interactions, the honest signer is assumed to return a special symbol ok after the final protocol message has been sent. This should indicate a completed interaction from the point of view of the honest signer. In terms of the blindness property, we consider two interactions with an honest user $\mathcal{U}$. Now blindness describes that it should be infeasible for a malicious signer $\mathcal{S}^*$ to determine which of two messages m_0 and m_1 has been signed first. In case that one of the two interactions has returned $\perp$, the signer is not provided with information about the other signature either.

Given a blind digital signature scheme Π = (Gen, $\langle \mathcal{S}, \mathcal{U} \rangle$, Vrfy), consider the following two experiments formally defining the unforgeability as well as the blindness property.

The unforgeability experiment $\mathsf{Unforge}_{\mathcal{U}^*,\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).
2. $((m_1, \sigma_1), \ldots, (m_{k+1}, \sigma_{k+1})) \leftarrow \mathcal{U}^{*\langle \mathcal{S}(sk), \cdot \rangle^\infty}(pk)$
3. Return 1 iff
   - $m_i \neq m_j$ for $1 \leq i < j \leq k + 1$, and
   - $\mathsf{Vrfy}_{pk}(m_i, \sigma_i) = 1$ for all $i = 1, \ldots, k + 1$, and
   - $\mathcal{S}$ has returned ok in at most k interactions.

In the following blindness experiment, $\mathcal{S}^*$ is working in modes find, issue and guess.

The blindness experiment $\mathsf{Blind}_{\mathcal{S}^*,\Pi}(n)$:

1. $(pk, m_0, m_1, \mathsf{st}_{\mathsf{find}}) \leftarrow \mathcal{S}^*(\mathsf{find}, 1^n)$
2. $b \leftarrow \{0, 1\}$
3. $\mathsf{st}_{\mathsf{issue}} \leftarrow \mathcal{S}^{*\langle \cdot, \mathcal{U}(pk, m_b) \rangle^1, \langle \cdot, \mathcal{U}(pk, m_{1-b}) \rangle^1}(\mathsf{issue}, \mathsf{st}_{\mathsf{find}})$ and let σ_b, σ_{1-b} denote the (possibly undefined) local outputs of $\mathcal{U}(pk, m_b)$ and $\mathcal{U}(pk, m_{1-b})$ respectively
4. Set $(\sigma_0, \sigma_1) = (\perp, \perp)$ if $\sigma_0 = \perp$ or $\sigma_1 = \perp$
5. $b^* \leftarrow \mathcal{S}^*(\mathsf{guess}, \sigma_0, \sigma_1, \mathsf{st}_{\mathsf{issue}})$

6. Return 1 iff $b = b^*$

A blind digital signature scheme $\Pi = (\mathsf{Gen}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Vrfy})$ is secure if for all probabilistic polynomial-time adversaries $\mathcal{U}^*$ and malicious signers $\mathcal{S}^*$, there exists a negligible function negl such that

$$Pr\left[\mathsf{Unforge}_{\mathcal{U}^*,\Pi}(n) = 1\right] \leq \mathsf{negl}(n)$$
and
$$Pr\left[\mathsf{Blind}_{\mathcal{S}^*,\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n).$$

3.7 Commitment Schemes

As illustrated in [30], a commitment scheme can be considered as a digital envelope. Sealing a message in an envelope and handing it over to another party guarantees privacy as long as the envelope has not been opened. In addition, there is a binding between the sealer later referred to as committer and the message in the envelope. Thus, commitment schemes allow to commit to a message m by sending a commitment com, preserving the following properties:

- Hiding: Commitment com does not reveal anything about m
- Binding: It is impracticable for the committer to output a commitment com that will allow her to later open it as two different messages m, m'.

  This property makes sure that com truly commits the committer to some well-defined value such that she can not pretend having committed to another value.

Definition

In [8], a non-interactive commitment scheme is defined as a pair of probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Com})$ such that:

1. The key-generation algorithm Gen takes the security parameter 1^n as input and outputs a commitment key ck. This key specifies a message space $\mathcal{M}$, a randomness space $\mathcal{R}$ and a commitment space $\mathcal{C}$.
2. The commitment algorithm Com takes ck, a message $m \in \mathcal{M}$ as well as a random value $r \in \mathcal{R}$ as input and outputs a commitment $\mathsf{com} \in \mathcal{C}$.

A sender commits to a message $m \in \mathcal{M}$ by choosing $r \in \mathcal{R}$ uniformly at random, computing the commitment $\mathsf{com} := \mathsf{Com}_{ck}(m; r)$, and sending it to a receiver. In order to later decommit com and reveal m, the sender sends m, r to the receiver who verifies the commitment by checking whether $\mathsf{Com}_{ck}(m; r) \stackrel{?}{=} \mathsf{com}$.

Security Definition

Given a commitment scheme $\Pi = (\mathsf{Gen}, \mathsf{Com})$ and an adversary $\mathcal{A}$, consider the following two experiments formally defining the hiding as well as the binding property adapted from [30].

The commitment hiding experiment $\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to obtain the commitment key ck.
2. $\mathcal{A}$ is given ck as input, and outputs a pair of messages $m_0, m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$.
3. A uniform bit $b \in \{0,1\}$ is chosen, and then a commitment $\mathsf{com} \leftarrow \mathsf{Com}_{ck}(m_b; r)$ is computed and given to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a bit b'. The output of the experiment is 1 if and only if $b' = b$. If $b' = b$ we say that $\mathcal{A}$ succeeds.

The commitment binding experiment $\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to obtain the commitment key ck.
2. $\mathcal{A}$ is given ck as input and outputs $(\mathsf{com}, m_0, r_0, m_1, r_1)$ where $m_0, m_1 \in \mathcal{M}$ and $r_0, r_1 \in \mathcal{R}$.
3. The output of the experiment is 1 if and only if $m_0 \neq m_1$ and $\mathsf{Com}_{ck}(m_0; r_0) = \mathsf{com} = \mathsf{Com}_{ck}(m_1; r_1)$.

A commitment scheme $\Pi = (\mathsf{Gen}, \mathsf{Com})$ is secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$Pr\left[\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

and

$$Pr\left[\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n) = 1\right] \leq \mathsf{negl}(n).$$

3.8 Secret Sharing

This section introduces secret sharing as outlined by Shamir in [45]. Encryption allows to protect data, but protection of the encryption key requires a different approach. The key management scheme that is most secure stores the encryption key in a single and well-guarded location as for example a computer or a human brain. However, this scheme is extremely unreliable since the encryption key can be made inaccessible by only a single misfortune such as a computer breakdown. In order to prevent that, another solution is to copy the encryption key and store it at different locations. Since this increases the risk of security breaches like computer penetration or betrayal, we still need a different solution. Efficient threshold schemes, introduced subsequently, can help to solve the problem of managing cryptographic keys.

Therefore, Shamir generalizes the cryptographic key management problem to a problem of managing a secret. The secret is some data D and manipulations of D are allowed. The goal is to split up data D into n pieces $D_1, \ldots, D_n$ such that:

(1) knowledge of any k or more D_i pieces allows to easily compute D;

(2) knowledge of any $k - 1$ or fewer D_i pieces reveals absolutely no information about D i. e., all possible values of D are equally likely.

A scheme meeting those requirements is called a (k, n) threshold scheme. A (k, n) threshold scheme with $n = 2k - 1$ gives us a very robust key management scheme. The original key can be reconstructed even in the case that $\lfloor \frac{n}{2} \rfloor = k - 1$ of the initial n pieces are destroyed. At the same time, the adversaries are not able to reconstruct the key even when they manage to expose $\lfloor \frac{n}{2} \rfloor = k - 1$ of the remaining k pieces. Choosing the parameters k and n properly enables any sufficiently large majority to reconstruct the original key and any sufficiently large minority to prevent the key from being reconstructed.

3.9 Zero-Knowledge Proofs

In the context of e-voting, ballots usually consist of encryptions of 0 and 1 representing whether the voter has voted for a certain candidate or not. It is important to make sure that no voter gets away with for example encrypting a 2, i. e., voting more than once for a single candidate. However, we want the votes to remain private. This can be accomplished by using a zero-knowledge proof (ZKP), proposed by Goldwasser et al. [29]. In the following, we introduce ZKPs on a high level as well as a special form called zero-knowledge range proof (ZKRP).

Zero-Knowledge Proofs

According to [8], a ZKP is an interaction between two parties, a prover P and a verifier V. In this interaction, P wants to convince V that some statement about secret data is true. Therefore, P does not reveal any other information than the statement itself. ZKPs are required to satisfy the following three conditions:

- Completeness: In case of true statements, a prover can convince the verifier.
- Soundness: In case of false statements, a prover can not convince the verifier.
- Zero-Knowledge: The verifier will learn nothing from the interaction except that the statement is true.

ZKPs can be either interactive or non-interactive. In case of interactive ZKPs, the prover and verifier exchange many messages, whereas for the non-interactive zero-knowledge proof (NIZKP), the prover only sends a single message containing the proof to the verifier. A formal introduction can be obtained from [28] or [8] respectively.

Zero-Knowledge Range Proofs

As stated in [41], ZKRPs allow the prover to convince the verifier that some secret value lies within a certain interval range without revealing any information about the secret value other than that it is in the interval. Since zero-knowledge range proofs are a special form of ZKPs, the requirement for completeness, soundness and zero-knowledge applies here as well.

3.10 Blockchain

This section illustrates the ideas of the blockchain technology [39] as well as of smart contracts [53], both applied in the Russian federal remote e-voting system.

Blockchain

A blockchain consists of data sets which are built out of a chain of data packages. Those data packages are referred to as blocks, each block consisting of several transactions. When extending the blockchain by adding an additional block, the new block is validated by nodes of the network applying cryptographic means. Thus, the blockchain depicts a complete ledger of the transaction history. Besides the transactions, every block includes a timestamp, the hash value of the previous block as well as a nonce which is a random number used in order to verify the hash. Thereby, the integrity of the whole blockchain, starting with the first so-called genesis block, is guaranteed. Since hash values are unique and a change of a certain block in the chain would result in a change of the corresponding hash value, forgery of the content of a block can be prohibited. In order to add a block to the chain, the majority of nodes in the blockchain network have to agree on the validity of both, the transactions in a block as well as the block itself. Therefore, they use a consensus mechanism. After extending the blockchain with a new block, it is not possible to change the added information.

Smart Contracts

Smart contracts can be considered as a computerized transaction protocol enforcing the contractual terms of an agreement. Those contractual terms integrated in smart contracts are automatically executed in case that certain, predefined conditions are met and do not require the intervention of a trusted third party. Thus, they ensure proper contract enforcement. Since smart contracts are implemented on top of blockchains, blockchains are enabling them. In order to realize the automatic execution, contractual terms are transformed into executable computer programs that also allow to preserve their logical relations. The execution of every contract clause is logged in form of an immutable transaction which is also stored in the blockchain.

4 E-Voting

This chapter based on [14] aims to present a high level view on the e-voting process and serves as preliminaries for the consideration of the Russian federal remote e-voting scheme in Section 6.3. Additionally, it introduces notation that will be used in Chapter 5.

In an e-voting system, each of the n eligible voter can use some voting device (VD) such as a computer in order to participant in the election. Therefore, she computes a ballot and casts it. The ballot contains the voter's choice in an encrypted form. Usually the cast ballots are put on a bulletin board. Tellers or voting authorities can collect the ballots from the bulletin board and tally them. In case of modern e-voting protocols, some of them perform the vote tallying by combining all ballots into one and then decrypting the resulting ballot. In this scenario, ballots are combined using homomorphic encryption (see Section 3.4).

At the beginning of an election, the voting authorities produce the election parameters. Commonly, those parameters contain keys and a set of valid choices C. Generally, C can be an arbitrary set. In case that voters can choose only one candidate, C would just be the set of candidates, whereas in case that voters can choose multiple candidates, C would contain tuples of candidates. Note that abstention is considered to be one of the valid choices in C. In what follows, V_i denotes the i-th voter and VD_i denotes her corresponding voting device for $1 \leq i \leq n$. In order to cast a ballot, a voter V_i selects her choice $c_i \in C$. Afterwards, she runs the voting procedure $\mathsf{Vote}(c_i)$. This may include to provide VD_i with her choice c_i. The output of $\mathsf{Vote}(c_i)$ is a ballot b_i containing c_i in encrypted form. Frequently, voters have to perform some verification procedure denoted by Verify during or at the end of an election. Running this procedure attempts to prevent/detect malicious behavior by the VDs or even the voting authorities. As an example, Verify could include checking that the voter's ballot actually appears on the bulletin board in order to make sure that the ballot is contained in the election result and has not been dropped. Besides this, the verification procedure may also involve executing some cryptographic tasks. Running Verify often requires a trusted device. Otherwise, malicious behavior could cause wrong results of running the verification procedure. The tellers, denoted by T_j are responsible for collecting the ballots, tallying them and putting out the election result. It is part of the result space R which is fixed for a certain election. A result function $\rho\colon C^n \to R$ is used to compute the result of the election. On input $c = (c_1, \ldots, c_n)$ representing the voters' choices, ρ outputs the result. In order to ensure that the election result will not be adjusted to someone's advantage, the result function should be specified by the election authorities before the start of an election. Note that dishonest tellers potentially attempt to manipulate the election result. That is what should be exposed by the verifiability property (see Chapter 5). Auditors/judges, who are considered to be an honest party J could verify specific information at the end or during the election to detect malicious behavior. As an example, these verifications could include checking certain zero-knowledge proofs (see Section 3.9). Usually, they are based on publicly available information exclusively. This means that they can not only be performed by auditors/judges, but can mostly be executed by any party as well. For most election protocols, the bulletin board B, mentioned above, allows to append information only. In case B is honest, it stores all the received inputs from any participant in a list, and outputs the list on request. This list usually contains public information such as public keys, the election result, voters' ballots, or zero-knowledge proofs generated by voting authorities.

5 KTV Framework

Informally, according to [14] the security property verifiability implies that an e-voting protocol enables voters and possibly external auditors to check whether the votes have actually been counted and thus have not been dropped as well as whether the published election result is correct, even in the case that voting devices and servers have programming errors or are outright malicious. In order to perform an accurate security analysis of e-voting protocols w.r.t. verifiability, a formal and precise definition capturing the notion of verifiability is required. Therefore, Küsters et al. designed a verifiability framework, the KTV framework [36]. In this chapter, as specified in [14], we present the KTV framework which is based on the computational model as introduced in Section 5.1. Additionally, the framework provides a generic definition of verifiability which can be instantiated in different ways. That is why this definition, illustrated in Section 5.2 basically can be applied to any kind of protocol. It allows to deal with different trust assumptions and to cover various kinds of verifiability. Section 5.3 describes approaches for specifying a goal which plays a central role in the definition of verifiability. These approaches are utilized in Section 5.4 for instantiating the generic definition of verifiability.

5.1 Computational Model

This section deals with the computational model the KTV framework is based on and is fundamental for the definition of verifiability in Section 5.2. The first part of the section presents the notion of a process which is the core of the computational model. Based on a process, a protocol is defined afterwards. In the end, the meaning of a property is introduced.

Process

A process is defined as a set of probabilistic polynomial-time interactive Turing machines (ITMs), from now on referred to as programs. They are connected with each other through named channels. Two programs with a channel named the same but having contrary directions (input/output) are connected via this channel. Furthermore, a process might have channels not being connected internally. Those channels are called external channels. At any time of a process run, there exists only one active program. This program is able to send a message to another program through a channel. The program that has received the message then becomes active. After performing some computation, it may send a message to another program, and so forth. Every process, i. e., every set of programs contains a so-called master program. This is a special program being the first one to be activated. In case that the currently active program did not produce output and consequently did not activate another program, the master program is activated. Whenever the master program is active however does not produce any output, a run stops.

A process π is denoted as $\pi = p_1 \| \cdots \| p_l$, with $p_1, \ldots, p_l$ being programs. Let π_1 and π_2 be two connectible processes, i. e., two processes with external channels having the same name as well as opposite directions (input/output), then but only then $\pi_1 \| \pi_2$ is a process as well. In case those processes have internal channels with the same name, the channels are renamed since they are internal and thus not connectible externally.

A process π for which every of its programs is given the security parameter 1^l is written as $\pi^{(l)}$. In all processes, the length of a run is polynomially bounded by l. Every run can be uniquely determined by the random coins utilized by the programs in π. Each program has access to a random coin, given to the program as additional input, that allows to model randomized computation.

