Blind accumulators for e-voting

Sergey Agievich
Research Institute for Applied Problems of Mathematics and Informatics
Belarusian State University
`agievich@{bsu.by,gmail.com}`

Abstract

We present a novel cryptographic primitive, blind accumulator, aimed at constructing e-voting systems. Blind accumulators collect private keys of eligible voters in a decentralized manner not getting information about the keys. Once the accumulation is complete, a voter processes the resulting accumulator deriving a public key that refers to the private key previously added by this voter. Public keys are derived deterministically and can therefore stand as fixed voter pseudonyms. The voter can prove that the derived key refers to some accumulated private key without revealing neither that key nor the voter itself. The voter uses the accumulated private key to sign a ballot. The corresponding public key is used to verify the signature. Since the public key is fixed, it is easy to achieve verifiability, to protect against multiple submissions of ballots by the same voter or, conversely, to allow multiple submissions but count only the last one. We suggest a syntax of blind accumulators and security requirements for them. We embed blind accumulators in the Pseudonymous Key Generation (PKG) protocol which details the use of accumulators in practical settings close to e-voting. We propose an implementation of the blind accumulator scheme whose main computations resemble the Diffie–Hellman protocol. We justify the security of the proposed implementation.

Keywords: e-voting, cryptographic accumulator, zero-knowledge proof, Diffie–Hellman protocol, square decisional Diffie–Hellman problem.

1 Preliminaries

In our country, state elections are held at polling stations, each serving from 20 to 3000 voters and is managed by a local electoral commission. Arriving at a station, a voter is authenticated by one of the commission members. At the same time as authentication, the affiliation of the voter to this station is checked. If successful, the voter receives a ballot signed by several commission members. There is a private (curtained) zone in which the voter fills out the ballot and folds it to hide the choice from persons at the polling station. The voter leaves the private zone, throws the ballot into a ballot box, and exits the polling station. At the end of the voting, the commission members open the box, unfold the ballots, process them, sum up the votes and publish the result.

The described voting scenario is typical, it is used in many other countries. Interestingly, the scenario follows fairly closely the process for papal selection known as the *conclave*, from the Latin *cum clavis* that literally means “with key” and implies “locked room”. The Latin basis of the term “conclave”, in our opinion, correctly indicates the key aspect of the scenario which is its hermeticity. All events are held in an isolated area under the control of the electoral commission. Only the private zone, which by analogy with “conclave” is naturally to call the *enclave*, falls out of control. Note that the boundary of the enclave is still controlled, the enclave remains part of the conclave.

The “conclave” functionality provides the following properties of the voting system (polling station).

1. Consistency: at any time during the voting, a voting system is in a correct state.

2. Eligibility: only eligible voters vote.

The “enclave” functionality provides one more property.

3. Privacy: individual votes remain secret.

The desirable features of voting systems do not end there. For example, the following property is also important.

4. Verifiability: voters should be able to verify if their votes are correctly accounted for.

The property can be supported by allowing voters to mark the ballots on their choice and requiring the electoral commission to publish scans of the processed ballots. A mark can be a special symbol in the voting field (our country’s electoral code allows any symbol). The symbol acts as a pseudonym that belongs to some eligible voter and only this voter knows to whom it belongs.

In this paper, the concept of pseudonym is important. Having a fixed pseudonym that is unambiguously associated with one of the voters of the polling station and at the same time unknown with whom specifically, it is possible to achieve not only verifiability but also other properties. For example, we may protect against multiple submissions of ballots by the same voter (with the same pseudonym) or, conversely, may allow multiple submissions but count only the last one.

Above, we started talking about electronic voting or *e-voting*. There are obvious analogies between the elements of conventional voting at polling stations and cryptographic schemes and protocols used in e-voting. So, for example, the enclave is associated with the voter’s private key and signing a ballot with this key. Or the folding and unfolding of a ballot are, obviously, encryption with a public key of the electoral commission and subsequent decryption with the corresponding private key (it can be preliminarily split into parts distributed among commission members). Moreover, many cryptographic primitives have emerged in response to challenges encountered when developing e-voting systems.

The transition to e-voting is a part of the general trend to improve the convenience and availability of public services. But that is not all. There have been and remain expectations that e-voting increases the transparency of voting and confidence in its results. In the voting scenario described at the beginning, this confidence is based solely on trust in electoral commissions which in many cases is seriously undermined. The question arises as to whether it is possible to organize voting so that the following additional property holds.

5. Decentralization: there is no electoral commission, voters jointly control the voting process.

Next, we propose a novel cryptographic primitive called *blind accumulator* that helps construct e-voting systems satisfying properties 1 –5. A blind accumulator acts as a digital conclave that collects private keys from digital enclaves of voters doing this in a decentralized manner and not getting information about the keys.

Once the accumulation is complete, a voter processes the resulting accumulator deriving a public key that refers to the private key previously added by this voter. Public keys are derived deterministically and can therefore stand as fixed voter pseudonyms. The voter can prove that the derived key refers to some accumulated private key without revealing neither that key nor the voter itself. The voter uses the accumulated private key to sign a ballot. The corresponding public key is used to verify the signature. Since the public key is hard to associate with the voter, the ballot does not even need to be encrypted to preserve privacy. However, in some cases (for example, when intermediate voting results cannot be announced during the voting process), encryption should be provided. This is beyond the scope of the paper.

The syntax of blind accumulators and security requirements for them are stated in Section 2. In Section 3, blind accumulators are embedded in the Pseudonymous Key Generation (PKG) protocol. The protocol details the use of blind accumulators in practical settings close to e-voting. The PKG protocol performs pseudonymization of public keys: an input public key associated with a *particular* party of the protocol is turned into a public key associated with *some* party. In Section 4, we propose an implementation of blind accumulators. The security of this implementation is discussed in Section 5.

With n voters, the proposed implementation requires storing $O(n^2)$ elements of a cyclic group of large prime order q and $O(n)$ scalars modulo q as final and intermediate accumulators and associated proofs. Validating the correctness of all proofs requires $O(n^2)$ scalar multiplications in the cyclic group. The time and memory requirements are not burdensome with the $n \leq 3000$ threshold mentioned at the beginning. However, if the threshold is exceeded, other implementations should be considered. One of the promising directions here is the division of voters into small random groups that separately run PKG. Once the grouping-then-PKG round is complete, voters use derived pseudonymous public keys in the second round, and then in several more rounds achieving full pseudonymization.