Protocol

A protocol P is defined as a set of agents Σ, also referred to as parties or protocol participants, as well as a program π_a for each agent $a \in \Sigma$ which is supposed to be run by the agent a. The program π_a is called the honest program of agent a. All agents are connected pairwise through channels. Additionally, each agent has a channel to the adversary. In general, one of the protocols' participants is a scheduler S acting as the master program of the protocol process. The scheduler is responsible for triggering the protocol participants as well as the adversary in the proper order. In the case of e-voting, S would trigger the agents corresponding to the election phases. As for example, in an election with phases register, vote, tally and verify, S would first trigger the agents involved in the register phase, then the agents involved in the vote phase, and so on.

Let $\pi_{a_1}, \ldots, \pi_{a_n}$ be the honest programs of the protocol participants of P. Then π_P denotes the process $\pi_{a_1} \| \cdots \| \pi_{a_n}$. Note, that an adversary A is always part of a run of the process π_P. A can run an arbitrary probabilistic polynomial-time program and is connected through channels to all agents in π_P. Thus, a run r of protocol P with adversary program π_A is a run of the process $\pi_P \| \pi_A$. The description of a run r always contains $\pi_P \| \pi_A$, such that we can identify to which process, including the adversary, r belongs to. Typically, the adversary A is able to corrupt the honest programs of the protocol participants of P by sending the message corrupt. After having received such a message and thus being corrupted, the protocol participant reveals all or some of its internal state to A and is controlled by A as of now. However, there are usually some protocol participants as for example the scheduler or a judge, that are not corruptible, meaning that they just ignore corrupt messages. In case of modeling static corruption, protocol participants might accept corrupt messages exclusively during initialization. In total, this enables great flexibility in terms of modeling various kinds of corruption such as different forms of static as well as dynamic corruption. An agent a is referred to as honest in a protocol run r if a has not accepted a corrupt message during the entire run and thus has not been corrupted. An agent a is referred to as honest if for all adversarial programs π_A, a is honest in all runs of $\pi_P \| \pi_A$. This means that the agent always ignores all corrupt messages.

Property

A property γ of P is defined as a subset of the set containing all runs of P. The complement of γ is denoted by $\neg\gamma$.

5.2 Verifiability Definition

This section is concerned with the general definition of verifiability provided by the KTV framework. In order to define verifiability, we need to introduce the judge as well as the meaning of a goal, both essential for the definition. Additionally, we formalize the notion of verifiability.

Judge

As already mentioned, the verifiability definition requires a judge J. Its role is to accept or reject a protocol run. Therefore, the judge either writes accept or reject on a (dedicated) channel decision$_J$. In order to decide whether to accept or reject a protocol run, J runs a so-called judging procedure. This procedure executes several checks, as for example the verification of all ZKPs. A protocol run is accepted by the judge if and only if all performed checks have been successful. Otherwise, J rejects the protocol run. In general, the performed checks depend on the specification of the protocol. Thus, the judging procedure should be part of that specification. Furthermore, for a protocol P, J should be included in its set of protocol participants and specified accurately. For running the judging procedure, the input of the procedure is public information exclusively. Public information usually includes all information (as well as complaints) from the bulletin board. Since solely public information is used, the judging procedure can be executed by any party. Thus, external observers and voters themselves can perform the judging procedure as well.

Goal

Besides the judge, the notion of a goal of a protocol is essential for the verifiability definition. A goal is formally defined as a property γ of the system, i. e., a set of runs. On a high level view, a goal contains those runs that are correct with respect to some protocol specification. In the case of e-voting, this would be all runs for which the published election result coincides with the voters' actual choices. The idea of the verifiability definition is that the judge J should only accept a protocol run r if the specified goal γ is satisfied ($r \in \gamma$). More specifically, the definition demands the probability (over the set containing all protocol runs) that the judge accepts the run although the goal γ is not met to be δ-bounded. Clearly, we would desire $\delta = 0$. However, for most of the e-voting protocols, $\delta = 0$ can not be achieved. One reason why this would be too strong is because usually not all voters check if their ballot(s) actually emerges on the bulletin board. This enables an adversary A to either manipulate or drop ballots without being detected.

$\Pr[\pi^{(l)} \mapsto (J\colon \text{accept})]$ denotes the probability that $\pi^{(l)}$ produces a run which is accepted by the judge J. On the other hand side, $\Pr[\pi^{(l)} \mapsto \neg\gamma, (J\colon \text{accept})]$ denotes the probability that $\pi^{(l)}$ produces a run which is not in γ but still accepted by the judge J.

Formal Definition of Verifiability

The following definition of verifiability originates from Küsters et al. [14].

Let P be a protocol with the set of agents Σ. Let $\delta \in [0, 1]$ be the tolerance, $J \in \Sigma$ be the judge and γ be a goal. Then, we say that the protocol P is (γ, δ)-verifiable by the judge J if for all adversaries π_A and $\pi = (\pi_P \| \pi_A)$, the probability

$$\Pr[\pi^{(l)} \mapsto \neg\gamma, (J\colon \text{accept})]$$

is δ-bounded as a function of l.

Strictly speaking, every protocol P with a judge always rejecting a run basically satisfies the verifiability definition. This is why it is necessary to demand a soundness or fairness condition. For example, we could require at least that a judge accepts a run in case the protocols' adversary is benign, i. e., the adversary would not corrupt parties. Formally, $\Pr[\pi^{(l)} \mapsto (J\colon \text{accept})]$ is required to be overwhelming for a benign adversary π_A. Since this property is not explicitly mentioned in most verifiability definitions in the literature, it is ignored here as well.

Whereas most definitions of verifiability are tailored to particular classes of e-voting protocols, the verifiability definition presented above is applicable to arbitrary classes of such protocols. Due to the general notion of a protocol (used in the verifiability definition) and the notion of a goal γ, the definition of verifiability from the KTV framework allows for great flexibility.

5.3 Approaches for Goals

In [14], Küsters et al. identified two approaches which are reasonable for defining a goal. This section describes those approaches that they characterize as qualitative and quantitative, respectively.

Qualitative Approach

The qualitative approach defines a goal as a set of such runs where votes of those honest voters who performed their checks successfully are definitely contained in the final election result. Providing such guarantees has the advantage that probabilities of voters performing their checks or not do not have to be taken into account, which simplifies the analysis of systems. This is due to the fact that votes of honest voters not performing their checks (successfully) can theoretically all be dropped, and the run would however still meet the goal. Thus, in order to produce a run that does not satisfy the goal, votes of honest voters successfully performing their checks (i. e., probability for performing checks successfully is 1) must not be contained in the final result. That is why analyzing systems w.r.t this approach for defining a goal is independent of these probabilities. However, considering such probabilities is important in order to measure the overall security of a system since regarding a system to be verifiable if dropping all votes of honest voters who did not (successfully) perform their checks, is critical. Given that there are checks as for example Benaloh checks [4] (performing some probabilistic checking) that might not detect manipulation with some probability, a zero tolerance level δ is generally not reasonable for this approach.

Quantitative Approach

The quantitative approach defines a goal as a set of such runs where the number of votes of honest voters being changed without anybody noticing is not bigger than some value k. In order to allow for stronger guarantees, it is advisable to distinguish between votes that have been manipulated and are thus reflected differently in the final result and votes that have been dropped and are thus not reflected in the final result at all. Compared to the qualitative approach, this approach does not provide guarantees for votes of honest voters performing their checks to be counted. Thus, in order to analyze systems w.r.t. this approach for defining a goal requires to consider the probabilities of voters performing their checks or not.

In the qualitative as well as the quantitative approach, the votes of dishonest voters are restricted to be counted at most once (i. e., no ballot stuffing). In the following, these two approaches will be formalized.

5.4 Exemplified Instantiation

In this section, we consider two instantiations of the KTV framework proposed by Küsters et al. One of them pursues the qualitative and the other one the quantitative approach. In order to instantiate the framework, we only have to define a goal γ that a protocol is expected to guarantee. For a given goal γ, we should always aim to achieve a δ which is as small as possible. This means that the value of δ is not fixed in advance, but rather the result of an analysis of a concrete e-voting system. In the following, we define the qualitative goal $\gamma_{ql}(\varphi)$ as well as the quantitative goal $\gamma_{qn}(k, \varphi)$. The parameter φ allows to describe the trust assumptions under which the protocol is expected to provide certain guarantees. The trust assumptions specify which protocol participants are assumed to be honest and which of them can be corrupted and at what point in time during a protocol run. Note that given a run, we can easily identify whether and when a protocol participant is corrupted or not. Hereafter, for a given run r of an e-voting protocol, n denotes the number of eligible, n_h the number of honest and n_d the number of dishonest voters in r. The actual choices of the honest voters in r are denoted by $c_1, \ldots, c_{n_h}$. Recall that abstention is a possible choice as well.

Qualitative Goal

Informally, the qualitative goal $\gamma_{ql}(\varphi)$ demands that, in case the trust assumptions φ are met in a protocol run, then

(i) the final result contains the choices of all honest voters who performed their checks successfully,

(ii) votes of those honest voters who did not perform their check may be dropped, but not altered, and

(iii) there exists at most one ballot cast for each dishonest voter, i. e., ballot stuffing is not permitted.

In case φ does not hold true in a protocol run, the protocol is not expected to guarantee anything in this run. Formally, $\gamma_{ql}(\varphi)$ is fulfilled in a protocol run r (i. e., $r \in \gamma_{ql}(\varphi)$) if either (a) the trust assumptions φ are not met in r, or if (b) φ is met in r and there exist valid choices $\tilde{c}_1, \ldots, \tilde{c}_n$ fulfilling the following conditions:

(i) An election result is published in r which is equal to $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$.

(ii) The multiset $\{\tilde{c}_1, \ldots, \tilde{c}_n\}$ includes all actual choices of honest voters who performed their check successfully, plus a subset of actual choices of honest voters who did not perform their check (successfully), and plus at most n_d additional choices.

Note, there are checks, such as the mentioned Benaloh checks, which do not completely guarantee that votes have actually been counted. If voters perform such checks, then applying this goal, a tolerance $\delta > 0$ will be obtained, since manipulation might remain undetected with some probability. In addition, the requirement that votes of honest voters who did not perform their check (successfully) can be at most dropped, however not altered, might only be achievable under certain trust assumptions.

Quantitative Goal

Informally, the quantitative goal $\gamma_{qn}(k, \varphi)$ demands that, in case the trust assumptions φ are met in a protocol run, then the distance between the published and the ideal[1] result is bounded by k. Therefore, a particular distance function d on the election results is introduced. For the purpose of defining d, we first consider a function $f_{\text{count}} \colon C^l \to \mathbb{N}^C$. It takes as input a vector $(c_1, \ldots, c_l) \in C^l$ which represents a multiset of the voters' choices and outputs the number of occurrences of every choice $c_i \in C$ in $(c_1, \ldots, c_l)$. For instance, $f_{\text{count}}(A, B, B)$ assigns 1 to choice A, 2 to choice B, and 0 to all the remaining choices. Now, we use f_{count} in order to define d. The distance function d takes as input two vectors of choices c, c' and is defined as

$$d(c, c') = \sum_{c_i \in C} |f_{\text{count}}(c)[c_i] - f_{\text{count}}(c')[c_i]| \,.$$

As an example, let $c = (A, B, B)$ be the published votes and $c' = (B, B, B, C)$ be the ideal votes. Then, one vote has been dropped and another one has been altered. Thus, for the distance d between the published and the ideal result, we would get $d((A, B, B), (B, B, B, C)) = 3$. This is because altering a vote increases the distance by 2 and dropping a vote increases the distance by 1 since we wanted to distinguish between votes that have been altered and votes that have been dropped as mentioned in the previous section. Formally, $\gamma_{qn}(k, \varphi)$ is fulfilled in a protocol run r if either (a) the trust assumptions φ are not met in r, or if (b) φ is met in r and there exist valid choices $c'_1, \ldots, c'_{n_d}$ (choices of dishonest voters) and $\tilde{c}_1, \ldots, \tilde{c}_n$ satisfying the following conditions:

(i) An election result is published which is equal to $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$, and

(ii) $d((c_1, \ldots, c_{n_h}, c'_1, \ldots, c'_{n_d}), (\tilde{c}_1, \ldots, \tilde{c}_n)) \leq k$.

In case the adversary drops a single honest vote, d is increased by one, whereas if the adversary alters a single choice of an honest vote (by another one), d is increased by two. This allows to model the actual effect of a manipulation on the final result. Since it is unlikely that all voters will check their receipts, some manipulation will remain undetected. Thus, $\delta = 0$ is usually not achievable for the quantitative goal $\gamma_{qn}(k, \varphi)$. Given the parameter k, performing a security analysis on a concrete protocol will determine the minimal δ.

[1]The ideal result describes the result we get if the actual choices of honest voters as well as one choice for each dishonest voter are counted.

6 Russian Federal Remote E-Voting Scheme

In this chapter, we first present the participants involved in the Russian federal remote e-voting scheme. Afterwards, we discuss the usage of the Blockchain. This is followed by a description of the e-voting protocol including its phases. At the end, we investigate the main cryptographic protocols and algorithms used in the e-voting scheme. The information in the first three sections is taken from [50].

6.1 Participants

In the Russian federal e-voting protocol, there are several participants interacting with each other. This section introduces the main protocol participants and describes their roles in the process of e-voting.

Voter

The Voter is any citizen of the Russian Federation meeting two requirements. At first, they need to be eligible to vote. Additionally, they have to be included in the lists of e-voters referred to as VoterList. In order to be included in the VoterList, each citizen has to submit an application in electronic form through `gosuslugi.ru`. This is a web portal where information about state and municipal services in the Russian Federation can be accessed. The Voters can only register as an e-voting participant in case they have verified their `gosuslugi.ru` account. The lists at the local polling stations exclude the Voters included in the VoterList. Voters own their personal SNILS which is an insurance account number individual to each citizen.

In order to participate in e-voting, the Voter uses a Voting Device, a device providing a browser and Internet access. There are two ways to vote, either through a browser at `gosuslugi.ru` or alternatively through their mobile application which is available for Android and iOS. During the authorisation phase, the Voting Device generates a key pair for the GOST signature scheme in order to sign the Voters' ballot.

Organiser

The Organiser is coordinating the e-voting process and in addition responsible for the generation of the Organiser's key pair as well as for the construction of the final encryption key. The final encryption key allows to encrypt all votes (see Figure 6.2).

Internal and External Observer

The Internal Observer's role is to monitor the e-voting process. This is done from a certain room. Additionally, the Internal Observer has access to individual nodes of the Blockchain and performs the audit which includes several verifications.

The External Observer is any user with access to https://stat.vybory.gov.ru. This website allows the Voter to check if the cast ballot was added to the Blockchain. However, it became inaccessible one month after the elections.

Internal and External Observers are referred to as the Election Observer.

Key Holders

Key holders are selected by the Organiser. They are responsible for holding shares of the Organiser's secret key.

Registrar

The Registrar is composed of the Voting Portal and VoterList components. It is responsible for the identification and authentication of the Voters. This is done through the unified identification and authentication system of the Russian Federation called ESIA. ESIA provides authorised access for citizens to the information contained in state information systems. The Registrar also issues blind signatures to the Voters' public keys.

Vote Collector

The Vote Collector is a separate component. It allows casting as well as maintaining the secrecy of votes and issues ballots to the Voters. After the Voter has cast a ballot, the Vote Collector collects encrypted votes. In order to publish the encrypted votes, it interacts with the Blockchain.

Tallier

The Tallier comprises the distributed storage, represented by the Blockchain as well as Decryptor components. The Blockchain is responsible for storing all the voting transactions and published keys. The Decryptor contains a hardware security module (HSM). It allows to generate the Tallier's key pair and to tally the votes.

6.2 Usage of Blockchain

This section is concerned with the usage of the Blockchain in the e-voting protocol. As described in Chapter 4, an e-voting system makes use of a bulletin board to store public information such as for example cast ballots. In the Russian e-voting protocol, this functionality is provided by the Blockchain. The used Blockchain platform was developed by Waves Enterprise [23]. It uses the Crash Fault Tolerance (CFT) consensus algorithm [21] that is based on Proof of Authority (PoA) consensus [22]. The Blockchain platform supports the development and usage of smart contracts. The smart contracts used in the e-voting process perform the following functions:

- storing the rules of the voting process and the list of participants,
- registering information, obtained during the setup phase, and
- verification and storage of the cast votes and voting results.

There are four data processing centers that run the Blockchain nodes. These centers are managed by Rostelecom (digital services provider in Russia) and the Registrar.

6.3 Protocol

This section presents the protocol the Russian federal remote e-voting scheme is based on. It is divided into four phases, the setup phase, the authorisation phase, the voting phase as well as the tallying phase. After each of the four phases, we present the current state of the Blockchain. For simplicity, there is one block for each phase. Note that this does not correspond to how the content is actually stored in the Blockchain. However, this is not of importance here as we only aim to visualize what information is contained in the Blockchain and thereby show what information is publicly available. Additionally, there is an audit. Finally, we provide a short discussion on the generation of the ElGamal encryption key that is used for the encryption of the votes and on the way key sharing is realized.