2 Blind accumulators

2.1 Concept

Cryptographic accumulators introduced in [2, 5, 6] are special encodings of tuples of objects. We write $\mathbf{a} = [S]$ to denote that an accumulator $\mathbf{a}$ encodes a tuple S. We assume that the encoding $S \mapsto [S]$ is driven by public parameters and that it is deterministic given the parameters. We interpret tuples as ordered multisets bringing standard set notations such as the curly braces, the membership ($\in$) and union ($\cup$) symbols. Accumulators are managed by algorithms that translate operations involving S into operations over $[S]$. We support only two operations: adding an object and membership verification. We avoid the usual requirement that the encoding $[S]$ has to be succinct.

Typically, an accumulator $[S]$ as well as the underlying set S are public. In our case, this is not true: $[S]$ remains public but S consists of private keys known only to their owners. Informally speaking, the accumulator collects objects *blindly.* That is why we call such accumulators *blind.*

A private key $sk \in S$ added to the accumulator $[S]$ relates to a public key pk which is derived from $[S]$ with sk. The derived key is accompanied by a proof that $sk \in S$. The important point here is that the proof does not reveal sk.

Another important point is that blind accumulators are not managed by any trusted party which is usually responsible for maintaining the consistency of accumulators during their updates. Without a trusted party, the consistency is maintained in a decentralized manner by validating transitions between $[S]$ and $[S \cup \{sk\}]$. Each transition is accompanied by a proof of consistency generated by a party who adds sk to S.

2.2 Syntax

A *blind accumulator scheme* is a tuple of polynomial-time algorithms $\mathsf{BAcc} = (\mathsf{Init}, \mathsf{Add}, \mathsf{PrvAdd}, \mathsf{VfyAdd}, \mathsf{Der}, \mathsf{PrvDer}, \mathsf{VfyDer})$ that are defined as follows.

1. The probabilistic algorithm $\mathsf{Init}\colon 1^l \mapsto \mathbf{a}_0$ takes a security level $l \in \mathbb{N}$ (in the unary form) and outputs an initial accumulator $\mathbf{a}_0 = [\varnothing]$.

   (a) We assume that $\mathbf{a}_0$ implicitly refers to l and public parameters (such as a description of an elliptic curve) and that these parameters implicitly define a set of private keys SKeys and a set of public keys PKeys.

2. The deterministic algorithm $\mathsf{Add}\colon (\mathbf{a}, sk) \mapsto \mathbf{a}'$ takes an accumulator $\mathbf{a} = [S]$ and a private key sk, and outputs an updated accumulator $\mathbf{a}' = [S \cup \{sk\}]$.

   (a) We assume that every accumulator $\mathbf{a}$ that is input to Add is an output of either Init or some previous call to Add. This ensures the *consistency* of $\mathbf{a}$, i.e. that it is constructed as

   $$\mathbf{a} \leftarrow \mathsf{Add}(\ldots(\mathsf{Add}(\mathsf{Add}(\mathbf{a}_0, sk_1), sk_2), \ldots), sk_n), \quad \mathbf{a}_0 \leftarrow \mathsf{Init}(1^l), \quad sk_i \in \mathsf{SKeys},$$

   and therefore is an incrementally built encoding $[S]$ of the multiset $S = \{sk_1, sk_2, \ldots, sk_n\}$.

   (b) We assume that public parameters referenced in the initial accumulator $\mathbf{a}_0$ are passed to all accumulators incrementally built from it.

   (c) For simplicity and without loss of generality, we suppose that all accumulators below relate to the same initial $\mathbf{a}_0$ and therefore belong to the same security level l and use the same public parameters.

3. The probabilistic algorithm $\mathsf{PrvAdd}\colon (\mathbf{a}, \mathbf{a}', sk) \mapsto \alpha$ takes accumulators $\mathbf{a}$, $\mathbf{a}'$ and a private key sk, and generates a proof α that $\mathbf{a}' = \mathsf{Add}(\mathbf{a}, sk)$.

4. The deterministic algorithm $\mathsf{VfyAdd}\colon (\mathbf{a}, \mathbf{a}', \alpha) \mapsto b$ takes accumulators $\mathbf{a}$, $\mathbf{a}'$ and a proof α that $\mathbf{a}' = \mathsf{Add}(\mathbf{a}, sk)$ for some private key sk. The algorithm verifies the proof and outputs either $b = 1$ for acceptance or $b = 0$ for rejection.

   (a) We require that if $\mathbf{a}' \leftarrow \mathsf{Add}(\mathbf{a}, sk)$ and $\alpha \leftarrow \mathsf{PrvAdd}(\mathbf{a}, \mathbf{a}', sk)$, then $\mathsf{VfyAdd}(\mathbf{a}, \mathbf{a}', \alpha) = 1$.

5. The deterministic algorithm $\mathsf{Der}\colon (\mathbf{a}, sk) \mapsto pk \mid\perp$ takes an accumulator $\mathbf{a}$ and a private key sk, and either derives a public key pk or outputs the error symbol $\perp$.

   (a) We require that for a consistent $\mathbf{a} = [S]$, $\mathsf{Der}(\mathbf{a}, sk) = \perp$ if and only if $sk \notin S$.

   (b) We require that if a private key sk is chosen uniformly at random from SKeys, $sk \in S$ and $\mathbf{a} = [S]$, then $pk \leftarrow \mathsf{Der}(\mathbf{a}, sk)$ has a fixed distribution D over PKeys regardless of the structure of S.

6. The probabilistic algorithm $\mathsf{PrvDer}\colon (\mathbf{a}, pk, sk) \mapsto \delta$ takes an accumulator $\mathbf{a}$, a private key sk and a public key pk, and generates a proof δ that $pk = \mathsf{Der}(\mathbf{a}, sk)$.

7. The deterministic algorithm $\mathsf{VfyDer}\colon (\mathbf{a}, pk, \delta) \mapsto b$ takes an accumulator $\mathbf{a}$, a public key pk and a proof δ that $pk = \mathsf{Der}(\mathbf{a}, sk)$ for some private key sk. The algorithm verifies the proof and outputs either $b = 1$ for acceptance or $b = 0$ for rejection.