6.3.1 Setup Phase

The setup phase is mostly an interaction between the Organiser, the Registrar and the Tallier where keys are generated, public keys are exchanged and public keys as well as information about the voting process are uploaded to the Blockchain. A more detailed description is provided in the following.

- The Organiser and Registrar generate key pairs for the GOST signature scheme and send their public key to the Tallier. All messages that are sent by the Organiser as well as the Registrar are signed with their secret key. The corresponding signatures are verified by the Tallier using the Organiser's and Registrar's public key respectively.
- The Registrar generates an RSA blind signature key pair (sk_b, pk_b) as well as a commitment key K_{com}. The public key pk_b is sent to the Organiser.

- The Organiser generates an ElGamal key pair (S_{org}, Q_{org}). The generation of the key pair is done in presence of the Election Observers and the media. After generating the key pair, S_{org} is split into shares using Shamir Secret Sharing. The reason why the secret key S_{org} is split into shares will become clear later. The resulting key shares are stored externally at the storage media of the secret key holders. After the key shares have been transferred, S_{org} is deleted from the generating device.
- The Decryptor generates an ElGamal key pair (S_t, Q_t). The public key Q_t is sent to the Organiser.
- The Organiser uploads the following information to the Blockchain:
  - Identifier of elections $(votingID)$
  - Starting time of receiving ballots (t_{start})
  - Hash of the ballot text $(h(\text{ballot text}))$
  - Number of options in each ballot (n)
  - Maximum number of options that each Voter can select (d)
  - pk_b

  This information is used to generate the Blockchain smart contracts. Additionally, the Organiser sends the VoterList to the Registrar.
- The Registrar computes commitments

  $$com = HMAC(K_{com}, SNILS||votingID)$$

  on the Voters' SNILS codes from the VoterList. They are uploaded into the Blockchain and the Blockchain smart contracts are updated by adding the commitments.
- The Organiser constructs the final encryption key

  $$Q_f = H(Q_t||Q_{org}) \cdot Q_{org} + H(Q_{org}||Q_t) \cdot Q_t$$

  and uploads Q_{org}, Q_t and Q_f to the Blockchain. The Registrar receives Q_f from the Blockchain. Since the construction of the final encryption key Q_f required the public keys of the Organiser (Q_{org}) and the Tallier (Q_t), both secret keys are necessary to decrypt the votes. Neither the Organiser nor the Tallier are able to learn intermediate results because of the Organiser's secret key S_{org} being split until the tallying phase.

After the setup phase is completed, the following information is contained in the Blockchain:

$\boxed{\text{Block 1}}$

$\boxed{\text{Hash Block}}$

$votingID$
t_{start}
$h(\text{ballot text})$
n
d
pk_b
$\ldots, com_i, \ldots$
Q_{org}
Q_t
Q_f

$\boxed{\text{Hash Block 1}}$

Figure 6.1: Blockchain in Setup Phase

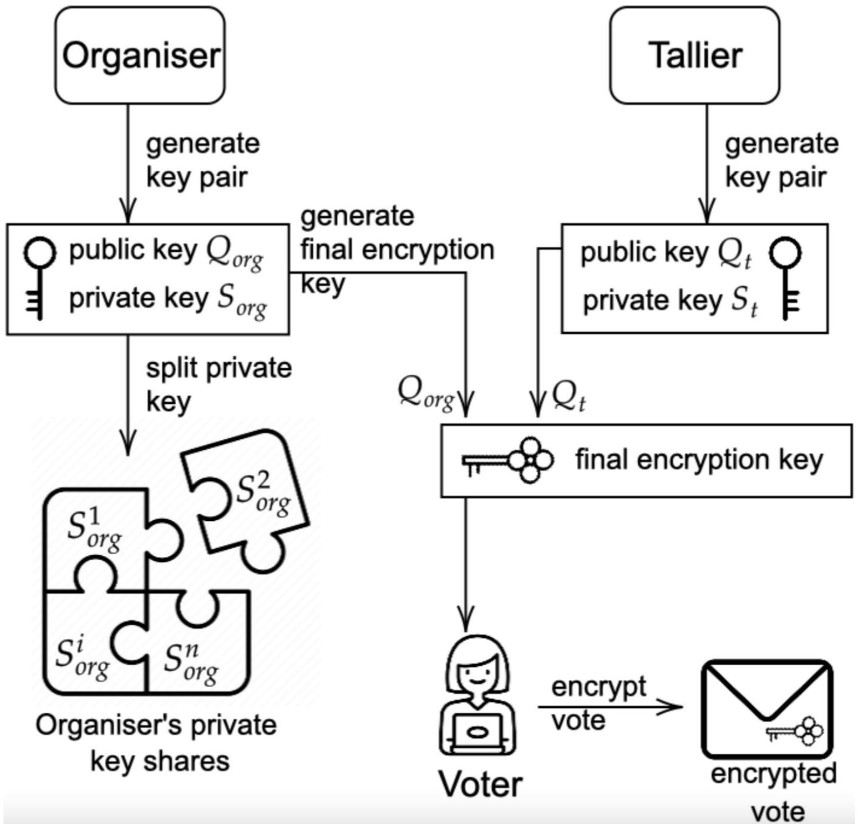

Figure 6.2: Relationship between Keys [50]

6.3.2 Authorisation Phase

The authorisation phase is mostly an interaction between the Voter and the Registrar where the Voter is authorised to the e-voting portal, the eligibility of the Voter is checked and the Registrar issues a signature on the Voter's public key. A more detailed description is provided below.

- In order to cast a ballot, the Voter has to authenticate to the e-voting portal. This is done through the ESIA system. The Registrar receives the signed id_token as well as the information about the Voter from ESIA. The Registrar uses the Voter's SNILS code to check for her eligibility. After eligibility has been verified, the Registrar sends an SMS or an email containing an authorisation code to the Voter. The Voter inputs the code into the e-voting portal.
- The Voting Device generates a key pair (sk_v, pk_v) for the GOST signature scheme. It interacts with the Registrar and sends the Voter's masked public key. The Registrar sends back an RSA blind signature s on the masked public key.
- The Registrar adds id_token, com, s issued for the Voter to its VoterList and publishes (com, s) into the Blockchain. Furthermore, the Registrar sends $votingID$ and Q_f to the Voting Device.
- The Voting Device removes the mask from the signature s resulting in σ_b which is a valid RSA signature on the Voter's public key.

The state of the Blockchain during the authorisation phase is depicted in the following:

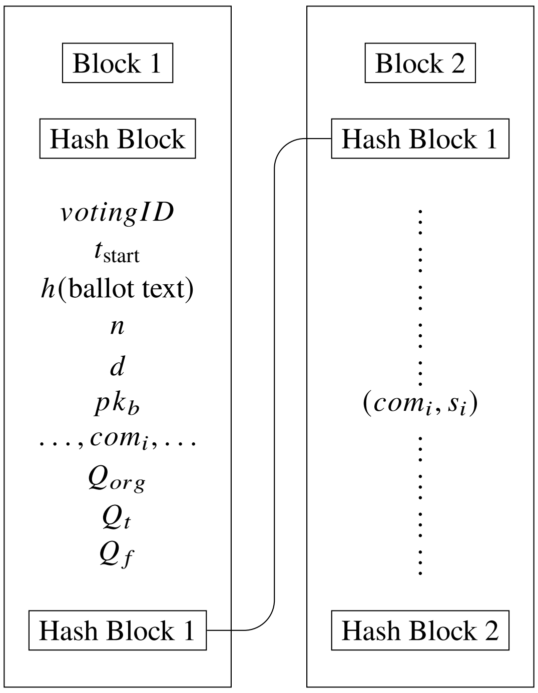

Figure 6.3: Blockchain in Authorisation Phase

6.3.3 Voting Phase

The voting phase is mostly an interaction between the Voter, the Blockchain, the Vote Collector and the Voting Device where the Voter casts one's ballot, the Voting Device generates proofs for the correctness of the cast ballot and the Voting Device generates the voting transaction which is sent to the Vote Collector and finally uploaded to the Blockchain. A more detailed description is provided in the following.

- After being authorised to the e-voting portal, the Voter is redirected to the anonymous zone of the portal, the Vote Collector. This is where the Voter is presented with a ballot in digital form and where the Voter makes her choice by selecting the preferred option.
- Each ballot b is represented as a bitstring of length n since there are n options to choose from. Every option b_i on the ballot is initialized with zero $(b_i = 0)$. The value of the options chosen by the Voter changes to one $(b_i = 1)$. ElGamal encryption is used to encrypt each option separately

$$c_i = Enc(b_i, Q_f), \text{ for } i \in \{0, \ldots, n-1\}.$$

- The Voting Device generates a range proof for each ciphertext c_i, demonstrating that $b_i \in \{0, 1\}$. Additionally, the Voting Device generates a range proof providing evidence that $\sum_{i=0}^{n-1} b_i \leq d$ since the Voter can choose a maximum of d options (see Figure 6.4).
- After generating the range proofs, the Voting Device prepares the transaction. The transaction consists of all created ciphertexts and their corresponding proofs, pk_v and σ_b (see Figure 6.4). This transaction is signed by the Voting Device using sk_v in order to receive the signature σ_v. Finally, it sends the signed transaction to the Vote Collector.
- After the Vote Collector has received the signed transaction, it verifies the cast ballot for well-formedness, the signature σ_b and uniqueness of the transaction containing pk_v. Furthermore, the Vote Collector adds the Voter's public key pk_v to its internal database and uploads the transaction to the Blockchain.
- The Blockchain smart contract verifies ballot well-formedness as well as the signatures σ_b and σ_v, and publishes the transaction[1].

After the Voter has cast a vote, she can visit the `https://stat.vybory.gov.ru` portal and use her public key pk_v in order to check whether her transaction has been added to the Blockchain.

Figure 6.4 depicts the process of generating the voting transaction.

[1]Here, we assume that in case one of the two signatures σ_b and σ_v is invalid, the voting transaction is marked as invalid. Note that this is our assumption and thus not part of the original protocol description from [50].

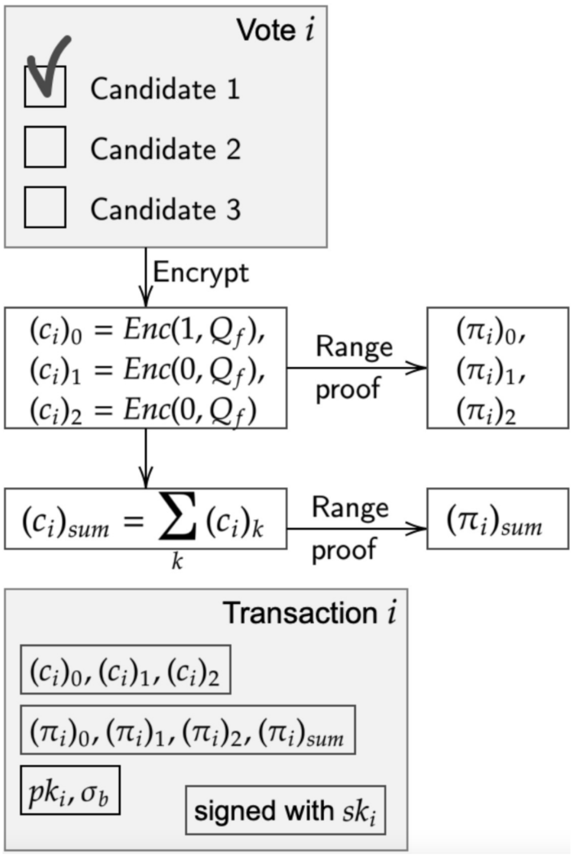

Figure 6.4: Voting Transaction [50]

The state of the Blockchain during the voting phase is shown below:

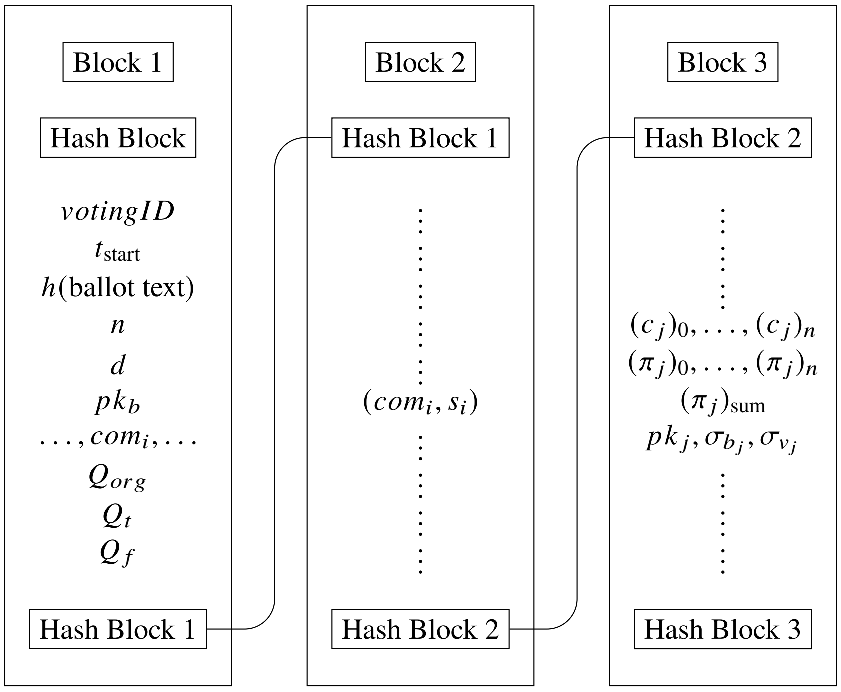

Figure 6.5: Blockchain in Voting Phase

6.3.4 Tallying Phase

The tallying phase is mostly an interaction between the Tallier and the Organiser where the proofs of each voting transaction are verified, the verified encrypted votes are aggregated and then decrypted and the voting result is uploaded to the Blockchain. A more detailed description is provided below.

- Since the votes should be tallied now, the Organiser asks the Registrar for stopping the authentication of new Voters. Additionally, the Blockchain is requested to stop accepting new voting transactions.
- The Organiser uses the secret key shares in order to reconstruct S_{org}.
- The Decryptor receives all the voting transactions from the Blockchain. For each transaction, the corresponding range proofs are verified[2]. In addition, the Decryptor aggregates verified encrypted votes separately[3]. This is done for each option from the ballot as

$$sum_i = \sum_{v=1}^{V} (c_v)_i = (R_i, C_i),$$

where $i \in \{0, \ldots, n-1\}$ and V is the total number of cast votes (see Figure 6.6). Thus, sum_i states the encryption of the amount of votes that option/candidate i received.

[2]Here, we assume that only the range proofs of such transactions that are not marked as invalid are verified. Note that this is our assumption and thus not part of the original protocol description from [50].

[3]According to the assumption in the previous footnote, this means that we assume that the Decryptor only aggregates verified encrypted votes that are contained in a transaction that is not marked as invalid.

- The decryption of the aggregated ciphertexts sum_i consists of two steps. At first, the Decryptor computes partial decryptions as $(R_i)_t = S_t \cdot R_i$. Afterwards, it generates proofs of correctness of partial decryptions. Besides, the Decryptor uploads all partial decryptions $(R_i)_t$ together with the corresponding proofs of decryption correctness into the Blockchain.
- The Organiser uploads S_{org} into the Blockchain. Thereupon, the Decryptor receives S_{org} and verifies its correspondence to the public key Q_{org}.
- After the correspondence has been verified, the Decryptor performs the final decryption of aggregated votes using S_{org} as

$$M_i = C_i - H(Q_t||Q_{org}) \cdot S_{org} \cdot R_i - H(Q_{org}||Q_t) \cdot (R_i)_t,$$

  where M_i states the amount of votes that candidate i received. The transaction (R_i, C_i), M_i $(i \in \{0, \dots, n-1\})$ is published by the Decryptor.

Note, that only the Organiser's secret key S_{org} but not the Tallier's secret key S_t is published. This means that there is no possibility to decrypt transactions containing individual votes by only using information that is publicly available. In order to decrypt transactions, the secret key S_t is required as well which is not publicly available. Thus, the Voter is not able to verify whether the published vote in the Blockchain actually corresponds to her original choice.

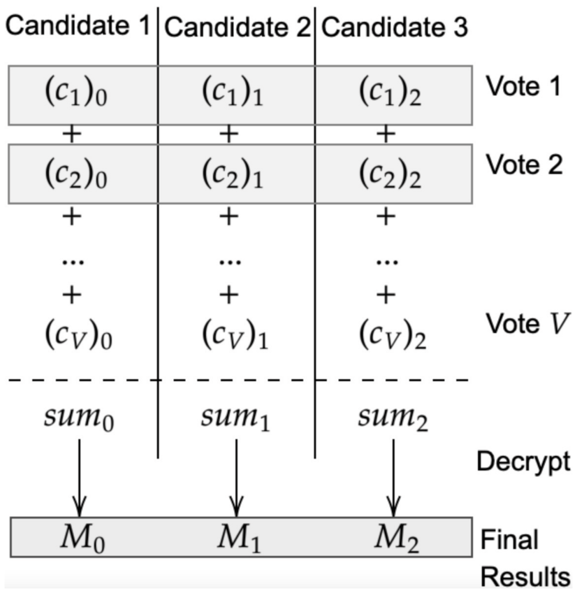

Figure 6.6: Vote Tallying [50]

At the end of the tallying phase, the following information is contained in the Blockchain:

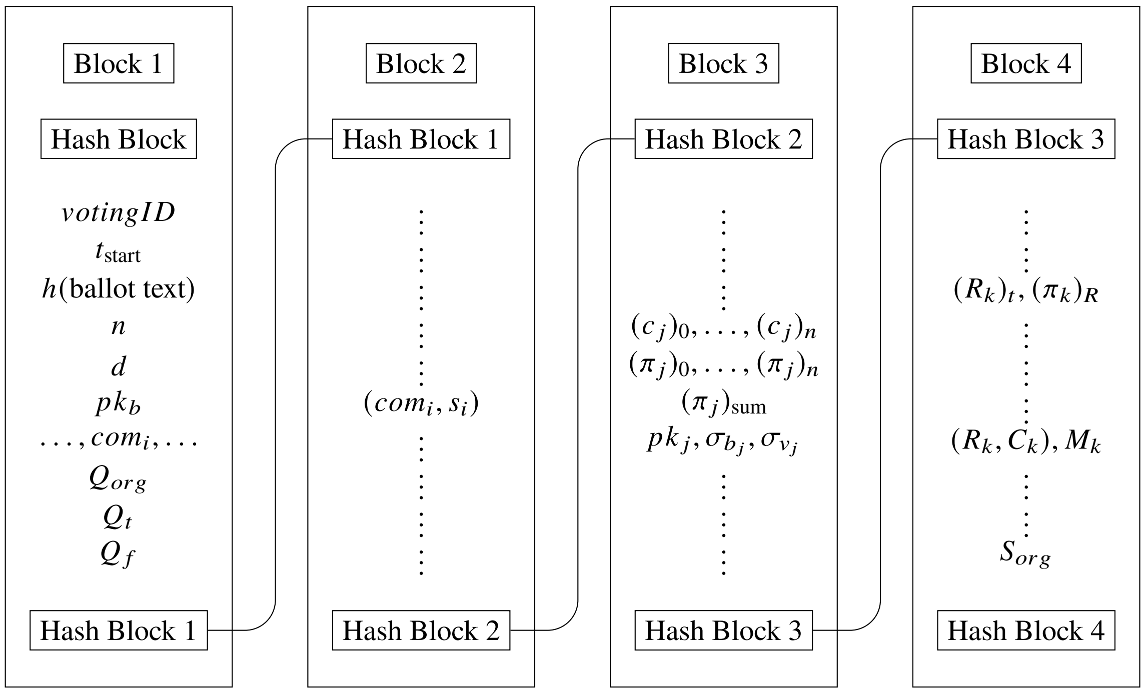

Figure 6.7: Blockchain in Tallying Phase

6.3.5 Audit

In the audit, the Internal Observer can perform the following verifications:

- verify that for every Voter who has been issued a blind signature according to the Registrar's VoterList, there exists a valid id_token from ESIA and a transaction in the Blockchain with a commitment on the Voter's SNILS code,
- verify commitments on the Voters' SNILS codes,
- verify that the amount of cast votes does not exceed the amount of Voters to whom the Registrar issued blind signatures,
- verify the correctness of the blind signatures and the Voters' signatures from the voting transaction,
- verify that there is only one transaction for each Voter's public key.

6.3.6 Discussion

This subsection aims to discuss the process of the generation as well as the usage of the final encryption/decryption key pair. The idea that the secret key is not held by a single participant, but rather shares of the ElGamal secret key are created and decryption is performed in a distributed manner, is not new. The usual approach for this idea is described next. At first, a set of trustees who generate their ElGamal key pairs independent of each other is selected. Afterwards, the public keys of each trustee are combined homomorphically in order to create a single ElGamal public key. This public key is the key used for the encryption of the votes. In this case, the secret key that belongs to the previously created public key allowing to decrypt the votes is not reconstructed from the shares. To decrypt the votes, each trustee individually computes a partial decryption using her secret key. The full decryption of the election results can be obtained by combining all partial decryptions. More information on realizing a distributed version of the ElGamal encryption scheme can be found in [13].

However, the approach that has been used in the Russian federal e-voting protocol (see Figure 6.2) is different to the one described above since the Organiser's secret key share is generated and split afterwards, instead of an independent generation of key shares from the beginning. The better practice is to use the more established standard approach for the generation of the ElGamal encryption key as well as for key sharing.

6.4 Main Cryptographic Protocols and Algorithms

In this section, we investigate the main cryptographic protocols and algorithms that have been used in the Russian federal remote e-voting scheme. The table below provides an overview of the main cryptographic protocols and algorithms used, with the left column representing the cryptographic primitives and the right column representing the used cryptographic protocols and algorithms.

Cryptographic Primitives	Cryptographic Protocols and Algorithms
Cryptographic Hash Function	GOST 34.11-2012
Public-Key Encryption Scheme	EC ElGamal Encryption Scheme
Digital Signature Scheme	GOST 34.10-2012
Blind Digital Signature Scheme	RSA Blind Digital Signature Scheme
Commitment Scheme	HMAC_GOSTR3411_2012_256
Secret Sharing	Shamir Secret Sharing
Zero-Knowledge Proofs	(Disjunctive) Chaum-Pedersen Proof

Table 6.1: Main Cryptographic Protocols and Algorithms Overview

6.4.1 Cryptographic Hash Function

In this subsection, we illustrate the Russian federal standard hash function GOST R 34.11-2012, the cryptographic hash function (see Section 3.2) used in the Russian federal remote e-voting scheme. The cryptographic hash function GOST R 34.11-2012 is used as part of the construction of the final encryption key Q_f, the final decryption of aggregated votes and the computation of the commitments as we will see later. The information of this subsection is based on the RFC (6986) document [19]. At first, we introduce necessary notation and then specify the calculation procedure of the hash function.

Notation

The standard for the hash function uses the subsequent notation:

- V^* is the set of all bit strings of finite length including the empty string
- V_n is the set of all bit strings of length n (non-negative integer)
- A^n is the concatenation of n instances of the bit string A
- Z_{2^n} is the ring of residues modulo 2^n
- $[+]$ is the addition operation in the ring Z_{2^n}
- $\mathrm{Vec}_n \colon Z_{2^n} \rightarrow V_n$ is a bijective mapping that maps an element from Z_{2^n} to its binary representation
- $\mathrm{Int}_n \colon V_n \rightarrow Z_{2^n}$ is the inverse mapping to Vec_n
- $\mathrm{MSB}_n \colon V^* \rightarrow V_n$ is a mapping that maps an element from V^* to an element from V_n which reflects the n most significant bits of the input element
- $H \colon V^* \rightarrow V_n$ is a mapping called hash function that maps an element from V^* to an element from V_n, the output is referred to as hash
- $g_N \colon V_{512} \times V_{512} \rightarrow V_{512}$ is a mapping that maps two bit strings of length 512 to a single bit string of length 512 (see [19] for how this mapping works internally)

Hash-Function Calculation Procedure

GOST R 34.11-2012 defines hash functions H for hash lengths of $n = 512$ and $n = 256$ bits. The procedure for calculating the hash $H(M)$ requires a message $M \in V^*$ as well as the initializing value $IV \in V_{512}$ as initial data.

Algorithm 6.1 GOST R 34.11-2012 Hash Function

```
Input: Message $M$, initializing value $IV$
Output: Hash $H(M)$
1: $h := IV$[2]
2: $N := 0^{512}$
3: $\varepsilon := 0^{512}$
4: if $|M| < 512$ then
5:     go to line 14
6: else
7:     $m :=$ Subvector belonging to $V_{512}$ of the message $M$ such that $M = M' \| m$
8:     $h := g_N(h, m)$
9:     $N := \mathrm{Vec}_{512}(\mathrm{Int}_{512}(N)\ [+]\ 512)$
10:    $\varepsilon := \mathrm{Vec}_{512}(\mathrm{Int}_{512}(\varepsilon)\ [+]\ \mathrm{Int}_{512}(m))$
11:    $M := M'$
12:    go to line 4
13: end if
14: $m := 0^{511-|M|} \| 1 \| M$
15: $h := g_N(h, m)$
16: $N := \mathrm{Vec}_{512}(\mathrm{Int}_{512}(N)\ [+]\ |M|)$
17: $\varepsilon := \mathrm{Vec}_{512}(\mathrm{Int}_{512}(\varepsilon)\ [+]\ \mathrm{Int}_{512}(m))$
18: $h := g_0(h, N)$
19: $h := \begin{cases} g_0(h,\varepsilon) & \text{for the function with 512-bit hash} \\ \mathrm{MSB}_{256}(g_0(h,\varepsilon)) & \text{for the function with 256-bit hash} \end{cases}$
20: $H(M) := h$
21: Return $H(M)$
```

6.4.2 Public-Key Encryption Scheme

This subsection provides a description of the EC-ElGamal encryption scheme which is utilized for the encryption and decryption of the votes. Its basis is the original ElGamal encryption scheme. The original ElGamal encryption scheme, based on the public key distribution scheme by Diffie and Hellman [17], was introduced by ElGamal in [20]. The Diffie-Hellman key distribution scheme will not be discussed further, information about this scheme can be extracted from [17] and [20]. In order to describe the EC-ElGamal encryption scheme, we specify the key-generation algorithm Gen, the encryption algorithm Enc as well as the decryption algorithm Dec (see Section 3.3). Contrary to the original ElGamal encryption scheme which is not additive homomorphic [49], the EC-ElGamal encryption scheme provides the additive homomorphic property (see Section 3.4) as we will prove at the end of this subsection.

Before specifying the corresponding algorithms, we have to introduce the preliminaries as outlined in [49] first. In the following, let $E_p(a,b)$ be an elliptic curve over the field F_p and $n = |E_p(a,b)|$ be its order. Further, let $G \in E_p(a,b)$ denote a generator point of $E_p(a,b)$. The elliptic curve

[2]For H with $|H(M)| = 512$, $IV = 0^{512}$. For H with $|H(M)| = 256$, $IV = (00000001)^{64}$.

$E_p(a,b)$, n and G are publicly known. The addition over an elliptic curve only works with points on that curve. Thus, for EC-ElGamal, we can not just encrypt the messages, but rather have to map the messages to points on $E_p(a,b)$ and then perform the actual encryption. This mapping of messages, as part of the encryption algorithm is done by the deterministic function $map\colon F_p \rightarrow E_p(a,b)$ such that the additive homomorphic property

$$map(m_1 + \ldots + m_n) = \underbrace{map(m_1)}_{M_1} + \ldots + \underbrace{map(m_n)}_{M_n}$$

holds true for messages $m_1, \ldots, m_n \in F_p$. The inverse mapping function which is denoted by *imap* maps points on $E_p(a,b)$ to messages on F_p. One possibility to map the messages to points on the elliptic curve looks as follows:

$$map\colon m \rightarrow mG \text{ with } m \in F_p$$

This function exhibits the requested additive homomorphic property since

$$\begin{aligned} map(m_1 + \ldots + m_n) &= (m_1 + \ldots + m_n)G \\ &= m_1G + \ldots + m_nG \\ &= map(m_1) + \ldots + map(m_n) \end{aligned}$$

holds true for messages $m_1, \ldots, m_n \in F_p$. Additional possibilities for the mapping function *map* can be taken from [32] and [33].

Key-Generation Algorithm Gen

Algorithm 6.2 EC ElGamal Key-Generation Algorithm Gen [49]

```
Input: Security parameter $1^n$
Output: Public key $Y$, private key $x$
1: Choose random $x \in F_p$
2: $Y := xG$
3: Return $(Y, x)$
```

Encryption Algorithm Enc

Algorithm 6.3 EC-ElGamal Encryption Algorithm Enc [49]

```
Input: Public key $Y$, plaintext $m$
Output: Ciphertext $(R, S)$
1: Choose random $k \in [1, n-1]$
2: $M := map(m)$
3: $R := kG$
4: $S := M + kY$
5: Return $(R, S)$
```

Decryption Algorithm Dec

Algorithm 6.4 EC-ElGamal Decryption Algorithm Dec [49]

```
Input: Private key $x$, ciphertext $(R, S)$
Output: Plaintext $m$
1: $M := S - xR$
2: $m := imap(M)$
3: Return $m$
```

Additive Homomorphic Property

Adapted from [37], we now prove that the EC-ElGamal encryption scheme provides the additive homomorphic property that allows the aggregation of encrypted votes. Therefore, we show that the following equality holds true.

$$\mathsf{Dec}(C_1 + \ldots + C_n) = m_1 + \ldots + m_n \tag{6.1}$$

At first, we prove that

$$C_1 + \ldots + C_n = (k'G, map(m_1 + \ldots + m_n) + k'Y), \tag{6.2}$$

i. e., that the addition of ciphertexts is equal to the encryption of the addition of plaintexts and then use this to show equality 6.1. In the following, $k' = k_1 + \ldots + k_n$.

$$\begin{aligned}
C_1 + \ldots + C_n &= (R_1, S_1) + \ldots + (R_n, S_n) \\
&= (k_1G, M_1 + k_1Y) + \ldots + (k_nG, M_n + k_nY) \\
&= (k_1G + \ldots + k_nG, M_1 + k_1Y + \ldots + M_n + k_nY) \\
&= (k_1G + \ldots + k_nG, M_1 + \ldots + M_n + k_1Y + \ldots + k_nY) \\
&= ((k_1 + \ldots + k_n)G, M_1 + \ldots + M_n + (k_1 + \ldots + k_n)Y) \\
&= (k'G, M_1 + \ldots + M_n + k'Y) \\
&= (k'G, map(m_1) + \ldots + map(m_n) + k'Y) \\
&= (k'G, map(m_1 + \ldots + m_n) + k'Y)
\end{aligned}$$

Using equality 6.2, we can conclude equality 6.1.

$$\begin{aligned}
\mathsf{Dec}(C_1 + \ldots + C_n) &= \mathsf{Dec}((k'G, map(m_1 + \ldots + m_n) + k'Y)) \\
&= map(m_1 + \ldots + m_n) + k'Y - xk'G \\
&= imap(map(m_1 + \ldots + m_n) + k'Y - xk'G) \\
&= imap(map(m_1 + \ldots + m_n)) \\
&= m_1 + \ldots + m_n
\end{aligned}$$

Remarks

At this point, we want to add some remarks on the last two steps of the tallying phase. In the penultimate step, the Decryptor verifies that the secret key S_{org} corresponds to the public key Q_{org}. According to the key generation, this verification can be done by multiplying S_{org} with the generator point G of the used elliptic curve and then checking whether the result is equal to Q_{org}. In the last step, the Decryptor performs the final decryption of aggregated votes as

$$M_i = C_i - H(Q_t||Q_{org}) \cdot S_{org} \cdot R_i - H(Q_{org}||Q_t) \cdot (R_i)_t$$

In the following, we show why this equality holds. Note that here, we ignore the mapping of messages to points on the elliptic curve.

$$\begin{aligned}
C_i - H(Q_t||Q_{org}) \cdot S_{org} \cdot R_i - H(Q_{org}||Q_t) \cdot (R_i)_t \\
&= M_i + k_i Q_f - H(Q_t||Q_{org}) \cdot S_{org} \cdot R_i - H(Q_{org}||Q_t) \cdot (R_i)_t \\
&= M_i + k_i Q_f - H(Q_t||Q_{org}) \cdot S_{org} \cdot k_i \cdot G - H(Q_{org}||Q_t) \cdot S_t \cdot R_i \\
&= M_i + k_i Q_f - H(Q_t||Q_{org}) \cdot S_{org} \cdot k_i \cdot G - H(Q_{org}||Q_t) \cdot S_t \cdot k_i \cdot G \\
&= M_i + k_i Q_f - k_i(H(Q_t||Q_{org}) \cdot S_{org} \cdot G - H(Q_{org}||Q_t) \cdot S_t \cdot G) \\
&= M_i + k_i Q_f - k_i(H(Q_t||Q_{org}) \cdot Q_{org} - H(Q_{org}||Q_t) \cdot Q_t) \\
&= M_i + k_i Q_f - k_i Q_f \\
&= M_i
\end{aligned}$$

6.4.3 Digital Signature Scheme

This subsection illustrates the Russian federal standard for digital signatures GOST R 34.10-2012 which is the digital signature scheme that is used in the Russian federal remote e-voting scheme. The following information is based on the RFC (7091) document [18]. The digital signature scheme is mostly used for the signing of votes but also to sign the messages sent by the Organiser and Registrar. In order to specify the standard for digital signatures, we essentially need to define the key-generation algorithm Gen, the signing algorithm Sign and the verification algorithm Vrfy (see Section 3.5). The algorithms Sign and Vrfy require digital signature scheme parameters as listed below[3]:

- Prime number p which is an elliptic curve modulus
- Elliptic curve $E_p(a, b)$
- Prime number q which is the order of a cyclic subgroup of the group consisting of the elliptic curve $E_p(a, b)$ points (q needs to satisfy two conditions that can be found in [18])
- Point $P \neq O$ (identity element of $E_p(a, b)$) of an elliptic curve $E_p(a, b)$ which satisfies $q \cdot P = O$
- Hash function h which is defined in GOST R 34.11-2012 [19]

Those parameters have to satisfy certain requirements that are specified in [18]. However, the standard for digital signatures does not determine the process for generating them. For example, [42] defines possible sets of those parameters.