   (a) We require that if $pk \leftarrow \mathsf{Der}(\mathbf{a}, sk)$ and $\delta \leftarrow \mathsf{PrvDer}(\mathbf{a}, pk, sk)$, then $\mathsf{VfyDer}(\mathbf{a}, pk, \delta) = 1$.

2.3 Consistency

Since the input sk of Add is secret and not revealed, a dishonest party involved in accumulator management can submit a counterfeit $\mathbf{a}'$ as the output of Add and thereby violate the consistency of accumulators. That is why we strengthen Add with PrvAdd and VfyAdd. We impose the following security requirement on the algorithms.

Definition 1. A scheme BAcc provides *consistency* if (i) implies (ii), where

(i) there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ that takes a pair of accumulators $(\mathbf{a}, \mathbf{a}')$ of security level l and outputs a proof α such that $\mathbf{P}\{\mathsf{VfyAdd}(\mathbf{a}, \mathbf{a}', \alpha) = 1\}$ is non-negligible in l;

(ii) there exists a probabilistic polynomial-time algorithm $\mathcal{E}$ that takes a pair of accumulators $(\mathbf{a}, \mathbf{a}')$ of security level l, uses $\mathcal{A}$ as an oracle and outputs a private key sk such that $\mathbf{P}\{\mathsf{Add}(\mathbf{a}, sk) = \mathbf{a}'\}$ is non-negligible in l. The algorithm $\mathcal{E}$ is allowed to manage the runtime environment of $\mathcal{A}$ without having access to its internals.

The consistency means that an algorithm $\mathcal{A}$ that claims to generate a correct proof α not using a private key sk actually almost certainly uses it, as $\mathcal{E}$ shows. So, a transition from $\mathbf{a}$ to $\mathbf{a}'$ that is confirmed by VfyAdd is almost certainly driven by a valid private key and $\mathbf{a}'$ is consistent provided that $\mathbf{a}$ is consistent.

Our notion of consistency relates to the (special) soundness in zero-knowledge proofs (ZKP, starting from [11]). There $\mathcal{E}$ is called a *knowledge extractor* [3].

Managing the runtime environment of a hypothetical adversary is commonplace in ZKP. The environment becomes convenient for $\mathcal{E}$ and sometimes even idealized but remains realistic. The algorithm $\mathcal{E}$ is usually allowed to replace hash functions with random oracles (see Section 5), to program these oracles, to feed $\mathcal{A}$ with random tapes, to rewind $\mathcal{A}$, that is, to run $\mathcal{A}$ several times repeating a random tape.

2.4 Soundness

To protect against an adversary who claims that a counterfeit pk is derived from an accumulator $\mathbf{a}$ using Der and therefore refers to some private key sk previously added to $\mathbf{a}$, such a claim has to be accompanied by a proof δ generated using PrvDer and verified using VfyDer. We impose the following security requirement that literally corresponds to the (special) soundness in ZKP.

Definition 2. A scheme BAcc provides *soundness* if (i) implies (ii), where

(i) there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ that takes a consistent accumulator $\mathbf{a}$ of security level l and a public key pk, and outputs a proof δ such that $\mathbf{P}\{\mathsf{VfyDer}(\mathbf{a}, pk, \delta) = 1\}$ is non-negligible in l;

(ii) there exists a probabilistic polynomial-time algorithm $\mathcal{E}$ that takes a consistent accumulator $\mathbf{a}$ of security level l and a public key pk, uses $\mathcal{A}$ as an oracle, and outputs a private key sk such that $\mathbf{P}\{\mathsf{Der}(\mathbf{a}, sk) = pk\}$ is non-negligible in l. The algorithm $\mathcal{E}$ is allowed to manage the runtime environment of $\mathcal{A}$ without having access to its internals.

The soundness means that if an algorithm $\mathcal{A}$ is able to generate a correct proof δ that a derived public key pk refers to some private key sk from an accumulator, then this algorithm almost certainly uses this sk and, therefore, is run by an eligible party who previously added sk to the accumulator.

2.5 Blindness

To protect against an adversary who extracts from proofs α and δ generated by PrvAdd and PrvDer an information about sk, we impose the following security requirement.

Definition 3. A scheme BAcc provides *blindness* if

(i) there exists a probabilistic polynomial-time algorithm $\mathcal{S}_1$ that takes consistent accumulators $\mathbf{a}$ and $\mathbf{a}' = \mathsf{Add}(\mathbf{a}, sk)$, and generates a proof α' that is statistically indistinguishable from $\alpha = \mathsf{PrvAdd}(\mathbf{a}, \mathbf{a}', sk)$ and is accepted by VfyAdd;

(ii) there exists a probabilistic polynomial-time algorithm $\mathcal{S}_2$ that takes a consistent accumulator $\mathbf{a}$ and a public key $pk = \mathsf{Der}(\mathbf{a}, sk)$, and generates a proof δ' that is statistically indistinguishable from $\delta = \mathsf{PrvDer}(\mathbf{a}, pk, sk)$ and is accepted by VfyDer.

The algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ are allowed to manage the runtime environment of BAcc without having access to private keys processed using BAcc.

The idea behind the blindness is that if there exists a way to generate proofs not distinguishable from correct ones not using a private key, then these proofs indeed contain no information about the key. The trick here is the control over the runtime environment. Such control is commonplace in ZKP where a similar requirement is called HVZK (honest verifier zero-knowledge).

2.6 Unlinkability

Consider the game $\mathsf{G}(1^l, n, m)$ between probabilistic algorithms $\mathcal{V}$ and $\mathcal{A}$. These algorithms represent honest and dishonest parties involved in accumulator management. The inputs of the game are a security level l, a total number of parties n and a number of honest parties m. We require that $1 \leq m \leq n$.

The rules of the game are defined below. Hereinafter we write $r_1, r_2, \ldots \overset{L}{\leftarrow} R$ to denote that $r_1, r_2, \ldots$ are chosen independently at random from a set R according to a probability distribution L and denote by \$ the uniform distribution.