[3]Parameters not directly relevant for the two algorithms are not mentioned here. They can be found in [18].

Key-Generation Algorithm Gen

Note that GOST R 34.10-2012 does not define an algorithm Gen for generating a pair of keys. However, according to [18], ways to realize the key-generation algorithm Gen are defined by involved subjects, who determine corresponding parameters based on their agreement.

Signing Algorithm Sign

Algorithm 6.5 GOST R 34.10-2012 Signing Algorithm Sign [18]

```
Input: Private key $d$[4], message $M$
Output: Signature $\sigma$
1: $H := h(M)$
2: $\alpha :=$ Integer whose binary representation is $H$
3: $e := \alpha \bmod q$
4: if $e = 0$ then
5:     $e := 1$
6: end if
7: $k :=$ Random (pseudorandom) integer such that $0 < k < q$
8: $C := k \cdot P$
9: $r := x_C \bmod q$                // $x_C$ is the $x$-coordinate of the point $C$
10: if $r = 0$ then
11:     go to line 7
12: end if
13: $s := (r \cdot d + k \cdot e) \bmod q$
14: if $s = 0$ then
15:     go to line 7
16: end if
17: $R :=$ Binary vector corresponding to $r$
18: $S :=$ Binary vector corresponding to $s$
19: $\sigma := (R \| S)$                // Concatenation of $R$ and $S$
20: Return $\sigma$
```

[4]The private key d is an integer which satisfies $0 < d < q$.

Verification Algorithm Vrfy

Algorithm 6.6 GOST R 34.10-2012 Verification Algorithm Vrfy [18]

```
Input: Public key $Q$[5], message $M$, signature $\sigma$
Output: 1 if signature is valid, 0 if signature is invalid
 1: $r, s :=$ Calculate integers from $\sigma$
 2: if $0 < r < q \wedge 0 < s < q$ then
 3:     continue
 4: else
 5:     Return 0
 6: end if
 7: $H := h(M)$
 8: $\alpha :=$ Integer whose binary representation is $H$
 9: $e := \alpha \bmod q$
10: if $e = 0$ then
11:     $e := 1$
12: end if
13: $v := e^{-1} \bmod q$
14: $z_1 := s \cdot v \bmod q$
15: $z_2 := -r \cdot v \bmod q$
16: $C := z_1 \cdot P + z_2 \cdot Q$
17: $R := x_C \bmod q$                // $x_C$ is the $x$-coordinate of the point $C$
18: if $R = r$ then
19:     Return 1
20: else
21:     Return 0
22: end if
```

6.4.4 Blind Digital Signature Scheme

This subsection provides a description of the RSA blind signature scheme. More precisely, we specify the key-generation algorithm Gen, the signing algorithm $\langle \mathcal{S}, \mathcal{U} \rangle$ as well as the deterministic verification algorithm Vrfy (see Section 3.6) of the RSA blind signature scheme. In the Russian federal remote e-voting scheme, this blind digital signature scheme is used to preserve the anonymity of the voters by the Registrar (taking the role of the singer $\mathcal{S}$) issuing blind signatures to the voters' (taking the role of the user $\mathcal{U}$) public keys (of the GOST R 34.10-2012 signature scheme). This means that the Registrar does not get to know any information about the voters' public keys which keeps the voters anonymous. In the following, $H\colon \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ denotes a public hash function [2].

[5]The public key Q is an elliptic curve point which satisfies $d \cdot P = Q$.

Key-Generation Algorithm Gen

Algorithm 6.7 RSA Key-Generation Algorithm Gen [2, 3, 7, 30]

```
Input: Security parameter $1^n$
Output: Public key $(N, e)$, private key $(N, d)$
1: Choose two random primes $p, q$ where $p \neq q, |p_2| = |q_2| = n$
2: Compute $N = p \cdot q$
3: Choose $e \in \mathbb{Z}^*_{\Phi(N)}$                    // $\Phi(N) = (p-1) \cdot (q-1)$
4: Compute $d = e^{-1} \bmod \Phi(N)$
5: Return $(N, e), (N, d)$
```

Signing Algorithm $\langle \mathcal{S}, \mathcal{U} \rangle$

Algorithm 6.8 RSA Signing Algorithm $\langle \mathcal{S}, \mathcal{U} \rangle$ [2]

```
Input: $\mathcal{S}$: Private key $(N, d)$
Input: $\mathcal{U}$: Public key $(N, e)$, message $m$
Output: $\mathcal{U}$: Blind signature $\sigma$
1: $\mathcal{U}$: Choose random $b \in \mathbb{Z}^*_N$
2: $\mathcal{U}$: $m' = b^e \cdot H(m) \bmod N$
3: $\mathcal{U}$ sends $m'$ to $\mathcal{S}$
4: $\mathcal{S}$: $\sigma' = m'^d \bmod N$
5: $\mathcal{S}$ sends $\sigma'$ to $\mathcal{U}$
6: $\mathcal{U}$: $\sigma = b^{-1} \cdot \sigma' \bmod N$
7: Return $\mathcal{U}$: $\sigma$
```

Verification Algorithm Vrfy

Algorithm 6.9 RSA Verification Algorithm Vrfy [2]

```
Input: Public key $(N, e)$, message $m$, blind signature $\sigma$
Output: 1 if blind signature is valid, 0 if blind signature is invalid
1: if $\sigma^e = H(m) \bmod N$ then
2:     Return 1
3: else
4:     Return 0
5: end if
```

6.4.5 Commitment Scheme

This subsection illustrates the HMAC_GOSTR3411_2012_256 commitment scheme which is utilized for computing commitments on the Voters' SNILS codes by the Registrar. In order to specify the commitment scheme, we essentially need to define the key-generation algorithm Gen and the commitment algorithm Com (see Section 3.7). The computation of commitments is based on the HMAC transformation which uses the GOST R 34.11-2012 hash function with 256-bit output (see Subsection 6.4.1), denoted by H [46].

The following information on the key-generation algorithm as well as the commitment algorithm is based on [34], where L = 32 and B = 64 [46].

Key-Generation Algorithm Gen

Note that [34] does not define an algorithm Gen for generating a commitment key ck. The only information given on ck is that it has a minimum (recommended) length of L bytes and a maximum length of B bytes. In case applications use commitment keys longer than B bytes, ck is hashed first. Then, the actual key is the resultant L byte string.

Commitment Algorithm Com

The commitment algorithm Com uses two fixed strings denoted by ipad and opad:

$$\text{ipad} = \text{byte 0x36 repeated } B \text{ times}$$
$$\text{opad} = \text{byte 0x5C repeated } B \text{ times}$$

The commitment com is computed as com $= H(ck$ XOR opad, $H(ck$ XOR ipad, $m))$ where XOR is the bitwise exclusive-OR. More precisely, the following algorithm is executed to compute com.

Algorithm 6.10 HMAC_GOSTR3411_2012_256 Commitment Algorithm Com [34]

```
Input: Commitment key $ck$, message $m$
Output: Commitment com
1: $ck_{\text{extended}}$ := append zeros to the end of $ck$ to create a $B$ byte string[6]
2: $\text{tmp}_1 := ck_{\text{extended}}$ XOR ipad
3: $\text{tmp}_2$ := append $m$ to $\text{tmp}_1$
4: $\text{tmp}_3 := H(\text{tmp}_2)$
5: $\text{tmp}_4 := ck_{\text{extended}}$ XOR opad
6: $\text{tmp}_5$ := append $\text{tmp}_3$ to $\text{tmp}_4$
7: com := $H(\text{tmp}_5)$
8: Return com
```

6.4.6 Key Sharing

The Russian federal remote e-voting scheme makes use of key sharing to split the Organiser's secret key S_{org} into shares. In order to realize key sharing, the following (k, n) threshold scheme (see Section 3.8) by Shamir as described in [45] is used. Since the scheme generally specifies how to divide secret data D, D also comprises secret keys such as S_{org}. The presented (k, n)

[6]For example, if ck is of length 20 bytes and $B = 64$, then ck will be appended with 44 zero bytes 0x00.

threshold scheme relies on polynomial interpolation. Given k points $(x_1, y_1), \ldots, (x_k, y_k)$ in the two-dimensional space with distinct x_i's, then there exists only a single polynomial $q(x)$ of degree $k - 1$ satisfying $q(x_i) = y_i$ for all $1 \leq i \leq k$. Without loss of generality, the data D is assumed to be (or be convertible into) a number. In order to split up the data D into n pieces D_i, a random $k - 1$ degree polynomial $q(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1}$ with $a_0 = D$ is picked. Afterwards, $q(x)$ is evaluated at n points:

$$D_1 = q(1), \ldots, D_n = q(n)$$

Now, we have divided our secret data D into n pieces. Next, we describe how to reconstruct D given any subset of k D_i pieces together with their corresponding indices. For reconstructing D, we essentially need to find the coefficients of $q(x)$. This can be done by interpolation. As all k points (i, D_i) are on $q(x)$ by construction and given those k points, the polynomial of degree $k - 1$ (such that every point is on the polynomial) is unique, we actually receive $q(x)$ and thus its coefficients through interpolation. Afterwards, we can just evaluate $q(0)$ since $q(0) = a_0 = D$ by construction. Note that on the other hand, the knowledge of any $k - 1$ D_i pieces is not sufficient for calculating the secret data D. In order to make this claim more precise, Shamir illustrates sharing of D with modular arithmetic instead of real arithmetic. Therefore, the set of integers modulo a prime number p is used. This set forms a field in which interpolation is possible. Given an integer valued data D, a prime $p > D, n$ is picked. At first, the coefficients $a_1, \ldots, a_{k-1}$ in the polynom $q(x)$ are randomly chosen from a uniform distribution over the integers in $[0, p)$. Next, the pieces $D_1, \ldots, D_n$ are computed modulo p. Now, we assume that an adversary gets access to $k - 1$ D_i pieces. There are a total of p possible values for D, since $D \in [0, p)$. For every possible value D', the adversary is able to construct only a single polynomial $q'(x)$ of degree $k - 1$ satisfying $q'(0) = D'$ as well as $q'(i) = D'_i$ for all $1 \leq i \leq k - 1$. As all p possible polynomials are equally likely by construction, the adversary can not deduce anything about the actual value of our secret data D.

6.4.7 Non-Interactive Zero-Knowledge Proofs

This subsection provides a description of the NIZKPs utilized in the Russian federal remote e-voting scheme. The utilized NIZKPs are the Chaum-Pedersen proof as well as the disjunctive Chaum-Pedersen proof, each consisting of a proof generation executed by the prover P and a proof verification executed by the verifier V. They are both exemplified on the basis of the exponential ElGamal encryption scheme that is described first.

Exponential ElGamal Encryption Scheme

In the following, we illustrate the exponential ElGamal encryption scheme based on $\mathbb{Z}_p^*$ (with p prime) and a generator $g \in \mathbb{Z}_p^*$. Therefore, we specify the key-generation algorithm Gen, the encryption algorithm Enc and the decryption algorithm Dec of the encryption scheme.

Algorithm 6.11 Exponential ElGamal Key-Generation Algorithm Gen [52, 54]

```
Input: Security parameter $1^n$
Output: Public key $y$, private key $x$
1: Choose random $x \in \mathbb{Z}_p^*$
2: $y := g^x$
3: Return $(y, x)$
```

Algorithm 6.12 Exponential ElGamal Encryption Algorithm Enc [52, 54]

```
Input: Public key $y$, plaintext $m$
Output: Ciphertext $(r, s)$
1: Choose random $k \in \mathbb{Z}_p^*$
2: $r := g^k$
3: $s := g^m \cdot y^k$
4: Return $(r, s)$
```

Algorithm 6.13 Exponential ElGamal Decryption Algorithm Dec [52, 54]

```
Input: Private key $x$, ciphertext $(r, s)$
Output: Plaintext $m$
1: $g^m := \frac{s}{r^x}$
2: $m := \log_g(g^m)$
3: Return $m$
```

Chaum-Pedersen Proof

The Chaum-Pedersen proof, introduced by Chaum and Pedersen in [10] is a proof of equality of discrete logarithms. This means that the Chaum-Pedersen proof enables a prover P to prove to a verifier V that

$$\log_g(x) = \log_h(y)$$

where the P knows a such that $g^a = x$ and $h^a = y$ [15, 44]. In the Russian federal remote e-voting scheme, the Chaum-Pedersen proof is used by the Decryptor in order to prove the correctness of (partial) decryptions of some ciphertexts using the private key. To prove the correctness of decryption of an exponential ElGamal ciphertext (r, s) using the private key x, P proves the following equality of discrete logarithms [40].

$$\log_g(y) = \log_r\,(s/g^m)$$

The left discrete logarithm directly follows from the generation of the public key since $g^x = y$. The right discrete logarithm is shown below.

$$\begin{aligned} r^x = \frac{s}{g^m} &\iff \left(g^k\right)^x = \frac{g^m \cdot y^k}{g^m} \\ &\iff (g^x)^k = y^k \\ &\iff y^k = y^k \end{aligned}$$

In the following, we present the Chaum-Pedersen proof generation as well as the Chaum-Pedersen proof verification[7]. Since the proof of correctness of decryption is sent together with the decryption, we added the decryption (plaintext m) to the input of the two algorithms. Note that the prover additionally has access to the secret key x [12].

Algorithm 6.14 Chaum-Pedersen Proof Generation [5, 6, 31]

```
Input: Ciphertext $(r, s)$, plaintext $m$
Output: Proof $(a, b, f)$
1: Choose random $j \in \mathbb{Z}_p^*$
2: $a := g^j$
3: $b := r^j$
4: $c := h(a, b, r, s/g^m)$
5: $f := j + c \cdot x$
6: Return $(a, b, f)$
```

Algorithm 6.15 Chaum-Pedersen Proof Verification [5, 6, 31]

```
Input: Proof $(a, b, f)$, ciphertext $(r, s)$, plaintext $m$
1: $c' := h(a, b, r, s/g^m)$
2: $g^f \overset{!}{=} a \cdot y^{c'}$
3: $r^f \overset{!}{=} b \cdot (s/g^m)^{c'}$
```

Now, we show why the two equalities in the proof verification should hold in case of correct decryption.

$$g^f = g^{j+c\cdot x} = g^j \cdot g^{cx} = a \cdot (g^x)^c = a \cdot \left(g^{\log_g(y)}\right)^c = a \cdot y^c = a \cdot y^{c'}$$
$$r^f = r^{j+c\cdot x} = r^j \cdot r^{cx} = b \cdot (r^x)^c = b \cdot \left(r^{\log_r(s/g^m)}\right)^c = b \cdot (s/g^m)^c = b \cdot (s/g^m)^{c'}$$

Disjunctive Chaum-Pedersen Proof

Disjunctive proofs generally allow a prover P to prove to a verifier V that one of two statements holds, where it is kept secret which one is correct [5]. In terms of the Chaum-Pedersen proof, the disjunctive version allows proving that one of two equalities of discrete logarithms holds. Concerning encryption of a message m, the disjunctive Chaum-Pedersen proof can be used to prove that one of two messages has been encrypted, i. e., $m \in \{m_0, m_1\}$. For the encryption with the exponential ElGamal encryption scheme, the disjunctive version of the proof looks as follows

$$\log_g(r) = \log_y(s/g^{m_0}) \lor \log_g(r) = \log_y(s/g^{m_1})$$

[7]The proof generation and the proof verification use a hash function h. Whereas the presented Chaum-Pedersen proof is non-interactive, the original Chaum-Pedersen proof is interactive. The proof can be made non-interactive by using what is called the Fiat-Shamir transformation [5, 24] which comes along with the hash function h. Note that depending on the used form of the Fiat-Shamir transformation, the content of h differs. This should just be seen as a hint. It is not of importance here as we do not focus on the content of the hash function. The same applies to the disjunctive Chaum-Pedersen proof.

where the prover either knows a witness for the left or the right equality of discrete logarithms, but not for both equalities simultaneously [15].