1. $\mathcal{V}$ takes 1^l, computes and publishes $\mathbf{a} \leftarrow \mathsf{Init}(1^l)$.

2. $\mathcal{A}$ and $\mathcal{V}$ make n moves of the form $\mathbf{a} \leftarrow \mathsf{Add}(\mathbf{a}, sk)$, where sk is a private key chosen by a player who moves and known only to this player. $\mathcal{A}$ makes $n - m$ moves for dishonest parties, $\mathcal{V}$ makes m moves for honest parties, the sequence of moves is determined by $\mathcal{A}$. The players accompany their moves with proofs constructed using PrvAdd and accepted by VfyAdd.

3. In their moves, $\mathcal{A}$ uses arbitrary private keys, $\mathcal{V}$ uses private keys $sk_1, sk_2, \ldots, sk_m \overset{\$}{\leftarrow} \mathsf{SKeys}$.

4. $\mathcal{V}$ is allowed to manage the runtime environment of BAcc and $\mathcal{A}$.

5. After completing n moves, $\mathcal{A}$ and $\mathcal{V}$ obtain an accumulator $\mathbf{a} = [S]$ such that $sk_1, sk_2, \ldots, sk_m \in S$.

6. $\mathcal{V}$ computes the tuple $\mathbf{pk} \leftarrow (pk_1, pk_2, \ldots, pk_m)$ in which $pk_i \leftarrow \mathsf{Der}(\mathbf{a}, sk_i)$.

7. $\mathcal{V}$ generates $b \overset{\$}{\leftarrow} \{0, 1\}$. If $b = 0$, then $\mathcal{V}$ additionally generates $j \overset{\$}{\leftarrow} \{1, 2, \ldots, m\}$, $\rho \overset{D}{\leftarrow} \mathsf{PKeys}$ and replaces in $\mathbf{pk}$ the key pk_j by ρ.

8. $\mathcal{V}$ passes $\mathbf{pk}$ (either original or corrected) to $\mathcal{A}$.

9. $\mathcal{A}$ outputs a guess $\hat{b} \in \{0, 1\}$ of the bit b.

In the game, $\mathcal{A}$ demonstrates and $\mathcal{V}$ validates the capabilities to distinguish a correct public key of an honest party from a random key. The negligibility of distinguishing capabilities, which is required in the following definition, means the hardness of associating public keys with their owners or, in short, unlinkability.

Definition 4. A scheme BAcc provides *unlinkability* if for any probabilistic polynomial-time algorithm $\mathcal{A}$ that plays the game $\mathsf{G}(1^l, n, m)$ and outputs a guess $\hat{b}$ of the bit b, it holds that

$$\mathbf{Adv}(\mathcal{A}) = \left| \mathsf{P}\left\{\hat{b} = 1 \mid b = 1\right\} - \mathsf{P}\left\{\hat{b} = 1 \mid b = 0\right\} \right|$$

is negligible in l uniformly in n.

This coalition of dishonest parties presented by $\mathcal{A}$ has a large attack potential being allowed to update accumulators with arbitrary private keys at arbitrary times. The potential of the verifier $\mathcal{V}$ which is allowed to manage the runtime environment is also large. Indeed, if BAcc satisfies the consistency and blindness requirements, then

- $\mathcal{V}$ is able to determine private keys added by $\mathcal{A}$;
- $\mathcal{V}$ is able to generate valid proofs of consistency even not knowing private keys sk_i.

We use these observations in Section 5 when justifying our implementation of BAcc.

3 Pseudonymous key generation

The BAcc scheme can be used for *pseudonymous key generation*, PKG for short. PKG directly relates to e-voting supporting 5 declared properties: consistency, eligibility, privacy, verifiability, decentralization.

PKG is a protocol in which n authorized parties (voters) $P_1, P_2, \ldots, P_n$ and moderator participate. The parties confirm their authenticity by signing messages with private keys. The corresponding public keys are registered in a trusted infrastructure. A signature of a message μ generated by a party P_i is denoted by $\mathsf{Sig}_{P_i}(\mu)$. The signature acts as a message dependent proof of knowledge of a private key. We assume that the signature $\mathsf{Sig}_{P_i}(\mu)$ along with authorization permissions of P_i can be verified by any other party P_j and the moderator. Let verification be performed through a publicly available trusted service VfySig that, for example, aggregates conventional public key infrastructure services.

The moderator is responsible for initializing the protocol, for storing accumulators that are updated by the parties during the protocol execution, for providing access to the accumulators, for verifying proofs of consistency of the accumulators. These functions are partially duplicated by the parties themselves, who independently verify the consistency. A virtual moderation through consensus decisions of the parties is potentially possible.

The PKG protocol runs as follows:

1. The moderator computes $\mathbf{a}_0 \leftarrow \mathsf{Init}(1^l)$, sets $(\mathbf{a}, \pi) \leftarrow (\mathbf{a}_0, \varnothing)$ and publishes the pair $(\mathbf{a}, \pi)$. The second element of the pair (initially empty) is the list of proofs and related data.
2. A party $P \in \{P_1, P_2, \ldots, P_n\}$:
   (a) gets access to $(\mathbf{a}, \pi)$;
   (b) verifies proofs in π using the algorithm VfyAdd and the service VfySig;
   (c) generates a private key $sk \xleftarrow{\$} \mathsf{SKeys}$ and saves it;
   (d) computes $\mathbf{a}' \leftarrow \mathsf{Add}(\mathbf{a}, sk)$, $\alpha \leftarrow \mathsf{PrvAdd}(\mathbf{a}, \mathbf{a}', sk)$, $\sigma \leftarrow \mathsf{Sig}_P(\mathbf{a}, \mathbf{a}', \alpha)$;
   (e) sends the request to the moderator to replace $\mathbf{a}$ with $\mathbf{a}'$ and append $(\mathbf{a}, \alpha, \sigma)$ to π.
3. The moderator verifies the request using VfyAdd and VfySig. The moderator additionally checks that P's signature is not present in π and, therefore, P's requests were either not sent or not accepted. If the checks are successful, then $\mathbf{a}$ is replaced with $\mathbf{a}'$ and π is appended with $(\mathbf{a}, \alpha, \sigma)$. The updated pair $(\mathbf{a}, \pi)$ is published, proofs in π can be verified by any party at any time.
4. The steps 2 and 3 are interpreted as registering P. Parties are registered in no particular order. When processing a registration request from one party, the moderator suspends requests from other parties. The registration closes at a pre-announced time. After that the pair $(\mathbf{a}, \pi)$ no longer changes. The list π confirms the consistency of $\mathbf{a}$ and the fact that only authorized (eligible) parties have been registered.

5. A registered party $P \in \{P_1, P_2, \ldots, P_n\}$ reads $\mathbf{a}$, derives the key $pk \leftarrow \mathsf{Der}(\mathbf{a}, sk)$ and computes the proof $\delta \leftarrow \mathsf{PrvDer}(\mathbf{a}, pk, sk)$.

   As a final result, P obtains the keys (sk, pk) and the proof δ that pk is correctly derived from $\mathbf{a}$. The proof can be verified using the algorithm VfyDer. The correctness of the proof confirms the fact of registration of the party. The key pk acts as party's fixed pseudonym. The unlinkability property of blind accumulators makes it difficult to match parties and pseudonyms.

The party P uses the resulting triple (sk, pk, δ) in cryptographic systems outside of PKG. Each time the pseudonym pk is used, the party has to prove knowledge of sk or, in other words, ownership of the pseudonym.

To prove knowledge of a private key, *BAcc-friendly* systems should be used. These systems are compatible with the relationship between sk and pk established in PKG by the BAcc algorithms. In the implementation of BAcc described in the next section, this relationship is standard for cryptography in cyclic groups and, therefore, the well-known ElGamal [10] and Schnorr [13] signatures are BAcc-friendly.

Constructing BAcc-friendly cryptographic systems is beyond the scope of this paper. We only note that, apparently, the always working way to construct BAcc-friendly digital signatures is to extend the interfaces of PrvDer and VfyDer with an additional input through which a message to be signed or a message whose signature to be verified is passed.

If a BAcc-friendly digital signature is constructed, then P signs the data with sk and accompanies the signature with the pair (pk, δ). For example, a voter signs a ballot. The correctness of the signature as well as the proof δ relative to $(\mathbf{a}, pk)$ means that the ballot is signed by one of the eligible voters that took part in creating the accumulator $\mathbf{a}$ although it is not known which exactly voter signed. The proofs in π accompanying $\mathbf{a}$ ensure the consistency of the accumulator and the e-voting in general and non-volatility of pk supports verifiability. With all this, the moderation, the only element of centralization in PKG, reduces to providing access to the pair $(\mathbf{a}, \pi)$.

4 Implementation

We propose an implementation of the BAcc scheme whose main computations resemble the Diffie–Hellman protocol and which is therefore called BAcc-DH.

In BAcc-DH, we use a cyclic group $\mathbb{G}_q$ of large prime order q. We write the group additively and denote by $\mathbb{G}_q^*$ the set of nonzero elements of $\mathbb{G}_q$. We also use the ring $\mathbb{Z}_q$ of residues of integers modulo q and the set $\mathbb{Z}_q^*$ of nonzero (invertible) residues.

The group $\mathbb{G}_q$ is constructed in the algorithm BAcc-DH.Init. An input security level l determines the bit length of q. Once $\mathbb{G}_q$ is constructed, the set of private keys SKeys and the set of public keys PKeys are defined as $\mathbb{Z}_q^*$ and $\mathbb{G}_q^*$ respectively.

The initial accumulator $\mathbf{a}_0$ and all subsequent accumulators are words in the alphabet $\mathbb{G}_q^*$. The set of non-empty words in an alphabet Σ is denoted by Σ^+. The notation $(\mathbb{G}_q^*)^+$ is shortened to $\mathbb{G}_q^{*+}$. For a word $\mathbf{w}$, let $|w|$ be its length, $\mathsf{first}(\mathbf{w})$ be the first symbol of $\mathbf{w}$, $\mathsf{last}(\mathbf{w})$ be the last symbol, and $\mathsf{most}(\mathbf{w})$ be the word $\mathbf{w}$ after dropping its last symbol.

Algorithm BAcc-DH.Init

Input: 1^l (security level).

Output: $\mathbf{a}_0 \in \mathbb{G}_q^{*+}$ (initial accumulator).

Steps:

```
1. Construct a group $\mathbb{G}_q$ of prime order $q$ such that $C_1 2^l < q < C_2 2^l$, where $C_1$, $C_2$ are some constants.
2. Choose $G \in \mathbb{G}_q^*$.
3. $\mathbf{a}_0 \leftarrow G$.
4. Return $\mathbf{a}_0$.
```

Since q is prime, the element G chosen at step 2 is a generator of $\mathbb{G}_q$. The mapping $\mathbb{Z}_q \to \mathbb{G}_q$, $u \mapsto V = uG$ is an (addition-preserving) homomorphism known as scalar multiplication. We suppose that images of the homomorphism can be computed in time polynomial in l. We also assume that the inversion of the homomorphism, that is, the discrete logarithm $V \mapsto u = \log_G V$ is hard. More precisely, we suppose that computing logarithms takes time $\Omega(2^{l/2})$ on average. In fact, we impose the strongest security requirements on $\mathbb{G}_q$ since discrete logarithm methods are known that run in time $O(\sqrt{q})$ for any group of order q. We call $\mathbb{G}_q$ *cryptographically strong.*

Algorithm BAcc-DH.Add

Input: $\mathbf{a} \in \mathbb{G}_q^{*+}$ (accumulator), $u \in \mathbb{Z}_q^*$ (private key).
Output: $\mathbf{a}' \in \mathbb{G}_q^{*+}$ (updated accumulator).
Steps:

```
1. Parse $\mathbf{a} = G_0 G_1 \ldots G_n$.
2. $\mathbf{a}' \leftarrow G_0' G_1' \ldots G_n' G_0$, where $G_i' = uG_i$.
3. Return $\mathbf{a}'$.
```

Further we add words and multiply them by scalars in a component-wise manner. For example, the accumulator $\mathbf{a}'$ constructed above satisfies the equation $\mathsf{most}(\mathbf{a}') = u\,\mathbf{a}$. In the following algorithms, a proof of the validity of this equation is constructed and verified. To construct the proof, we use the fact that the mapping $u \mapsto u\,\mathbf{a}$ (where zero u is allowed) is a hard invertible homomorphism and therefore a well developed technique from [12] can be applied. The proof possesses the special soundness and HVZK properties that provide the consistency and blindness of BAcc-DH.