In the Russian federal remote e-voting scheme, the disjunctive Chaum-Pedersen proof is used by the Voter in order to prove that either $m = 0$ or $m = 1$ has been encrypted, i. e., $m \in \{0, 1\}$. For this purpose, the Voter proves that one of the following two equalities of discrete logarithms holds.

$$\log_g(r) = \log_y\left(s/g^1\right) \vee \log_g(r) = \log_y\left(s/g^0\right)$$

The left equality holds in case of encrypting $m = 1$ whereas the right equality holds in case of encrypting $m = 0$. The witness that one of the two equalities holds is the random value k used for the encryption of a message m with the exponential ElGamal encryption scheme. The left side of each equality directly follows from how the first part of the ciphertext (r, s) is computed since $g^k = r$. The right side of each equality is shown below.

$$\begin{aligned} y^k = \frac{s}{g^1} &\iff (g^x)^k = \frac{g^1 \cdot y^k}{g^1} \\ &\iff y^k = y^k \\ y^k = \frac{s}{g^0} &\iff (g^x)^k = \frac{g^0 \cdot y^k}{g^0} \\ &\iff y^k = y^k \end{aligned}$$

In the following, we present the disjunctive Chaum-Pedersen proof generation and the disjunctive Chaum-Pedersen proof verification. There is one proof generation for the encryption of $m = 1$ (see Algorithm 6.16) and one proof generation for the encryption of $m = 0$ (see Algorithm 6.17). Each proof generation consists of two parts. More precisely, the proof generation consists of the $m = 1$ or the $m = 0$ part as well as of a simulation of the contrary part [31]. If we proof that $m = 1$ has been encrypted, then the $m = 0$ part of the proof needs to be simulated. Note that both proofs, i. e., the proof for $m = 0$ and the proof $m = 1$ are validated using the same algorithm.

Algorithm 6.16 Disjunctive Chaum-Pedersen Proof Generation [5, 31]

```
Input: Ciphertext $(r, s)$
Output: Proof $(a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$
1: Choose random $j, c_0, f_0 \in \mathbb{Z}_p^*$
2: $a_0 := g^{f_0}/r^{c_0}$
3: $b_0 := y^{f_0}/s^{c_0}$
4: $a_1 := g^j$
5: $b_1 := y^j$
6: $c := h(r, s, a_0, b_0, a_1, b_1)$
7: $c_1 := c - c_0$
8: $f_1 := j + c_1 \cdot k$
9: Return $(a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$
```

Algorithm 6.17 Disjunctive Chaum-Pedersen Proof Generation [5, 31]

```
Input: Ciphertext $(r, s)$
Output: Proof $(a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$
1: Choose random $j, c_1, f_1 \in \mathbb{Z}_p^*$
2: $a_1 := g^{f_1}/r^{c_1}$
3: $b_1 := y^{f_1}/(s/g)^{c_1}$
4: $a_0 := g^j$
5: $b_0 := y^j$
6: $c := h(r, s, a_0, b_0, a_1, b_1)$
7: $c_0 := c - c_1$
8: $f_0 := j + c_0 \cdot k$
9: Return $(a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$
```

Algorithm 6.18 Disjunctive Chaum-Pedersen Proof Verification [5, 31]

```
Input: Proof $(a_0, b_0, a_1, b_1, c_0, c_1, f_0, f_1)$, ciphertext $(r, s)$
1: $c' := h(r, s, a_0, b_0, a_1, b_1)$
2: $c_0 + c_1 \overset{!}{=} c'$
3: $g^{f_0} \overset{!}{=} a_0 \cdot r^{c_0}$
4: $g^{f_1} \overset{!}{=} a_1 \cdot r^{c_1}$
5: $y^{f_0} \overset{!}{=} b_0 \cdot s^{c_0}$
6: $y^{f_1} \overset{!}{=} b_1 \cdot (s/g)c_1$
```

Now, we show why the five equalities in the proof verification should hold in case either $m = 0$ or $m = 1$ has been encrypted, i. e., $m \in \{0, 1\}$. At first, we consider the proof verification in case of encrypting $m = 1$, where we have that $\log_g(r) = k = \log_y(s/g)$.

$$\begin{aligned}
c_0 + c_1 &= c_0 + (c - c_0) = c = c' \\
g^{f_0} &= a_0 \cdot r^{c_0} \\
g^{f_1} &= g^{j+c_1\cdot k} = g^j \cdot g^{c_1\cdot k} = a_1 \cdot \left(g^k\right)^{c_1} = a_1 \cdot \left(g^{\log_g(r)}\right)^{c_1} = a_1 \cdot r^{c_1} \\
y^{f_0} &= b_0 \cdot s^{c_0} \\
y^{f_1} &= y^{j+c_1\cdot k} = y^j \cdot y^{c_1\cdot k} = b_1 \cdot \left(y^k\right)^{c_1} = b_1 \cdot \left(y^{\log_y(s/g)}\right)^{c_1} = b_1 \cdot (s/g)^{c_1}
\end{aligned}$$

Next, we consider the proof verification in case of encrypting $m = 0$, where we have that $\log_g(r) = k = \log_y(s)$.

$$\begin{aligned}
c_0 + c_1 &= (c - c_1) + c_1 = c = c' \\
g^{f_0} &= g^{j+c_0\cdot k} = g^j \cdot g^{c_0\cdot k} = a_0 \cdot \left(g^k\right)^{c_0} = a_0 \cdot \left(g^{\log_g(r)}\right)^{c_0} = a_0 \cdot r^{c_0} \\
g^{f_1} &= a_1 \cdot r^{c_1} \\
y^{f_0} &= y^{j+c_0\cdot k} = y^j \cdot y^{c_0\cdot k} = b_0 \cdot \left(y^k\right)^{c_1} = b_0 \cdot \left(y^{\log_y(s)}\right)^{c_1} = b_0 \cdot s^{c_1} \\
y^{f_1} &= b_1 \cdot (s/g)^{c_1}
\end{aligned}$$

The Russian federal remote e-voting scheme makes use of another ZKRP associated with the encrypted vote. More precisely, it uses a proof to affirm that the sum of all encrypted values for each option on the ballot does not exceed the bound d where d denotes the maximum number of options that each Voter can choose. As described above, the disjunctive Chaum-Pedersen proof can not only be used to prove that either $m = 0$ or $m = 1$ has been encrypted, but can also be used to prove that one of two arbitrary messages has been encrypted. Suppose that we want to prove that either $m = 1$ or $m = 2$ has been encrypted, then the disjunctive version of the proof looks as follows.

$$\log_g(r) = \log_y\left(s/g^1\right) \vee \log_g(r) = \log_y\left(s/g^2\right)$$

The proof generation for $m = 2$ would look the same as the proof generation for $m = 0$. However, line 3 in the proof generation for $m = 1$ would change to $b_0 := y^{f_0}/(s/g^2)^{c_0}$. For the proof verification, line 5 would change to $y^{f_0} \overset{!}{=} b_0 \cdot (s/g^2)^{c_0}$. In order to prove that the aggregated value that we get when summing up all encrypted values for each option on the ballot lies in the range $[0, d]$, disjunctive Chaum-Pedersen proofs are combined. An intuition on how this could be realized can be found in [16].

7 Security Analysis

This chapter deals with the security analysis of the Russian federal remote e-voting scheme specified in [50] w.r.t. verifiability as defined in Section 5.2. In order to perform a security analysis w.r.t this definition of verifiability, we need to do the preliminary work described next. At first, we have to define the assumptions underlying our analysis. This includes assumptions about the channels used for communication, assumptions about the security of the used cryptographic primitives as well as trust assumptions indicating which of the protocol participants are honest and which can be corrupted by an adversary. In general, it is preferable to make as less assumptions and assume as less protocol participants to be honest as possible. The reason is that this allows for stronger guarantees concerning verifiability. The definition of the assumptions has a big impact on the result of the security analysis since even small changes in the assumptions could lead to different results. In the next step, we have to define the goal γ, i. e., specify the conditions that need to be satisfied such that a protocol run is considered correct in some protocol-specific sense. As the verifiability definition is centered around the notion of a goal [14], the result of the analysis depends strongly on the selected goal. Since the definition of verifiability requires the probability that a run r which does not meet the goal γ (i. e., $r \notin \gamma$) is still accepted by the judge J to be δ-bounded, we have to define the judging procedure, determining whether a run is accepted by J or not. Based on the assumptions, the goal as well as the judging procedure, we finally perform the security analysis.

7.1 Assumptions

The performed security analysis will be based on the following assumptions including assumptions about the channels used for communication, assumptions about the guarantees of the cryptographic primitives as well as trust assumptions about the protocol participants. In the following, a valid voting transaction is defined as a voting transaction j where all included range proofs are correct and both signatures σ_{b_j} and σ_{v_j} are valid. This definition will be used throughout the whole chapter.

(A1) The following authenticated channels[1] are assumed to exist.

All protocol participants have unilaterally authenticated channels to the Blockchain B. This makes sure that when accessing B, all protocol participants have the same view on the Blockchain B [35].

There are authenticated channels between every Voting Device and the Vote Collector.

[1] In [11], Coretti et al. define a channel as a resource involving a sender, a receiver and an attacker that enables the sender to transmit messages to the receiver. They specify an authenticated channel as a channel that allows the adversary to read, forward or delete messages. In this work, an authenticated channel refers to the authenticated channel from Coretti et al. but does not allow the adversary to delete messages, i. e., the adversary is limited to reading and forwarding the messages. Note that we abstract away from how the authenticated channels are established.

(A2) The public-key encryption scheme is correct, i. e., $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$.

Since for verifiability, we are not interested in whether the adversary can find out how a Voter has voted, however require that the adversary is not able to alter or drop voting transactions, CPA-security is not necessary. Thus, it is sufficient to assume that the public-key encryption scheme is correct.

(A3) The digital signature scheme is (EUF-CMA)-secure.

In case we would not assume the digital signature scheme to be (EUF-CMA)-secure, this would allow the adversary to alter the Voter's voting transaction. More precisely, the adversary could change the encrypted options $(c_j)_i$ together with the corresponding ZKRPs of the Voter's voting transaction and still produce a valid signature (w.r.t. the Voter's public key pk_{v_j}) for the manipulated voting transaction. The described manipulation would remain unnoticed since the voting transaction would be valid and the Voter's public key pk_{v_j} has not been changed. Thus, we assume the digital signature scheme to be (EUF-CMA)-secure.

(A4) The blind digital signature scheme satisfies unforgeability.

Without the assumption that the blind digital signature scheme is unforgeable, this would give the adversary the possibility to alter the Voter's voting transaction. More precisely, the adversary could forge a valid signature $\sigma_{b_{\text{forged}}}$ (w.r.t. the Registrar's public key pk_b) on its own public key pk_{adv}. When corrupting the Vote Collector, the adversary can alter the Voter's voting transaction by changing the encrypted votes $(c_j)_i$, adapting the range proofs accordingly, changing the Voter's public key pk_{v_j} to pk_{adv} and changing σ_{b_j} to $\sigma_{b_{\text{forged}}}$. The adversary would then sign the altered voting transaction with its secret key and upload it to the Blockchain. As both signatures $\sigma_{b_{\text{forged}}}$ and $\sigma_{v_{\text{adv}}}$ would be valid, this voting transaction would not be marked as invalid and thus be reflected in the published election result. This manipulation could only be noticed if the Voter would check if her voting transaction has been added to the Blockchain.

The blindness property does not need to be ensured. This is because we are not interested in whether it is feasible for a malicious signer (i. e., the Registrar) to determine which of two public keys pk_{v_1} and pk_{v_2} has been signed first. In case the malicious signer would be able to do so, the anonymity of the Voter would not be guaranteed anymore which is however not of relevance for verifiability.

(A5) All NIZKPs have to fulfill completeness and soundness.

The zero-knowledge property does not need to be fulfilled. As for verifiability, we are not interested in preventing the adversary from knowing the content of the Voter's ballot, it does not matter whether the adversary gains any information about the statement to be proven (except that the statement is true) when verifying the NIZKPs.

In case completeness would not be satisfied, an honest Voter would not be able to convince the verifier that for each option she only encrypted 0 or 1 and that she did not choose more than d options. Additionally, an honest Decryptor could not convince the verifier that the partial decryptions are correct. As none of this is considered malicious behavior and it has to be ensured that it is possible to convince the verifier in case of correct behavior, we assume completeness to be satisfied.

In case soundness would not be satisfied, a dishonest Voter could convince the verifier that for each option she only encrypted 0 or 1 and that she did not choose more than d options, although she did for example encrypt 2 or has chosen more than d options. This would allow a dishonest Voter to deviate from the protocol specification without being detected. As this is considered malicious behavior and it has to be ensured that it is not possible to convince the verifier in case of such behavior, we assume soundness to be satisfied.

(A6) The Scheduler S, the Judge J, the Blockchain B, the Registrar R and n_h Voters V_i are honest, i. e.,

$$\varphi = hon(S) \wedge hon(J) \wedge hon(B) \wedge hon(R) \bigwedge_{i=1}^{n_h} hon(V_i)$$

Honesty of Scheduler S:

For the security analysis, we consider the Organiser O to be part of the Scheduler S. As described in Section 5.1, the Scheduler S acts as the master program of the protocol process and is responsible for triggering the protocol participants in the proper order. This also includes triggering the Judge. In case of corrupting S, the adversary could for example prevent the Scheduler from triggering J. Since the Organiser is coordinating the e-voting process and is generating and publishing the election parameters, it is reasonable to assume O to be honest. Thus, the Scheduler S (including the Organiser O) is assumed to be honest.

Note that the Scheduler S does not exist in real systems and is rather a modeling tool [35].

Honesty of Judge J:

Obviously, the Judge J necessarily has to be honest since J is the one accepting or rejecting a protocol run.

Honesty of Blockchain B:

The Blockchain B stores public information that is used by J for performing the judging procedure, among other things. Since we want this information to be stored correctly and want every protocol participant to have the same view on the Blockchain and thus on the stored information, the Blockchain B is assumed to be honest.

Honesty of Registrar R:

The reason why the Registrar is assumed to be honest is similar to the reason why the blind digital signature scheme is assumed to be unforgeable. In case of corrupting R, the adversary would not have to forge a signature $\sigma_{b_{\text{forged}}}$ on its public key pk_{adv}, but could request the Registrar to issue a signature s on its masked public key and then remove the mask from s in order to get a valid signature $\sigma_{b_{\text{adv}}}$ on its public key. Afterwards, the adversary would proceed in the same way as described in the justification for the assumption of the unforgeability.

Additionally, if we would allow the adversary to corrupt the Registrar, the VoterList could be manipulated such that also Voters that are not eligible to vote are included and signatures s could also be issued to Voters that are not eligible to vote. As we do not permit voting transactions of Voters that are not eligible to vote to be reflected in the published election result, R is assumed to be honest.

Honesty of Voters V_i:

It would not make sense at all to allow the adversary to corrupt all Voters. This is why there are n_h honest Voters V_i.

Note that here, we do not distinguish between honesty and dishonesty of Voter and Voting Device, i. e., honesty/dishonesty of a Voter is equal to the honesty/dishonesty of the corresponding Voting Device. In case of allowing honest Voters but dishonest Voting Devices, there are means as for example Voter Verification Devices that enable to check that the Voting Device behaved correctly [35].

7.2 Goal

In order to specify the goal γ, one of the central questions is whether we want to provide guarantees for those honest Voters who performed their checks successfully or not. Since there are checks where cheating might not be detected (with some probability) as already mentioned in Section 5.3, we consider this guarantee to be too severe. This means that we do not treat votes of honest Voters successfully performing their checks different to votes of honest Voters who did not perform their checks (successfully). Thus, we want to limit the deviation from the ideal result to the actual result. As for the deviation, we consider a different treatment of dropping and altering votes to be meaningful, we ended up with the following definition of a goal that corresponds to the quantitative goal from Section 5.4.