When constructing and verifying the proof, a hash function H is used. The function processes arbitrary input data (assuming they are pre-encoded into a binary word) and outputs a residue $h \in \mathbb{Z}_q$.

Algorithm BAcc-DH.PrvAdd

Input: $\mathbf{a}, \mathbf{a}' \in \mathbb{G}_q^{*+}$ (accumulators), $u \in \mathbb{Z}_q^*$ (private key).
Output: $\alpha \in \mathbb{G}_q^{+} \times \mathbb{Z}_q$ (proof).
Steps:

```
1. If $|\mathbf{a}'| \neq |\mathbf{a}| + 1$ or $\mathsf{last}(\mathbf{a}') \neq \mathsf{first}(\mathbf{a})$, return $(G, 0)$ (dummy proof).
2. $k \xleftarrow{\$} \mathbb{Z}_q$.
3. $\mathbf{r} \leftarrow k\,\mathbf{a}$.
4. $h \leftarrow H(\mathbf{a}, \mathbf{a}', \mathbf{r})$.
5. $s \leftarrow (k - hu) \bmod q$.
6. $\alpha \leftarrow (\mathbf{r}, s)$.
7. Return $\alpha$.
```

Algorithm BAcc-DH.VfyAdd

Input: $\mathbf{a}, \mathbf{a}' \in \mathbb{G}_q^{*+}$ (accumulators), $\alpha \in \mathbb{G}_q^{+} \times \mathbb{Z}_q$ (proof).
Output: 1 (accept) or 0 (reject).
Steps:

```
1. Parse $\alpha = (\mathbf{r}, s)$.
2. If $|\mathbf{r}| \neq |\mathbf{a}|$ or $|\mathbf{a}'| \neq |\mathbf{a}| + 1$ or $\mathsf{last}(\mathbf{a}') \neq \mathsf{first}(\mathbf{a})$, return 0.
3. $h \leftarrow H(\mathbf{a}, \mathbf{a}', \mathbf{r})$.
4. If $\mathbf{r} \neq s\,\mathbf{a} + h\,\mathsf{most}(\mathbf{a}')$, return 0.
5. Return 1.
```

Adding n keys and constructing the corresponding proofs require $n(n+1)$ scalar multiplications in $\mathbb{G}_q$. Interestingly, the amount of computation increases with each new key added: $2i$ multiplications for the ith key. In the terms of the PKG protocol, each new registration is computationally harder.

After adding the private keys $u_1, u_2, \ldots, u_n$, the resulting accumulator is the word $G_0 G_1 \ldots G_n$ in which

$$G_0 = UG, \quad G_i = \frac{U}{u_i} G, \quad i = 1, 2, \ldots, n.$$

Here $U = \prod_{i=1}^{n} u_i$. Note that the size of the accumulator grows linearly with n, the accumulator is not succinct.

The algorithm Der assigns to a private key u_i a public key $V = u_i G_0$. By construction,

$$u_i = \log_{G_i} G_0 = \log_{G_0} V.$$

An owner of u_i can prove the last equation by representing it as the knowledge of two equal discrete logarithms, employing the homomorphism $G_i G_0 \mapsto u_i(G_0 V)$ and using the mentioned technique from [12]. To hide i, the proof is concealed in the OR-composition

$$\bigvee_{j=1}^{n} \left[u_i = \log_{G_0} V = \log_{G_j} G_0 \right].$$

Such a composition is a well-known ZKP tool introduced in [8]. We use this tool in the algorithms BAcc-DH.PrvDer and BAcc-DH.VfyDer.

Algorithm BAcc-DH.Der

Input: $\mathbf{a} \in \mathbb{G}_q^{*+}$ (accumulator), $u \in \mathbb{Z}_q^*$ (private key).
Output: $V \in \mathbb{G}_q^*$ (public key).
Steps:

```
1. Parse $\mathbf{a} = G_0 G_1 \ldots G_n$.
2. Find $i \in \{1, 2, \ldots, n\}$ such that $uG_i = G_0$. If such $i$ does not exist, return $\perp$.
3. Return $uG_0$.
```

Algorithm BAcc-DH.PrvDer

Input: $\mathbf{a} \in \mathbb{G}_q^{*+}$ (accumulator), $u \in \mathbb{Z}_q^*$ (private key), $V \in \mathbb{G}_q^*$ (public key).

```
Output: $\delta \in \mathbb{Z}_q^+ \times \mathbb{Z}_q^+$ (proof).
Steps:
  1. Parse $\mathbf{a} = G_0G_1 \ldots G_n$.
  2. Find $i \in \{1, 2, \ldots, n\}$ such that $uG_i = G_0$. If such $i$ does not exist, return $(0, 0)$.
  3. For $j = 1, 2, \ldots, n$, $j \neq i$:
     (a) $h_j, s_j \xleftarrow{\$} \mathbb{Z}_q$;
     (b) $\mathbf{r}_j \leftarrow s_j(G_jG_0) + h_j(G_0V)$.
  4. $k_i \xleftarrow{\$} \mathbb{Z}_q$.
  5. $\mathbf{r}_i \leftarrow k_i(G_iG_0)$.
  6. $h_i \leftarrow \left(H(\mathbf{a}, \mathbf{r}_1\mathbf{r}_2 \ldots \mathbf{r}_n, V) - \sum_{j \neq i} h_j\right) \bmod q$.
  7. $s_i \leftarrow (k_i - uh_i) \bmod q$.
  8. $\delta \leftarrow (h_1h_2 \ldots h_n, s_1s_2 \ldots s_n)$.
  9. Return $\delta$.
```

Algorithm BAcc-DH.VfyDer

```
Input: $\mathbf{a} \in \mathbb{G}_q^{*+}$ (accumulator), $V \in \mathbb{G}_q^*$ (public key), $\delta \in \mathbb{Z}_q^+ \times \mathbb{Z}_q^+$ (proof).
Steps:
  1. Parse $\delta = (\mathbf{h}, \mathbf{s})$. If $|\mathbf{h}| \neq |\mathbf{s}|$ or $|\mathbf{a}| \neq |\mathbf{h}| + 1$, return 0.
  2. Parse $\mathbf{a} = G_0G_1 \ldots G_n$, $\mathbf{h} = h_1h_2 \ldots h_n$ and $\mathbf{s} = s_1s_2 \ldots s_n$.
  3. For $j = 1, 2, \ldots, n$:
     (a) $\mathbf{r}_j \leftarrow s_j(G_jG_0) + h_j(G_0V)$.
  4. If $H(\mathbf{a}, \mathbf{r}_1\mathbf{r}_2 \ldots \mathbf{r}_n, V) \not\equiv h_1 + h_2 + \ldots + h_n \pmod{q}$, return 0.
  5. Return 1.
```

It is easy to check that BAcc-DH meets the requirements for the BAcc syntax (see § 2.2). In particular, the public key $V = uG_0$ that corresponds to a random private key $u \xleftarrow{\$} \mathbb{Z}_q^*$ is uniformly distributed over $\mathbb{G}_q^*$ independently of other accumulated private keys.

A private key u added to the accumulator $\mathbf{a} = G_0G_1 \ldots G_n$ and the corresponding public key $V = uG_0$ can be used in the ElGamal and Schnorr signatures. The Schnorr signature algorithms are similar to the algorithms BAcc-DH.PrvAdd and BAcc-DH.VfyAdd exploiting the same scheme. The signature of a message μ is a pair $(h, s) \in \mathbb{Z}_q \times \mathbb{Z}_q$ that is generated as follows:

$$k \xleftarrow{\$} \mathbb{Z}_q, \quad r \leftarrow kG_0, \quad h \leftarrow H(r, \mu), \quad s \leftarrow (k - hu) \bmod q.$$

The verification equation: $H(sG_0 + hV, \mu) = h$.

5 Security

In this section, we justify the security of BAcc-DH. We examine 4 security requirements stated in Section 2 each time switching to the context of the corresponding security definition.

The security definitions allow runtime environments to be managed. We use this to replace the hash function H with a random oracle (see [4]) and permit this oracle to be programmed. The random oracle responds to a fresh input μ with a random output $h \xleftarrow{\$} \mathbb{Z}_q$ and repeats a previous output when an input is repeated. Programming the oracle consists in assigning a given random output h to a given input μ. Collisions can potentially occur when programming: the input μ may already be associated with an output $h' \neq h$. Fortunately, we avoid collisions.

Consistency. Let $\mathcal{E}$ control a random tape of the algorithm $\mathcal{A}$ and be able to restart (rewind) the algorithm with the tape repeating. This is possible since $\mathcal{E}$ is allowed to manage the runtime environment of $\mathcal{A}$. Let $\mathcal{A}$ return a proof $(\mathbf{r}, s)$ with $s = (k - hu) \bmod q$ on the first run. On the second run, the random tape is repeated and, therefore, the word $\mathbf{r}$ as well as the input $(\mathbf{a}, \mathbf{a}', \mathbf{r})$ to the oracle H are also repeated. The output h' of H, chosen at random, differs from the first output h with probability $(q-1)/q$. Consequently, after $q/(q-1) = 1 + O(1/2^l)$ restarts on average $\mathcal{E}$ gets $h' \neq h$ and $s' = (k - h'u) \bmod q$. After that $\mathcal{E}$ determines

$$u = (s - s')(h' - h)^{-1} \bmod q.$$

We repeat here the standard arguments for Σ-protocols [7, 9].

Soundness. It is justified similarly to the consistency.

Blindness. The algorithm $\mathcal{S}_1$ generates $h, s \xleftarrow{\$} \mathbb{Z}_q$, constructs $\mathbf{r} \leftarrow s\,\mathbf{a} + h\,\mathsf{most}(\mathbf{a}')$ and programs H, that is, assigns the output h to the input $(\mathbf{a}, \mathbf{a}', \mathbf{r})$. The algorithm $\mathcal{S}_1$ returns a pair $(\mathbf{r}, s)$ as a proof α. This proof is accepted by $\mathsf{BAcc\text{-}DH.VfyAdd}$ and is statistically indistinguishable from the standard proof generated by $\mathsf{BAcc\text{-}DH.PrvAdd}$ provided that H is a random oracle.

The algorithm $\mathcal{S}_2$ is constructed similarly.

Unlinkability. Let us show that the algorithm $\mathcal{V}$ that participates in the game $\mathsf{G}(1^l, n, m)$ can be transformed into an algorithm that solves the SDDH (Square Decisional Diffie–Hellman) problem. This problem is proposed in [1] as a special case of the well-known DDH (Decisional Diffie–Hellman) problem.

The SDDH problem is specified with respect to a cyclic group $\mathbb{G}_q$ with a generator G and consists in deciding for a given triple (G, uG, vG), $u, v \in \mathbb{Z}_q^*$, if $v \equiv u^2 \pmod{q}$. The algorithm $\mathcal{B}$ that solves this problem guesses if this is indeed the case and outputs either 1 (true) or 0 (false). The distinguishing capabilities of $\mathcal{B}$ are characterized by the advantage

$$\mathbf{Adv}(\mathcal{B}) = \left| \mathsf{\mathbf{P}}\left\{\mathcal{B}(G, uG, u^2G) = 1 \colon u \xleftarrow{\$} \mathbb{Z}_q^*\right\} - \mathsf{\mathbf{P}}\left\{\mathcal{B}(G, uG, vG) = 1 \colon u, v \xleftarrow{\$} \mathbb{Z}_q^*\right\} \right|.$$

The probabilities here are over a random tape of $\mathcal{B}$ and over a random choice of u and v. Further we assume that G and the implicit accompanying description of $\mathbb{G}_q$ are valid outputs of $\mathsf{BAcc\text{-}DH.Init}$.

Let us show how $\mathcal{B}$ can solve SDDH by playing $\mathsf{G}(1^l, n, m)$ for the role of $\mathcal{V}$. Taking an instance (G, uG, vG) of SDDH, $\mathcal{B}$ acts as follows.