Let r be a run of an e-voting protocol with n eligible Voters. Let n_h denote the number of honest Voters in r and $n_d = n - n_h$ denote the number of dishonest Voters in r. The actual choices of the honest Voters in this run are denoted by $c_1, \dots, c_{n_h}$. Then, the goal $\gamma(k, \varphi)$ is fulfilled in a protocol run r (i. e., $r \in \gamma(k, \varphi)$) if either (a) the trust assumptions φ are not met in r, or if (b) φ is met in r and there exist valid choices $c'_1, \dots, c'_{n_d}$ (choices of dishonest Voters) and $\tilde{c}_1, \dots, \tilde{c}_n$ such that the following conditions are satisfied:

(i) An election result is published in r which is equal to $\rho(\tilde{c}_1, \dots, \tilde{c}_n)$.

(ii) $d((c_1, \dots, c_{n_h}, c'_1, \dots, c'_{n_d}), (\tilde{c}_1, \dots, \tilde{c}_n)) \leq k$.

Since for the definition of verifiability, we are interested in scenarios where the goal is not met in a protocol run, we describe those scenarios subsequently.

The goal $\gamma(k, \varphi)$ is not satisfied in a run r (i. e., $r \notin \gamma(k, \varphi)$) if the trust assumptions φ are met in r and there exist valid choices $c'_1, \dots, c'_{n_d}$ and $\tilde{c}_1, \dots, \tilde{c}_n$ such that at least one of the following conditions holds true:

(i) No election result is published in r.

(ii) An election result is published in r but is not equal to $\rho(\tilde{c}_1, \dots, \tilde{c}_n)$.

(iii) $d((c_1, \ldots, c_{n_h}, c'_1, \ldots, c'_{n_d}), (\tilde{c}_1, \ldots, \tilde{c}_n)) > k$.

7.3 Judging Procedure

In this section, we describe the judging procedure that is used for the security analysis of the Russian federal remote e-voting system. In the following, a denotes the number of options in each ballot (denoted by n in Section 6.3) and n denotes the number of eligible Voters.

For the security analysis, as already specified above, we assume that the Judge J is honest. The honest program of J, denoted by π_J solely uses information that is publicly available. This means that each protocol participant is able to run the judging procedure. The Scheduler S triggers the program π_J. Upon being triggered, π_J reads data from the Blockchain and verifies its correctness. In the following situations, J outputs reject on a distinct tape:

(J1) If at least one of the proofs of correctness of partial decryptions is not correct.

In case at least one of the proofs of correctness of partial decryptions is not correct, the Decryptor did not compute correct partial decryptions $(R_i)_t$ for each option $i \in \{0, \ldots, a-1\}$. This would lead to an election result being published that does not correspond to the expected election result $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$ since incorrect partial decryptions $(R_i)_t$ lead to incorrect final decryptions M_i. Thus, the Judge should reject this protocol run.

(J2) If there is more than one valid voting transaction for a Voter's public key pk_j.

In case there is more than one valid voting transaction for a Voter's public key pk_j, the published election result would reflect more than one valid ballot cast by the Voter with public key pk_j. This would lead to an election result being published that does not correspond to the expected election result $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$ as ballot stuffing is not permitted, and therefore the published election result is expected to reflect only one valid cast ballot per Voter. Thus, the Judge should reject this protocol run.

(J3) If the number of valid voting transactions exceeds the number of issued signatures s.

If the number of valid voting transactions exceeds the number of issued signatures s, it was either managed to forge a valid signature σ_b (whose generation usually requires s) or a valid signature σ_b was reused. Since the Registrar R is assumed to be honest, signatures s are only issued to eligible Voters and only once per eligible Voter. As the published election result is expected to reflect only valid cast ballots of eligible Voters as well as only one valid cast ballot per eligible Voter, this would lead to an election result being published that does not correspond to the expected election result $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$. Thus, such a protocol run should be rejected.

(J4) If the result of the aggregation of encrypted votes is not equal to the correct and expected result for at least one option, i. e., if there exists an option $i \in \{0, \ldots, a-1\}$ such that

$$(R_i, C_i) \neq \sum_{j=1}^{V} (c_j)_i$$

where V is the total number of valid voting transactions.

In order to check whether the result of the aggregation of encrypted votes is equal to the correct and expect result for each option, the Judge proceeds as described in the following. At first, the two signatures σ_{b_j} and σ_{v_j} are verified for each voting transaction j contained in the Blockchain. In the next step, the range proofs, i. e., the NIZKPs for showing that $b_i \in \{0, 1\}$ and $\sum_{i=0}^{a-1} b_i \leq d$ only for those voting transactions where both signatures are valid, are verified. Now, the valid voting transactions are used in order to perform the aggregation of encrypted votes as the sum described above.

In case the result of the aggregation of encrypted votes is not equal to the correct and expected result for at least one option, the Decryptor did not compute correct aggregations (R_i, C_i) for each option $i \in \{0, \dots, a-1\}$. There are several cases leading to incorrect aggregations. At first, votes $(c_j)_i$ of invalid voting transactions could have been included. Another case is that votes of valid voting transactions could have been excluded. Finally, arbitrary votes could have been added. All these three cases would lead to an election result being published that does not correspond to the expected election result $\rho(\tilde{c}_1, \dots, \tilde{c}_n)$ which is why the J should reject this protocol run.

(J5) If the result of the final decryption of aggregated votes is not equal to the correct and expected result for at least one option, i. e., if there exists an option $i \in \{0, \dots, a-1\}$ such that

$$M_i \neq C_i - H(Q_t || Q_{org}) \cdot S_{org} \cdot R_i - H(Q_{org} || Q_t) \cdot (R_i)_t.$$

In case the result of the final decryption of aggregated votes is not equal to the correct and expected result for at least one option, the Decryptor did not compute correct final decryptions M_i for each option $i \in \{0, \dots, a-1\}$. This would lead to an election result being published that does not correspond to the expected election result $\rho(\tilde{c}_1, \dots, \tilde{c}_n)$. Thus, such a protocol run should be rejected.

(J6) If there are valid voting transactions, but no election result is published.

Obviously, if there are valid voting transactions, an election result is expected to be published. If this is not the case, the Judge should reject this protocol run.

If none of these situations occur, then J outputs accept on a distinct tape.

7.4 Verifiability Analysis

In this section, we perform the security analysis of the Russian federal remote e-voting protocol w.r.t. verifiability. Therefore, we consider attacks that the adversary could perform in order to manipulate a protocol run r such that the goal $\gamma(k, \varphi)$ is not satisfied. For each of those attacks, we investigate whether the manipulated protocol run $r \notin \gamma(k, \varphi)$ is accepted or rejected by the Judge J.

In the Russian federal remote e-voting system, d denotes the maximum number of options that each Voter can select. In this analysis, we only consider single choice (i. e., $d = 1$). Multiple choice (i. e., $d > 1$) is not considered here since it is more complex to capture the result of the analysis as the probability $\Pr[\pi^{(l)} \mapsto \neg\gamma(k, \varphi), (J\colon \text{accept})]$ depending on k. This is due to the fact that in case of multiple choice, the probability depends on the exact Voters' voting transactions in a certain protocol run. However, the presented attacks to manipulate a protocol run are similar.

In order to manipulate a protocol run r such that the goal $\gamma(k, \varphi)$ is not satisfied, the adversary can perform the following attacks:

(i) The adversary could attempt that no election result is published in r at all, although there exist valid voting transactions.

In order to achieve this, the adversary could corrupt the Decryptor and thereby hinder the Decryptor to publish the final decryptions of aggregated votes M_i. This would mean that no election result is published even though there are valid voting transactions. Due to (J6), the Judge would reject this protocol run.

(ii) The adversary could attempt to achieve an election result being published which is unequal to the expected election result $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$.

The adversary could achieve this by corrupting the Decryptor and performing at least one of the following actions:

- Forge the result of the aggregation of encrypted votes by computing incorrect aggregations (R_i, C_i) for at least one option. More precisely, the adversary could either include votes $(c_j)_i$ of invalid voting transactions, exclude votes of valid voting transactions or add arbitrary votes to the aggregation of encrypted votes. Due to (J4), such a protocol run would be rejected by the Judge.
- Compute incorrect partial decryptions $(R_i)_t$ for at least one option. Due to (J1), such a protocol run would be rejected by the Judge.
- Forge the result of the final decryption of aggregated votes by computing incorrect final decryptions M_i for at least one option. Due to (J5), such a protocol run would be rejected by the Judge.

Additionally, the adversary could perform ballot stuffing. Therefore, the adversary could either corrupt dishonest Voters and send the voting transaction more than once to the Vote Collector or corrupt the Vote Collector and upload voting transactions more than once to the Blockchain. After this attack, the Blockchain contains more than one voting transaction per Voter's public key. Due to (J2), such a protocol run would be rejected by the Judge.

To circumvent this, the adversary could create its own voting transaction by including its valid choices and forging a valid signature $\sigma_{b_{\text{adv}}}$ (w.r.t. the Registrar's public key pk_b) on its public key pk_{adv}. This voting transaction would be valid and thus reflected in the published election result meaning that the published election result is unequal to the expected one $\rho(\tilde{c}_1, \ldots, \tilde{c}_n)$ that only reflects one voting transaction per eligible Voter. As we assumed the blind digital signature scheme to satisfy unforgeability (Assumption (A4)), forging a valid signature $\sigma_{b_{\text{adv}}}$ is not possible. If it would be possible to forge $\sigma_{b_{\text{adv}}}$, the Judge would reject the protocol run due to (J3).

(iii) The adversary could attempt to either alter or drop voting transactions of honest Voters such that $d((c_1, \ldots, c_{n_h}, c'_1, \ldots, c'_{n_d}), (\tilde{c}_1, \ldots, \tilde{c}_n)) > k$. Clearly, this is dependent on the chosen parameter k. Hereinafter, we investigate whether one of these attacks could be performed by the adversary without the Judge rejecting the protocol run or not, more precisely the probability thereof.

For the investigation, we distinguish between the following two cases. In the first case, Voters do not have the possibility to publish a complaint that their voting transaction has not been published into the Blockchain, although they have cast a ballot and sent the voting transaction to the Vote Collector. In the second case, Voters have the possibility to publish such complaints. For instance, this case could be realized by the Vote Collector sending the Voter a signature on her voting transaction as it has been implemented in [35]. If such complaints can be published into the Blockchain, the judging procedure needs to be adapted, such that the complaints are taken into account as well. Note that we abstract away from the exact way the complaints are realized as well as from how the judging procedure has to be adapted. We only consider the fact that there are possibilities to implement the second case and that the complaints allow the Judge to detect manipulations in which voting transactions of honest Voters have been dropped. Detailed information on the realization can be obtained from [35].

We start by considering the first case. In this case, Voters can not publish complaints if their voting transaction has not been published, even though they have cast a ballot and sent the voting transaction to the Vote Collector. The adversary could corrupt the Vote Collector and drop an arbitrary amount of voting transactions containing the actual choices of honest Voters by not uploading the voting transactions to the Blockchain. This leads to a deviation of the actual election result from the ideal election result since the dropped choices are not part of the actual election result. As the Judge does not have any possibility to detect that voting transactions containing the actual choices of honest Voters have been dropped and it is not possible for Voters to complain, this malicious behavior remains undetected and the corresponding protocol run is accepted. Independent of the chosen parameter k, there is an attack where the adversary drops $k + 1$ voting transactions of honest Voters such that $d((c_1, \dots, c_{n_h}, c'_1, \dots, c'_{n_d}), (\tilde{c}_1, \dots, \tilde{c}_n)) > k$ as the amount of voting transactions to drop without being detected is arbitrary. This means that $\Pr[\pi^{(l)} \mapsto \neg\gamma(k, \varphi), (J\colon \text{accept})]$ would be bounded by 1. Thus, in the first case, the Russian federal remote e-voting protocol does not provide verifiability at all.

Next, we investigate the second case, i. e., Voters have the possibility to publish complaints if their voting transaction has not been published, even though they have cast a ballot and sent the voting transaction to the Vote Collector. In the following, let $k = 0$. This means that the security analysis w.r.t verifiability is based on the goal $\gamma(0, \varphi)$. Thus, for a protocol run to be accepted, among other things, the distance between the ideal and the actual election result has to be 0, i. e., the actual election result has to be equal to the ideal one and thus, neither dropping nor altering votes is permitted. To manipulate the protocol run such that the goal $\gamma(0, \varphi)$ is not satisfied, the adversary has to achieve that the distance between ideal and actual election result is bigger than 0, i. e., $d((c_1, \dots, c_{n_h}, c'_1, \dots, c'_{n_d}), (\tilde{c}_1, \dots, \tilde{c}_n)) > 0$. However, the Judge should not be able to detect the manipulation. In general, the adversary has two options, either to alter or drop voting transactions of honest Voters.

In order to achieve the desired effect of altered voting transactions, i. e., that the choices made by honest Voters are replaced by different choices, the altered voting transaction has to remain valid. Otherwise, the altered choices would not be contained in the actual election result since the voting transactions are not valid. In case that they are not valid, altering of voting transactions has the same effect as dropping them. In order to alter a voting transaction of an honest Voter, the adversary could corrupt the Vote Collector and change the actual choices

$(c_j)_i$ to different valid ones. However, the adversary is not able to generate a signature $\sigma_{v_{\text{adv}}}$ for the altered voting transaction that is valid w.r.t. the Voter's public key pk_{v_j} (that is also contained in the voting transaction) since the digital signature scheme is assumed to be (EUF-CMA)-secure (assumption (A3)). This means that the altered voting transaction would not be valid and thus not reflected in the actual election result which makes the consequence of the attack equal to dropping voting transactions.

Another possibility to alter a voting transaction, without the necessity to generate a valid signature $\sigma_{v_{\text{adv}}}$ is to additionally replace the Voter's public key pk_{v_j} by the adversary's public key pk_{adv}. However, for the altered voting transaction to be valid, a valid signature $\sigma_{b_{\text{adv}}}$ (w.r.t. the Registrar's public key pk_b) on pk_{adv} is required. As the Registrar is assumed to be honest, the adversary would have to forge a valid signature $\sigma_{b_{\text{adv}}}$ such that the altered voting transaction would be valid and thus reflected in the actual election result. Due to assumption (A4), forging a valid signature $\sigma_{b_{\text{adv}}}$ is not possible. Thus, the altered voting transaction would not be valid. This means that the effect of this attack is equal to dropping voting transactions as well.

In case of dropping a voting transaction of an honest Voter, the adversary could corrupt the Vote Collector to do so. This only remains undetected if the Voter did not perform her checks. Otherwise, if the Voter has performed her checks and notices that her vote was not published into the Blockchain, she can publish a complaint which would enable the Judge to detect the manipulation. Let p_{check} denote the probability that a Voter performs her checks. As the Judge has no possibility to detect this manipulation without a complaint, $\Pr[\pi^{(l)} \mapsto \neg\gamma(0,\varphi), (J\colon \text{accept})]$ would be bounded by $1 - p_{\text{check}}$.

Thus, in the second case, the Russian federal remote e-voting protocol does provide verifiability with some probability. To be more precise, the investigated protocol is $(\gamma(0,\varphi), \delta_0(p_{\text{check}}))$-verifiable where $\delta_0(p_{\text{check}}) = 1 - p_{\text{check}}$.

Finally, we generalize the result of the analysis and keep k flexible instead of fixing it. In order to manipulate the protocol run such that the goal $\gamma(k,\varphi)$ is not met, the adversary has to achieve that $d((c_1,\ldots,c_{n_h},c'_1,\ldots,c'_{n_d}),(\tilde{c}_1,\ldots,\tilde{c}_n)) > k$. Therefore, the adversary could corrupt the Vote Collector and drop voting transactions of $k+1$ honest Voters. This manipulation would only remain undetected if none of the $k+1$ honest Voters did perform their checks. This implies that the probability $\Pr[\pi^{(l)} \mapsto \neg\gamma(k,\varphi), (J\colon \text{accept})]$ would be bounded by $(1 - p_{\text{check}})^{k+1}$. Thus, the Russian federal remote e-voting protocol does provide verifiability with some probability. To be more precise, the investigated protocol is $(\gamma(k,\varphi), \delta_k(p_{\text{check}}))$-verifiable where $\delta_k(p_{\text{check}}) = (1 - p_{\text{check}})^{k+1}$.

8 Conclusion and Outlook

In this chapter, we conclude by summarizing the work as well as the results gained from the security analysis. In addition, in the outlook, we discuss future work concerning the Russian federal remote e-voting system.

Conclusion

In this work, we have presented one of the two remote e-voting systems that were used in the Russian parliamentary elections of 2021, called the Russian federal remote e-voting system. Therefore, we introduced a description of the protocol that the remote e-voting system is based on and investigated the cryptographic primitives that are used in the protocol.

Finally, we have performed a security analysis of the Russian federal remote e-voting protocol w.r.t. verifiability as defined in the KTV framework. For this purpose, we determined assumptions about the used cryptographic primitives as well as assumptions about the honesty of the protocol participants. Since the KTV framework is centered around the notion of a goal γ as well as a Judge J, we defined a suitable goal specifying the conditions that need to be satisfied such that a protocol run is valid and defined a judging procedure that states the situations in which a certain protocol run should be rejected by the Judge in case the goal is not met.

For the analysis, we distinguished between the following two cases. In the first case, Voters do not have the possibility to publish complaints that their voting transaction has not been published into the Blockchain, although they have cast a ballot and sent the voting transaction to the Vote Collector. In the second case, Voters have the possibility to publish such complaints.

As a result of the security analysis w.r.t. verifiability, we found out that in the first case, the Russian federal remote e-voting protocol does not provide verifiability at all. In contrast, in the second case, for $k = 0$, the protocol is $(\gamma(0, \varphi), \delta_0(p_{\text{check}}))$-verifiable where $\delta_0(p_{\text{check}}) = 1 - p_{\text{check}}$ and p_{check} denotes the probability that a Voter performs her checks. For a general k, the analysis has shown that the investigated protocol is $(\gamma(k, \varphi), \delta_k(p_{\text{check}}))$-verifiable where $\delta_k(p_{\text{check}}) = (1 - p_{\text{check}})^{k+1}$. This means that in the second case, the verifiability of the Russian federal remote e-voting protocol is dependent on the probability that Voters perform their checks.

Outlook

Since this work has provided a security analysis w.r.t. verifiability, future work could analyze the Russian federal remote e-voting system focusing on other security properties of e-voting protocols such as accountability or vote secrecy. Basically, accountability characterizes e-voting systems that ensure the possibility to detect if the published election result does not match the election result reflecting how voters actually voted (as required for verifiability) and additionally guarantee that misbehaving protocol participants can even be identified as well as blamed for their misbehavior [35]. Additionally, future work could be concerned with code audits.

As stated in the introduction, two remote e-voting systems have been deployed in the parliamentary elections of 2021. The system used to conduct e-voting in Moscow is based on a completely different protocol [50]. Thus, it would require a separate security analysis to analyze this system w.r.t. verifiability.

Bibliography

[1] B. Adida. “Helios: Web-based Open-Audit Voting”. In: *17th USENIX Security Symposium (USENIX Security 08)*. San Jose, CA: USENIX Association, July 2008. URL: https://www.usenix.org/conference/17th-usenix-security-symposium/helios-web-based-open-audit-voting (cit. on p. 15).

[2] M. Bellare, C. Namprempre, D. Pointcheval, M. Semanko. *The One-More-RSA-Inversion Problems and the Security of Chaum’s Blind Signature Scheme*. Cryptology ePrint Archive, Paper 2001/002. 2001. URL: https://eprint.iacr.org/2001/002 (cit. on pp. 57, 58).

[3] M. Bellare, C. Namprempre, D. Pointcheval, M. Semanko. “The Power of RSA Inversion Oracles and the Security of Chaum’s RSA-Based Blind Signature Scheme”. In: *Financial Cryptography*. Ed. by P. Syverson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 319–338. ISBN: 978-3-540-46088-6 (cit. on p. 58).

[4] J. Benaloh. “Simple Verifiable Elections”. In: *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*. EVT’06. Vancouver, B.C., Canada: USENIX Association, 2006, p. 5 (cit. on p. 36).

[5] D. Bernhard, O. Pereira, B. Warinschi. “How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2012, pp. 626–643 (cit. on pp. 62–64).

[6] D. Bernhard, B. Warinschi. “Cryptographic Voting — A Gentle Introduction”. In: *Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures*. Ed. by A. Aldini, J. Lopez, F. Martinelli. Cham: Springer International Publishing, 2014, pp. 167–211. ISBN: 978-3-319-10082-1. DOI: 10.1007/978-3-319-10082-1_7. URL: https://doi.org/10.1007/978-3-319-10082-1_7 (cit. on p. 62).

[7] G. Bleumer. “Blinding Techniques”. In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg, S. Jajodia. Boston, MA: Springer US, 2011, pp. 150–152. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_182. URL: https://doi.org/10.1007/978-1-4419-5906-5_182 (cit. on p. 58).

[8] J. Bootle, A. Cerulli, P. Chaidos, J. Groth. “Efficient Zero-Knowledge Proof Systems”. In: *Foundations of Security Analysis and Design VIII: FOSAD 2014/2015/2016 Tutorial Lectures*. Ed. by A. Aldini, J. Lopez, F. Martinelli. Cham: Springer International Publishing, 2016, pp. 1–31. ISBN: 978-3-319-43005-8. DOI: 10.1007/978-3-319-43005-8_1. URL: https://doi.org/10.1007/978-3-319-43005-8_1 (cit. on pp. 27, 29).

[9] D. Chaum. “Blind Signatures for Untraceable Payments”. In: *Advances in Cryptology*. Ed. by D. Chaum, R. L. Rivest, A. T. Sherman. Springer US, 1983, pp. 199–203. ISBN: 978-1-4757-0602-4 (cit. on p. 25).

[10] D. Chaum, T. P. Pedersen. “Wallet Databases with Observers”. In: *Advances in Cryptology — CRYPTO’ 92*. Ed. by E. F. Brickell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105. ISBN: 978-3-540-48071-6 (cit. on p. 61).

[11] S. Coretti, U. Maurer, B. Tackmann. “Constructing Confidential Channels from Authenticated Channels—Public-Key Encryption Revisited”. In: *Advances in Cryptology - ASIACRYPT 2013*. Ed. by K. Sako, P. Sarkar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 134–153. ISBN: 978-3-642-42033-7 (cit. on p. 67).

[12] V. Cortier, D. Galindo, S. Glondu, M. Izabachene. *A generic construction for voting correctness at minimum cost - Application to Helios*. Cryptology ePrint Archive, Paper 2013/177. https://eprint.iacr.org/2013/177. 2013. URL: https://eprint.iacr.org/2013/177 (cit. on p. 62).

[13] V. Cortier, D. Galindo, S. Glondu, M. Izabachène. “Distributed ElGamal à La Pedersen: Application to Helios”. In: *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*. WPES ’13. Berlin, Germany: Association for Computing Machinery, 2013, 131–142. ISBN: 9781450324854. DOI: 10.1145/2517840.2517852. URL: https://doi.org/10.1145/2517840.2517852 (cit. on p. 50).

[14] V. Cortier, D. Galindo, R. Kuesters, J. Mueller, T. Truderung. *Verifiability Notions for E-Voting Protocols*. Cryptology ePrint Archive, Paper 2016/287. 2016. URL: https://eprint.iacr.org/2016/287 (cit. on pp. 15, 17, 19, 31, 33, 35, 36, 67).

[15] R. Cramer, R. Gennaro, B. Schoenmakers. “A secure and optimally efficient multi-authority election scheme”. In: *European Transactions on Telecommunications* 8.5 (1997), pp. 481–490. DOI: https://doi.org/10.1002/ett.4460080506. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.4460080506. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4460080506 (cit. on pp. 61, 63).

[16] Y. Desmedt, P. Chaidos. “Applying Divertibility to Blind Ballot Copying in the Helios Internet Voting System”. In: *Computer Security – ESORICS 2012*. Ed. by S. Foresti, M. Yung, F. Martinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 433–450. ISBN: 978-3-642-33167-1 (cit. on p. 65).

[17] W. Diffie, M. E. Hellman. “New Directions in Cryptography”. In: *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 1st ed. New York, NY, USA: Association for Computing Machinery, 2022, 365–390. ISBN: 9781450398275. URL: https://doi.org/10.1145/3549993.3550007 (cit. on p. 52).

[18] V. Dolmatov, A. Degtyarev. *GOST R 34.10-2012: Digital Signature Algorithm*. RFC 7091. Dec. 2013. DOI: 10.17487/RFC7091. URL: https://www.rfc-editor.org/info/rfc7091 (cit. on pp. 55–57).

[19] V. Dolmatov, A. Degtyarev. *GOST R 34.11-2012: Hash Function*. RFC 6986. Aug. 2013. DOI: 10.17487/RFC6986. URL: https://www.rfc-editor.org/info/rfc6986 (cit. on pp. 50, 51, 55).

[20] T. ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *Advances in Cryptology*. Ed. by G. R. Blakley, D. Chaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18. ISBN: 978-3-540-39568-3 (cit. on p. 52).

[21] W. Enterprise. *CFT consensus algorithm*. URL: https://docs.wavesenterprise.com/en/latest/description/consensus/CFT.html (visited on 11/22/2022) (cit. on p. 41).

[22] W. Enterprise. *PoA consensus algorithm*. URL: https://docs.wavesenterprise.com/en/latest/description/consensus/PoA.html#poa-consensus (visited on 11/22/2022) (cit. on p. 41).

[23] W. Enterprise. *Technical description of the Waves Enterprise platform*. URL: https://docs.wavesenterprise.com/en/latest/wedocs.pdf (visited on 11/22/2022) (cit. on p. 41).

[24] A. Fiat, A. Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by A. M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194. ISBN: 978-3-540-47721-1 (cit. on p. 62).

[25] M. Fischlin, D. Schröder. "Security of Blind Signatures under Aborts". In: *Public Key Cryptography – PKC 2009*. Ed. by S. Jarecki, G. Tsudik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 297–316. ISBN: 978-3-642-00468-1 (cit. on p. 25).

[26] J. P. Gibson, R. Krimmer, V. Teague, J. Pomares. "A review of e-voting: the past, present and future". In: *Annals of Telecommunications* 71.7 (2016), pp. 279–286 (cit. on p. 15).

[27] K. Gjøsteen. "The Norwegian internet voting protocol". In: *International Conference on E-Voting and Identity*. Springer. 2011, pp. 1–18 (cit. on p. 15).

[28] O. Goldreich. *Foundations of Cryptography: Volume 1*. USA: Cambridge University Press, 2006. ISBN: 0521035368 (cit. on p. 29).

[29] S Goldwasser, S Micali, C Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, 291–304. ISBN: 0897911512. DOI: 10.1145/22145.22178. URL: https://doi.org/10.1145/22145.22178 (cit. on p. 29).

[30] J. Katz, Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261 (cit. on pp. 20–22, 24, 27, 28, 58).

[31] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, B. Stiller. "Provotum: A Blockchain-based and End-to-end Verifiable Remote Electronic Voting System". In: *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. 2020, pp. 172–183. DOI: 10.1109/LCN48667.2020.9314815 (cit. on pp. 62–64).

[32] N. Koblitz. *A course in number theory and cryptography*. Vol. 114. Springer Science & Business Media, 1994 (cit. on pp. 20, 53).

[33] N. Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209 (cit. on pp. 20, 53).

[34] D. H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Feb. 1997. DOI: 10.17487/RFC2104. URL: https://www.rfc-editor.org/info/rfc2104 (cit. on p. 59).

[35] R. Kuesters, J. Liedtke, J. Mueller, D. Rausch, A. Vogt. *Ordinos: A Verifiable Tally-Hiding E-Voting System*. Cryptology ePrint Archive, Paper 2020/405. 2020. URL: https://eprint.iacr.org/2020/405 (cit. on pp. 15, 67, 69, 70, 74, 78).

[36] R. Küsters, T. Truderung, A. Vogt. “Accountability: Definition and Relationship to Verifiability”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS ’10. Chicago, Illinois, USA: Association for Computing Machinery, 2010, 526–535. ISBN: 9781450302456. DOI: 10.1145/1866307.1866366. URL: https://doi.org/10.1145/1866307.1866366 (cit. on p. 33).

[37] L. Li, A. A. Abd El-Latif, X. Niu. “Elliptic curve ElGamal based homomorphic image encryption scheme for sharing secret images”. In: *Signal Processing* 92.4 (2012), pp. 1069–1078. ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2011.10.020. URL: https://www.sciencedirect.com/science/article/pii/S0165168411003823 (cit. on pp. 20, 54).

[38] V. Mateu, J. M. Miret, F. Sebé. “A hybrid approach to vector-based homomorphic tallying remote voting”. In: *Int. J. Inf. Sec.* 15.2 (2016), pp. 211–221. DOI: 10.1007/s10207-015-0279-8. URL: https://doi.org/10.1007/s10207-015-0279-8 (cit. on p. 23).

[39] M. Nofer, P. Gomber, O. Hinz, D. Schiereck. “Blockchain”. In: *Business & Information Systems Engineering* 59 (Mar. 2017). DOI: 10.1007/s12599-017-0467-3 (cit. on p. 30).

[40] A. Parsovs. *Homomorphic Tallying for the Estonian Internet Voting System*. Cryptology ePrint Archive, Paper 2016/776. 2016. URL: https://eprint.iacr.org/2016/776 (cit. on p. 61).

[41] K. Peng, F. Bao. “An Efficient Range Proof Scheme”. In: *Proceedings of the 2010 IEEE Second International Conference on Social Computing*. SOCIALCOM ’10. USA: IEEE Computer Society, 2010, 826–833. ISBN: 9780769542119. DOI: 10.1109/SocialCom.2010.125. URL: https://doi.org/10.1109/SocialCom.2010.125 (cit. on p. 29).

[42] V. Popov, S. Leontiev, I. Kurepkin. *Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms*. RFC 4357. Jan. 2006. DOI: 10.17487/RFC4357. URL: https://www.rfc-editor.org/info/rfc4357 (cit. on p. 55).

[43] K. Rabah. “Elliptic Curve ElGamal Encryption and Signature Schemes”. In: *Information Technology Journal* (Mar. 2005). DOI: 10.3923/itj.2005.299.306 (cit. on p. 20).

[44] B. Schoenmakers. “A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting”. In: *Advances in Cryptology — CRYPTO’ 99*. Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 148–164. ISBN: 978-3-540-48405-9 (cit. on p. 61).

[45] A. Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (1979), 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176 (cit. on pp. 28, 59).

[46] S. V. Smyshlyaev, E. Alekseev, I. Oshkin, V. Popov, S. Leontiev, V. Podobaev, D. Belyavsky. *Guidelines on the Cryptographic Algorithms to Accompany the Usage of Standards GOST R 34.10-2012 and GOST R 34.11-2012*. RFC 7836. Mar. 2016. DOI: 10.17487/RFC7836. URL: https://www.rfc-editor.org/info/rfc7836 (cit. on p. 59).

[47] R. Sobti, G. Ganesan. “Cryptographic Hash Functions: A Review”. In: *International Journal of Computer Science Issues, ISSN (Online): 1694-0814* 9 (Mar. 2012), pp. 461 –479 (cit. on p. 20).

[48] M. Stadler, J.-M. Piveteau, J. Camenisch. "Fair Blind Signatures". In: *Advances in Cryptology — EUROCRYPT '95*. Ed. by L. C. Guillou, J.-J. Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 209–219. ISBN: 978-3-540-49264-1 (cit. on p. 25).

[49] O. Ugus, D. Westhoff, R. Laue, A. Shoufan, S. A. Huss. "Optimized Implementation of Elliptic Curve Based Additive Homomorphic Encryption for Wireless Sensor Networks". In: *CoRR* abs/0903.3900 (2009). arXiv: 0903.3900. URL: http://arxiv.org/abs/0903.3900 (cit. on pp. 52–54).

[50] J. Vakarjuk, N. Snetkov, J. Willemson. *Russian Federal Remote E-voting Scheme of 2021 – Protocol Description and Analysis*. Cryptology ePrint Archive, Paper 2021/1454. 2021. URL: https://eprint.iacr.org/2021/1454 (cit. on pp. 15, 17, 39, 43, 45–48, 67, 78).

[51] C. Vegas, J. Barrat. "Overview of current state of e-voting worldwide". In: *Real-World Electronic Voting*. Auerbach Publications, 2016, pp. 67–92 (cit. on p. 15).

[52] X. Yang, X. Yi, S. Nepal, A. Kelarev, F. Han. "A Secure Verifiable Ranked Choice Online Voting System Based on Homomorphic Encryption". In: *IEEE Access* 6 (2018), pp. 20506–20519. DOI: 10.1109/ACCESS.2018.2817518 (cit. on p. 61).

[53] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, M. Imran. "An overview on smart contracts: Challenges, advances and platforms". In: *Future Generation Computer Systems* 105 (2020), pp. 475–491 (cit. on p. 30).

[54] R. Zhou, Z. Lin. "An Improved Exponential ElGamal Encryption Scheme with Additive Homomorphism". In: *2022 International Conference on Blockchain Technology and Information Security (ICBCTIS)*. 2022, pp. 25–27. DOI: 10.1109/ICBCTIS55569.2022.00017 (cit. on p. 61).

All links were last followed on December 28, 2022.

Declaration