1. Publishes $\mathbf{a} = G$ as the initial accumulator (output of $\mathsf{BAcc\text{-}DH.Init}$).

2. Generates $j \xleftarrow{\$} \{1, 2, \ldots, m\}$.

3. Processes $\mathsf{BAcc\text{-}DH.Add}$ and $\mathsf{BAcc\text{-}DH.PrvAdd}$ calls made by $\mathcal{A}$ and determines used private keys. To do this, $\mathcal{B}$ restarts $\mathcal{A}$ several times and extracts private keys from the provided proofs acting as the algorithm $\mathcal{E}$ that justifies the consistency. It takes $m + O(m/2^l)$ restarts on average to determine all the keys.

4. Makes its own calls to $\mathsf{BAcc\text{-}DH.Add}$ (the order of calls is determined by $\mathcal{A}$) numbered $1, \ldots, j-1$, $j+1, \ldots, m$ using keys $u_1, \ldots, u_{j-1}, u_{j+1}, \ldots, u_m \xleftarrow{\$} \mathbb{Z}_q^*$ generated by itself. The calls are accompanied by proofs constructed using $\mathsf{BAcc\text{-}DH.PrvAdd}$.

5. Makes the jth call to BAcc-DH.Add in a non-standard way embedding the private key u hidden in the input (G, uG, vG). To do this, performs transitions $G_i \mapsto uG_i$, using the knowledge of $d_i = \log_G G_i$ and determining uG_i as $d_i(uG)$. The discrete logarithms d_i are indeed known to $\mathcal{B}$, since they are products of its own private keys and $\mathcal{A}$'s private keys extracted from the proofs.

6. Accompanies the jth call to BAcc-DH.Add with a proof of consistency indistinguishable from the real one and obtained by programming the oracle H. Here $\mathcal{B}$ acts as the algorithm $\mathcal{S}_1$ that justifies the blindness. Note that the inputs of H when constructing proofs of consistency at different steps of accumulator management are certainly different since the length of the accumulators as words increases. Therefore, there are no collisions when programming.

7. Processes the final accumulator $\mathbf{a} = G_0 G_1 \ldots G_n$ and generates public keys. The symbol G_0 has the form $G_0 = duG$, where d is the product of all private keys except u and this product is known to $\mathcal{B}$.

   The public keys V_i, $i \neq j$, are constructed using BAcc-DH.Der as $u_i G_0$. The public key V_j is constructed as $d(vG)$. This is the correct public key with $v \equiv u^2 \pmod{q}$ and a random public key with a random v. Let b be the indicator of the correctness of V_j. The bit b is unknown to $\mathcal{B}$ and is not used by it (unlike $\mathcal{V}$).

8. Passes $\mathcal{A}$ the public keys $(V_1, V_2, \ldots, V_m)$, waits the guess $\hat{b}$ and outputs it as its own guess to SDDH(G, uG, vG).

The algorithm $\mathcal{B}$ requires $m + O(m/2^l)$ restarts of $\mathcal{A}$ on average and additional time polynomial in l. Thus, if $\mathcal{A}$ is polynomial, then $\mathcal{B}$ is expected polynomial. At the same time,

$$\begin{aligned}\mathbf{Adv}(\mathcal{B}) &= |\mathbf{P}\{\mathcal{B} = 1 \mid b = 1\} - \mathbf{P}\{\mathcal{B} = 1 \mid b = 0\}| \\ &= \left|\mathbf{P}\left\{\hat{b} = 1 \mid b = 1\right\} - \mathbf{P}\left\{\hat{b} = 1 \mid b = 0\right\}\right| = \mathbf{Adv}(\mathcal{A}).\end{aligned}$$

This means that if SDDH is hard, i.e. $\mathbf{Adv}(\mathcal{A})$ is negligible, then $\mathbf{Adv}(\mathcal{B})$ is also negligible and the unlinkability is ensured.

It remains to say that the hardness of SDDH in a cryptographically strong cyclic group $\mathbb{G}_q$ is a reasonable assumption reflecting the similar assumption for DDH.

Acknowledgments: The author thanks Vlad Semenov and Valery Shchehlik for their helpful comments on the preliminary version of this paper.

References

[1] F. Bao, R. H. Deng, and H. Zhu. Variations of Diffie-Hellman problem. In: *Information and Communications Security. ICICS 2003.* Ed. by S. Qing, D. Gollmann, and J. Zhou. Vol. 435. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 301–312.

[2] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In: *Advances in Cryptology — EUROCRYPT '97.* Ed. by W. Fumy. Vol. 1233. Lecture Notes in Computer Science. Konstanz, Germany: Springer-Verlag, 1997, pp. 480–494.

[3] M. Bellare and O. Goldreich. On defining proofs of knowledge. In: *Advances in Cryptology — CRYPTO '92*. Ed. by E.F. Brickell. Vol. 740. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 1992, pp. 390–420.

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

[5] J. Benaloh and M. de Mare. *Efficient broadcast time-stamping. Technical Report 1 TR-MCS-91-1*. Tech. rep. 1991.

[6] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In: *Advances in Cryptology — CRYPTO 2002*. Ed. by M. Yung. Vol. 2442. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 61–76.

[7] R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis. Amsterdam: Universiteit van Amsterdam, 1997.

[8] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In: *Advances in Cryptology – CRYPTO '94*. Ed. by Y.G. Desmedt. Vol. 839. Lecture Notes in Computer Science. Springer, 1994, pp. 174–187.

[9] I. Damgard. *On Σ-protocols*. University of Aarhus, 2002. URL: `https://cs.au.dk/~ivan/Sigma.pdf`.

[10] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theor.* **31** (4) (1985), pp. 469–472.

[11] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18** (1) (1989), pp. 186–208.

[12] U. Maurer. Unifying zero-knowledge proofs of knowledge. In: *Progress in Cryptology*. Ed. by B. Preneel. Vol. 5580. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 272–286. URL: `https://doi.org/10.1007/978-3-642-02384-2_17`.

[13] C.P. Schnorr. Efficient identification and signatures for smart cards. In: *Advances in Cryptology*. Ed. by G. Brassard. Vol. 435. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1990, pp. 239–252. URL: `https://doi.org/10.1007/0-387-34805-0_22`.